Planemo Documentation

Release 0.75.22

Galaxy Project and Community

Apr 04, 2024

CONTENTS

1 Quick Start **3**
- 1.1 Obtaining . . . 3
- 1.2 Basics - Galaxy . . . 3
- 1.3 Basics - Common Workflow Language . . . 4
- 1.4 Tool Shed . . . 5
- 1.5 Conda . . . 5
- 1.6 Docker and Containers . . . 5

2 Installation **7**
- 2.1 pip . . . 7
- 2.2 Conda (Experimental) . . . 7

3 Configuration **9**

4 Virtual Appliance **11**
- 4.1 Quick Links . . . 11
- 4.2 The Setup . . . 11
- 4.3 Launching the Appliance . . . 12
- 4.4 Building the Appliance . . . 20

5 Building Galaxy Tools **21**
- 5.1 Building Galaxy Tools Using Planemo . . . 21
- 5.2 Building Galaxy Tools (Using the Planemo Appliance) . . . 36
- 5.3 Advanced Tool Development Topics . . . 53
- 5.4 How do I… . . . 102

6 Building Common Workflow Language Tools **109**
- 6.1 Building Common Workflow Language Tools Using Planemo . . . 109
- 6.2 Building Common Workflow Language Tools (Using the Planemo Appliance) . . . 117
- 6.3 Advanced Tool Development Topics . . . 126

7 Publishing to the Tool Shed **147**
- 7.1 Configuring a Shed Account . . . 147
- 7.2 Creating a Repository . . . 147
- 7.3 Updating a Repository . . . 148
- 7.4 Advanced Usage . . . 149

8 Test Format **151**
- 8.1 `job` . . . 152
- 8.2 `outputs` . . . 154
- 8.3 Engines for Testing . . . 157

8.4 Galaxy Testing Template . . . 159

9 Best Practices for Maintaining Galaxy Workflows 163
9.1 Workflow Structure . . . 163
9.2 Tests . . . 166
9.3 Publishing . . . 166

10 Running Galaxy workflows 169
10.1 The Basics . . . 169
10.2 Workflow execution against an external Galaxy . . . 171
10.3 Generating the job file . . . 171
10.4 Using workflow and dataset IDs . . . 173
10.5 Using aliases . . . 174
10.6 Checking invocations . . . 174
10.7 Profile configuration files . . . 174
10.8 Rerunning failed jobs . . . 175

11 Automating Galaxy Workflows 177

12 Autoupdating tools 179
12.1 Formatting tools . . . 179
12.2 Updating workflows . . . 180
12.3 Implementing an autoupdate CI job . . . 180

13 Commands 181
13.1 `autoupdate` command . . . 181
13.2 `ci_find_repos` command . . . 185
13.3 `ci_find_tools` command . . . 185
13.4 `ci_setup` command . . . 186
13.5 `clone` command . . . 187
13.6 `conda_build` command . . . 188
13.7 `conda_env` command . . . 188
13.8 `conda_init` command . . . 189
13.9 `conda_install` command . . . 189
13.10 `conda_search` command . . . 190
13.11 `config_init` command . . . 191
13.12 `container_register` command . . . 191
13.13 `create_alias` command . . . 192
13.14 `database_create` command . . . 192
13.15 `database_delete` command . . . 194
13.16 `database_list` command . . . 195
13.17 `delete_alias` command . . . 196
13.18 `docker_build` command . . . 197
13.19 `docker_shell` command . . . 198
13.20 `dockstore_init` command . . . 198
13.21 `docs` command . . . 199
13.22 `lint` command . . . 199
13.23 `list_alias` command . . . 200
13.24 `list_invocations` command . . . 200
13.25 `list_repos` command . . . 200
13.26 `merge_test_reports` command . . . 201
13.27 `mull` command . . . 201
13.28 `mulled_init` command . . . 202
13.29 `normalize` command . . . 203
13.30 `open` command . . . 203

13.31 `profile_create` command . . . 204
13.32 `profile_delete` command . . . 205
13.33 `profile_list` command . . . 206
13.34 `project_init` command . . . 206
13.35 `pull_request` command . . . 206
13.36 `rerun` command . . . 207
13.37 `run` command . . . 208
13.38 `serve` command . . . 212
13.39 `share_test` command . . . 216
13.40 `shed_build` command . . . 217
13.41 `shed_create` command . . . 217
13.42 `shed_diff` command . . . 218
13.43 `shed_init` command . . . 219
13.44 `shed_lint` command . . . 220
13.45 `shed_serve` command . . . 221
13.46 `shed_test` command . . . 225
13.47 `shed_update` command . . . 229
13.48 `shed_upload` command . . . 230
13.49 `syntax` command . . . 231
13.50 `test` command . . . 232
13.51 `test_reports` command . . . 236
13.52 `tool_init` command . . . 237
13.53 `training_fill_data_library` command . . . 238
13.54 `training_generate_from_wf` command . . . 238
13.55 `training_init` command . . . 242
13.56 `upload_data` command . . . 246
13.57 `workflow_convert` command . . . 249
13.58 `workflow_edit` command . . . 252
13.59 `workflow_job_init` command . . . 255
13.60 `workflow_lint` command . . . 256
13.61 `workflow_test_init` command . . . 257
13.62 `workflow_test_on_invocation` command . . . 257
13.63 `workflow_upload` command . . . 258

14 FAQs 259
14.1 What is Planemo? . . . 259
14.2 Is Planemo the right tool for me? . . . 259
14.3 What is the difference between Galaxy and CWL? Which should I use? . . . 259
14.4 Can I develop Galaxy tools without Planemo? . . . 260
14.5 Where can I learn to write tools? . . . 260
14.6 How do I set up deployment to the ToolShed with Planemo? . . . 260
14.7 How can I automate workflow execution with Planemo? . . . 260
14.8 How can I contribute to the project? . . . 260

15 Galaxy Tool Framework Changes 261
15.1 John Chilton . . . 261

16 Code of Conduct 269

17 Contributing 271
17.1 Types of Contributions . . . 271
17.2 Get Started! . . . 272
17.3 Pull Request Guidelines . . . 273
17.4 Tips . . . 273

18 Project Governance **275**
18.1 Benevolent Dictator for Now (BDFN) 275
18.2 Committers 275

19 Release Checklist **277**

20 History **279**
20.1 0.75.22 (2024-04-04) 279
20.2 0.75.21 (2024-02-01) 279
20.3 0.75.20 (2024-01-30) 279
20.4 0.75.19 (2023-12-06) 279
20.5 0.75.18 (2023-11-16) 280
20.6 0.75.17 (2023-11-01) 280
20.7 0.75.16 (2023-11-01) 280
20.8 0.75.15 (2023-10-29) 280
20.9 0.75.14 (2023-10-19) 280
20.10 0.75.13 (2023-10-18) 280
20.11 0.75.12 (2023-09-18) 281
20.12 0.75.11 (2023-09-14) 281
20.13 0.75.10 (2023-09-01) 281
20.14 0.75.9 (2023-06-29) 281
20.15 0.75.8 (2023-06-09) 281
20.16 0.75.7 (2023-03-01) 282
20.17 0.75.6 (2023-02-22) 282
20.18 0.75.5 (2023-02-10) 282
20.19 0.75.4 (2023-02-09) 282
20.20 0.75.3 (2022-11-30) 282
20.21 0.75.2 (2022-11-02) 283
20.22 0.75.1 (2022-10-31) 283
20.23 0.75.0 (2022-10-28) 283
20.24 0.74.11 (2022-06-08) 284
20.25 0.74.10 (2022-05-31) 284
20.26 0.74.9 (2021-11-03) 285
20.27 0.74.8 (2021-10-10) 285
20.28 0.74.7 (2021-09-21) 285
20.29 0.74.6 (2021-07-23) 286
20.30 0.74.5 (2021-06-25) 286
20.31 0.74.4 (2021-06-01) 286
20.32 0.74.3 (2021-02-25) 286
20.33 0.74.2 (2021-02-21) 287
20.34 0.74.1 (2021-01-03) 287
20.35 0.74.0 (2020-12-30) 287
20.36 0.73.0 (2020-12-28) 287
20.37 0.72.0 (2020-08-04) 288
20.38 0.71.0 (2020-08-03) 288
20.39 0.70.0 (2020-01-29) 289
20.40 0.62.1 (2019-10-14) 290
20.41 0.62.0 (2019-10-11) 290
20.42 0.61.0 (2019-07-08) 290
20.43 0.60.0 (2019-05-31) 290
20.44 0.59.0 (2019-05-09) 290
20.45 0.58.2 (2019-03-01) 291
20.46 0.58.1 (2019-01-03) 291
20.47 0.58.0 (2019-01-01) 291

20.48 0.57.1 (2018-11-23) 291
20.49 0.57.0 (2018-11-19) 291
20.50 0.56.0 (2018-10-30) 292
20.51 0.55.0 (2018-09-12) 292
20.52 0.54.0 (2018-06-06) 292
20.53 0.53.0 (2018-05-22) 293
20.54 0.52.0 (2018-05-20) 293
20.55 0.51.0 (2018-05-19) 293
20.56 0.50.1 (2018-05-11) 294
20.57 0.50.0 (2018-05-10) 294
20.58 0.49.2 (2018-05-09) 294
20.59 0.49.1 (2018-05-06) 294
20.60 0.49.0 (2018-05-06) 294
20.61 0.48.0 (2018-02-28) 295
20.62 0.47.0 (2017-11-18) 295
20.63 0.46.1 (2017-09-26) 295
20.64 0.46.0 (2017-09-15) 295
20.65 0.45.0 (2017-09-06) 296
20.66 0.44.0 (2017-06-22) 296
20.67 0.43.0 (2017-06-22) 296
20.68 0.42.1 (2017-06-16) 296
20.69 0.42.0 (2017-06-15) 296
20.70 0.41.0 (2017-06-05) 297
20.71 0.40.1 (2017-05-03) 297
20.72 0.40.0 (2017-03-16) 297
20.73 0.39.0 (2017-03-15) 297
20.74 0.38.1 (2017-02-06) 298
20.75 0.38.0 (2017-02-06) 298
20.76 0.37.0 (2017-01-25) 299
20.77 0.36.1 (2016-12-12) 299
20.78 0.36.0 (2016-12-11) 299
20.79 0.35.0 (2016-11-14) 299
20.80 0.34.1 (2016-10-12) 299
20.81 0.34.0 (2016-10-05) 300
20.82 0.33.2 (2016-09-28) 300
20.83 0.33.1 (2016-09-28) 300
20.84 0.33.0 (2016-09-23) 300
20.85 0.32.0 (2016-09-16) 301
20.86 0.31.0 (2016-09-06) 301
20.87 0.30.2 (2016-09-01) 301
20.88 0.30.1 (2016-09-01) 301
20.89 0.30.0 (2016-09-01) 301
20.90 0.29.1 (2016-08-19) 302
20.91 0.29.0 (2016-08-19) 302
20.92 0.28.0 (2016-08-17) 302
20.93 0.27.0 (2016-06-22) 303
20.94 0.26.0 (2016-05-20) 303
20.95 0.25.1 (2016-05-11) 303
20.96 0.25.0 (2016-05-11) 303
20.97 0.24.2 (2016-04-25) 304
20.98 0.24.1 (2016-04-08) 304
20.99 0.24.0 (2016-03-29) 304
20.100 0.23.0 (2016-02-15) 305
20.101 0.22.2 (2016-01-14) 305

20.102 0.22.1 (2016-01-14) . . . 305
20.103 0.22.0 (2016-01-13) . . . 305
20.104 0.21.1 (2015-11-29) . . . 305
20.105 0.21.0 (2015-11-29) . . . 306
20.106 0.20.0 (2015-11-11) . . . 306
20.107 0.19.0 (2015-11-03) . . . 307
20.108 0.18.1 (2015-10-22) . . . 307
20.109 0.18.0 (2015-10-20) . . . 307
20.110 0.17.0 (2015-10-19) . . . 307
20.111 0.16.0 (2015-10-07) . . . 308
20.112 0.15.0 (2015-10-01) . . . 308
20.113 0.14.0 (2015-08-06) . . . 308
20.114 0.13.2 (2015-07-06) . . . 309
20.115 0.13.1 (2015-07-01) . . . 309
20.116 0.13.0 (2015-06-28) . . . 309
20.117 0.12.2 (2015-05-23) . . . 309
20.118 0.12.1 (2015-05-21) . . . 309
20.119 0.12.0 (2015-05-21) . . . 310
20.120 0.11.1 (2015-05-12) . . . 310
20.121 0.11.0 (2015-05-12) . . . 310
20.122 0.10.0 (2015-05-06) . . . 311
20.123 0.9.0 (2015-05-03) . . . 311
20.124 0.8.4 (2015-04-30) . . . 311
20.125 0.8.3 (2015-04-29) . . . 312
20.126 0.8.2 (2015-04-29) . . . 312
20.127 0.8.1 (2015-04-28) . . . 312
20.128 0.8.0 (2015-04-27) . . . 312
20.129 0.7.0 (2015-04-13) . . . 313
20.130 0.6.0 (2015-03-16) . . . 313
20.131 0.5.0 (2015-02-22) . . . 314
20.132 0.4.2 (2015-02-21) . . . 314
20.133 0.4.1 (2015-02-16) . . . 314
20.134 0.4.0 (2015-02-16) . . . 314
20.135 0.3.1 (2015-02-15) . . . 314
20.136 0.3.0 (2015-02-13) . . . 315
20.137 0.2.0 (2015-01-13) . . . 315
20.138 0.1.0 (2014-12-16) . . . 315
20.139 0.0.1 (2014-10-04) . . . 316

21 Indices and tables **317**

Contents:

Command-line utilities to assist in developing Galaxy and Common Workflow Language artifacts - including tools, workflows, and training materials.

- Free software: MIT License
- Documentation: https://planemo.readthedocs.io.
- Code: https://github.com/galaxyproject/planemo

CHAPTER

ONE

QUICK START

1.1 Obtaining

For a traditional Python installation of Planemo, first set up a virtualenv for `planemo` (this example creates a new one in `.venv`) containing Python 3.7 or newer and then install with `pip`. Planemo must be installed with pip 7.0 or newer.

```
$ virtualenv .venv; . .venv/bin/activate
$ pip install "pip>=7" # Upgrade pip if needed.
$ pip install planemo
```

For information on updating Planemo, installing the latest development release, or installing Planemo via Bioconda - checkout the installation documentation.

Planemo is also available as a virtual appliance bundled with a preconfigured Galaxy server and set up for Galaxy and Common Workflow Language tool development. You can choose from open virtualization format (OVA, .ova) or Docker appliances.

1.2 Basics - Galaxy

This quick start will assume you have a directory with one or more Galaxy tool XML files. If no such directory is available, one can be quickly created for demonstrating `planemo` as follows `project_init --template=demo mytools; cd mytools`.

Planemo can check tool XML files for common problems and best practices using the `lint` command (also aliased as `l`).

```
$ planemo lint
```

Like many `planemo` commands - by default this will search the current directory and use all tool files it finds. It can be explicitly passed a path to tool files or a directory of tool files.

```
$ planemo l randomlines.xml
```

The `lint` command takes in additional options related to reporting levels, exit code, etc. These options are described in the docs or (like with all commands) can be accessed by passing `--help` to it.

```
$ planemo l --help
Usage: planemo lint [OPTIONS] TOOL_PATH
```

Once tools are syntactically correct - it is time to test. The `test` command can be used to test a tool or a directory of tools.

```
$ planemo test --galaxy_root=../galaxy randomlines.xml
```

If no `--galaxy_root` is defined, Planemo will download and configure a disposable Galaxy instance for testing.

Planemo will create a HTML output report in the current directory named `tool_test_output.html` (override with `--test_output`). See an example of such a report for Tophat.

Once tools have been linted and tested - the tools can be viewed in a Galaxy interface using the `serve` (`s`) command.

```
$ planemo serve
```

Like `test`, `serve` requires a Galaxy root and one can be explicitly specified with `--galaxy_root` or installed dynamically with `--install_galaxy`.

For more information on building Galaxy tools in general please check out Building Galaxy Tools Using Planemo.

For more information on developing Galaxy workflows with Planemo checkout best practices for Galaxy Workflows and the description of Planemo's test format. For information on developing Galaxy training materials checkout the contributing documentation on training.galaxyproject.org.

1.3 Basics - Common Workflow Language

This quick start will assume you have a directory with one or more Common Workflow Language YAML files. If no such directory is available, one can be quickly created for demonstrating `planemo` as follows `planemo project_init --template=seqtk_complete_cwl mytools; cd mytools`.

Planemo can check tools YAML files for common problems and best practices using the `lint` command (also aliased as `l`).

```
$ planemo lint
```

Like many `planemo` commands - by default this will search the current directory and use all tool files it finds. It can be explicitly passed a path to tool files or a directory of tool files.

```
$ planemo l seqtk_seq.cwl
```

The `lint` command takes in additional options related to reporting levels, exit code, etc. These options are described in the docs or (like with all commands) can be accessed by passing `--help` to it.

```
$ planemo l --help
Usage: planemo lint [OPTIONS] TOOL_PATH
```

Once tools are syntactically correct - it is time to test. The `test` command can be used to test a CWL tool, workflow, or a directories thereof.

```
$ planemo test --engine cwltool seqtk_seq.cwl
```

Planemo will create a HTML output report in the current directory named `tool_test_output.html`. Check out the file `seqtk_seq_tests.yml` for an example of Planemo test for a CWL tool. A test consists of any number of jobs (with input descriptions) and corresponding output assertions.

Checkout the Commmon Workflow User Guide for more information on developing CWL tools in general and Building Common Workflow Language Tools for more information on using Planemo to develop CWL tools.

1.4 Tool Shed

Planemo can help you publish tools to the Galaxy Tool Shed. Check out Publishing to the Tool Shed for more information.

1.5 Conda

Planemo can help develop tools and Conda packages in unison. Check out the Galaxy or CWL version of the "Dependencies and Conda" tutorial for more information.

1.6 Docker and Containers

Planemo can help develop tools that run in "Best Practice" containers for scientific workflows. Check out the Galaxy or CWL version of the "Dependencies and Containers" tutorial for more information.

CHAPTER

TWO

INSTALLATION

2.1 pip

For a traditional Python installation of Planemo, first set up a virtual environment for `planemo` (this example creates a new one in `planemo`) and then install with `pip`. Planemo requires pip 7.0 or newer.

```
$ python3 -m venv planemo
$ . planemo/bin/activate
$ pip install planemo
```

When installed this way, planemo can be upgraded as follows:

```
$ . planemo/bin/activate
$ pip install -U planemo
```

To install or update to the latest development branch of planemo with `pip`, use the following `pip install` idiom instead:

```
$ pip install -U git+git://github.com/galaxyproject/planemo.git
```

If your `PATH` contains a Python installed through Conda it should likely not be used to run Planemo, consider using the `virtualenv` argument `-p` to point at a non-Conda Python 2 executable installed natively on your system or using a tool such pyenv. `virtualenv` can be installed via Conda, pyenv, or a package manager - it should make no difference.

Planemo runs on Python 3.7 or newer. Planemo can be used to run multiple versions of Galaxy, but please note that the last Galaxy release that fully supports Python 2.7 is 19.09.

2.2 Conda (Experimental)

Another approach for installing Planemo is to use Conda (most easily obtained via the Miniconda Python distribution). Afterwards run the following commands.

```
$ conda config --add channels bioconda
$ conda config --add channels conda-forge
$ conda install planemo
```

Galaxy is known to have issues when running with a Conda Python so this approach should be considered experimental for now. If you have problems with it or hacks to make it work better - please report them.

Note: The options `--docker` and `--biocontainers` for `planemo serve` and `planemo test` require that docker is installed on the system and that it can be run without root privileges. See Docker installation and run Docker without root privileges for further instructions.

CHAPTER THREE

CONFIGURATION

See the `--help` or the documentation for a detailed list of individual options for each command. Certain options that make sense for multiple commands or tools suites can be declared instead globally by placing them in a `.planemo.yml` file in your home directory.

Below is an annotated outline of some of the options that can be set in this file.

```
## Specify a default galaxy_root for the `test` and `serve` commands here.
#galaxy_root: /home/user/galaxy

## Specify github credentials for publishing gist links (e.g. with
## the `share_test` command).
#github:
#  username: <username>
#  password: <password>

sheds:
  # For each Tool Shed you wish to target enter the API key
  # or both email and password.
  toolshed:
    key: "<TODO>"
    #email: "<TODO>"
    #password: "<TODO>"
  testtoolshed:
    key: "<TODO>"
    #email: "<TODO>"
    #password: "<TODO>"
  local:
    key: "<TODO>"
    #email: "<TODO>"
    #password: "<TODO>"
  custom_shed:
    key: "<TODO>"
    url: "http://customurl/"
    #email: "<TODO>"
    #password: "<TODO>"
```

CHAPTER

FOUR

VIRTUAL APPLIANCE

You can use Planemo as part of a Galaxy tool development virtual appliance which comes pre-configured with Planemo, Galaxy, Docker, Conda, a local Tool Shed, linuxbrew, Komodo and Atom editors.

4.1 Quick Links

If you already know what to do. Otherwise please read on.

- Planemo OVA image: https://images.galaxyproject.org/planemo/latest.ova
- `docker run -p 8010:80 -p 9009:9009 -v `pwd`:/opt/galaxy/tools -i -t planemo/interactive` This assumes your tools are in your current working directory (replace *`pwd`* with a path to your tools if this is not the case).

4.2 The Setup

The Galaxy instance that runs in these appliances has been optimized for tool development - Galaxy will monitor your tool directory for changes and reload the tools as they are modified, the server will directly log you into Galaxy as an admin (no need to worry about user management or configuration), Galaxy is configured to use a PostgreSQL database, and execute jobs via SLURM for robustness. If something goes wrong and Galaxy needs to be restarted manually - run `restart_galaxy` from the command-line.

The virtual appliance is available in four flavors via open virtualization format (OVA, .ova), Docker, Vagrant, and as a Google Compute Engine cloud image.

The OVA image is a stable way to boot a Planemo virtual machine on any platform and comes with a pre-configured Xubuntu-based windowed operating system with graphical editing tools including Komodo and Atom editors. This approach can be thought of more as a complete environment and may be better for tutorials and workshops where consistent user experience is more important.

The Docker and Vagrant versions make it trivial to mount an external directory in the appliance so that one can use their own development tools (such as editors). These can be used in traditional development environments with existing tools and will probably be the preference of power users whereas.

The Google Compute Engine variant is ideal when local compute resources are unavailable or insufficient.

4.3 Launching the Appliance

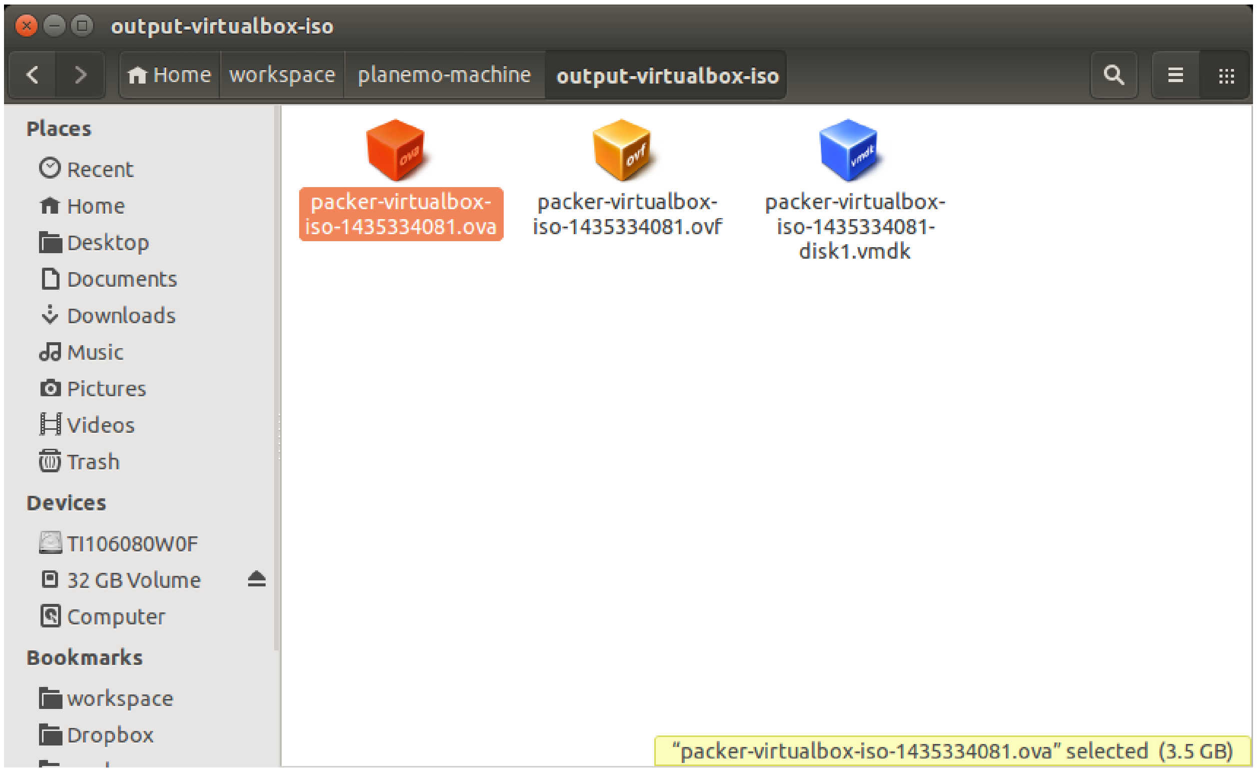
The following sections will describe how to launch the appliance using various platforms.

4.3.1 Launching the Appliance (VirtualBox - OVA)

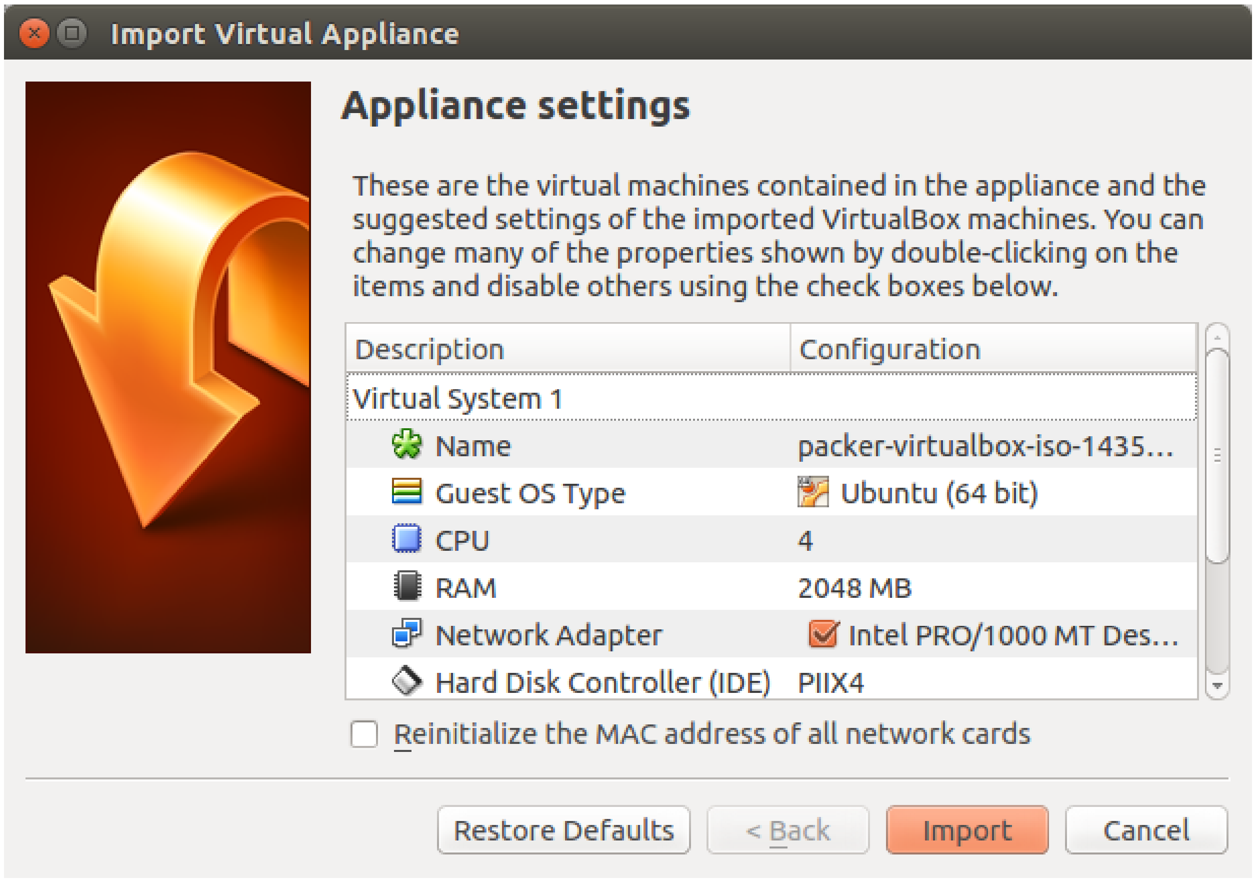
The VirtualBox OVA variant of the Planemo appliance comes preconfigured with a full windowed development environment (based on Xubuntu). Encompassing the complete environment means it is easier to setup and provides an identical experience for every developer using it. These make the OVA image ideal for tutorials and workshops.

The latest VirtualBox version of the planemo appliance can be found here.

Please download and install VirtualBox. When VirtualBox has been installed - the planemo machine can be imported by downloading the latest image and double clicking the resulting file. This should result in VirtualBox being opened to an import screen. Just follow the prompt and the machine should become available.



"packer-virtualbox-iso-1435334081.ova" selected (3.5 GB)



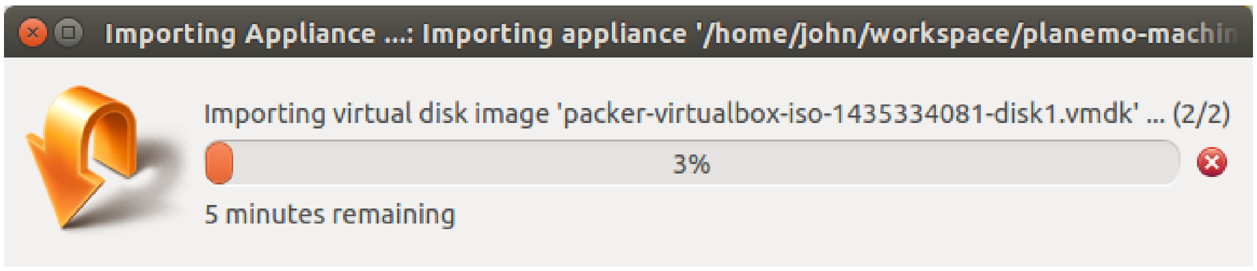
Import Virtual Appliance

Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Description	Configuration
Virtual System 1	
Name	packer-virtualbox-iso-1435...
Guest OS Type	Ubuntu (64 bit)
CPU	4
RAM	2048 MB
Network Adapter	✓ Intel PRO/1000 MT Des...
Hard Disk Controller (IDE)	PIIX4

☐ Reinitialize the MAC address of all network cards



Importing Appliance ...: Importing appliance '/home/john/workspace/planemo-machin

Importing virtual disk image 'packer-virtualbox-iso-1435334081-disk1.vmdk' ... (2/2)

3%

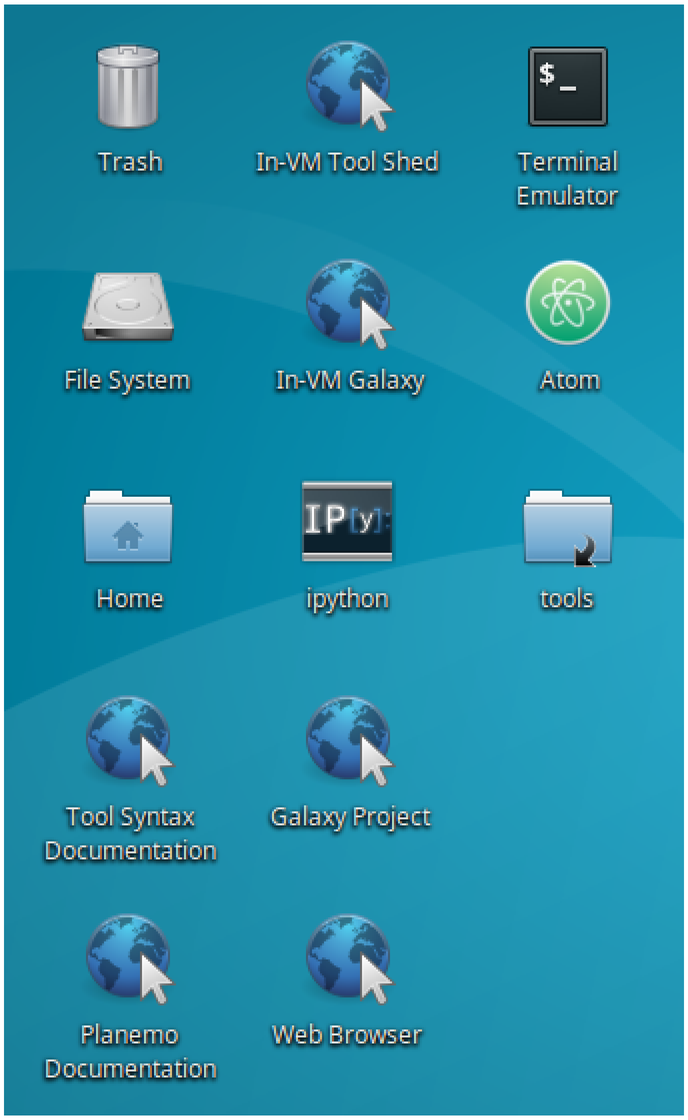
5 minutes remaining

When the import is finished (and before starting the VM), right-click on the new appliance and select "Settings":

1. in the "General" -> "Advanced" tab, select "Bidirectional" for "Shared Clipboard"
2. in the "Display" -> "Screen" tab, tick "Enable 3D Acceleration"
3. in the "Network" -> "Adapter 1" tab, select "PCnet-FAST III (Am79c973)" as "Adapter Type"
4. Click the "OK" button

Now start the appliance by clicking the "Start" button.

The Firefox browser, Komodo and Atom editors, Galaxy, Planemo and everything else is available right away on the desktop along with useful links.



In-VM Galaxy runs at http://localhost and the In-VM Tool Shed at http://localhost:9009.

Various relevant applications are available under the Xubuntu menu which has the following icon.

4.3.2 Launching the Appliance (Docker)

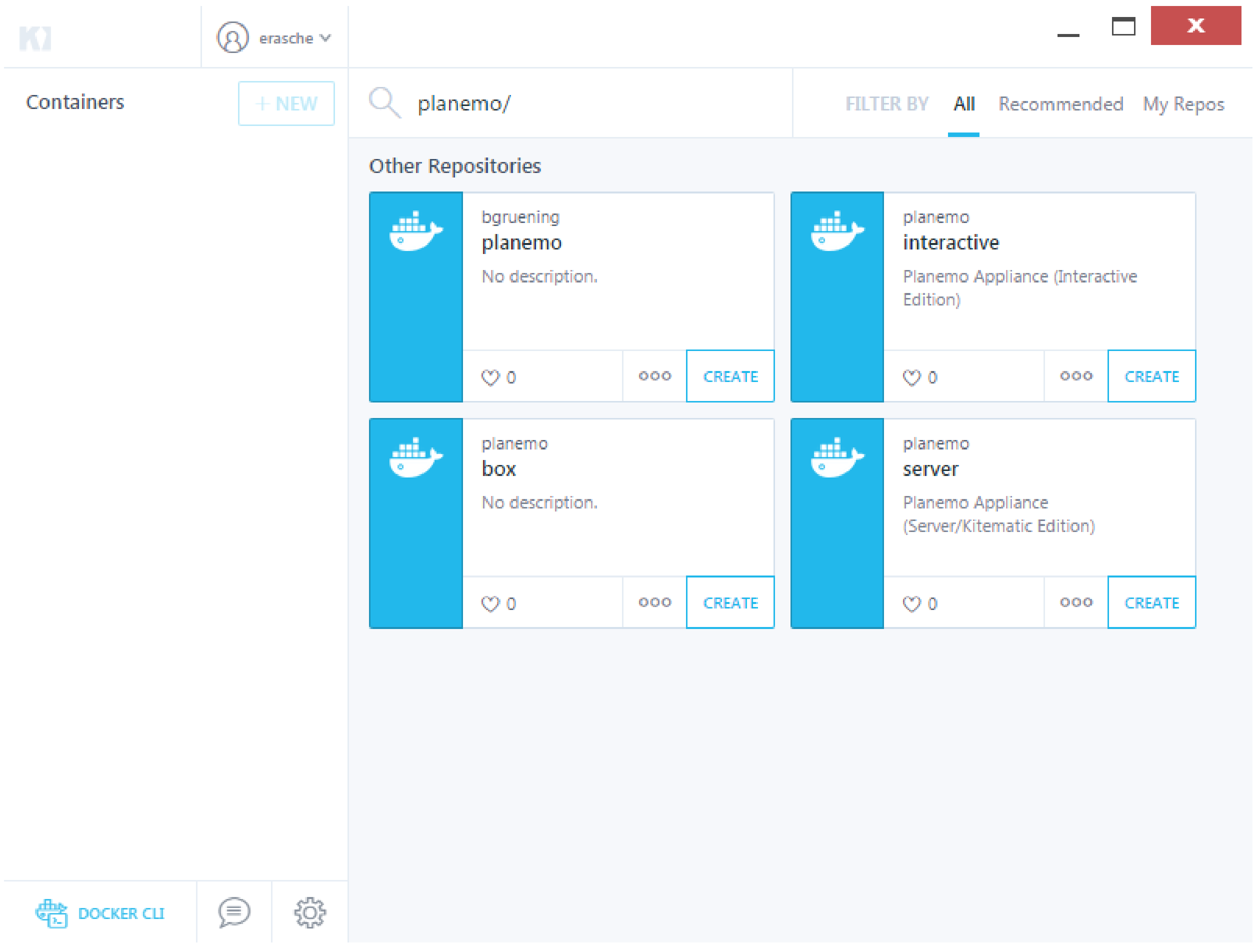
There are two variants of the Docker appliance - one is specifically designed for Kitematic a GUI application available for Mac OS X and Windows that claims to be "the easiest way to start using Docker" and the other is designed to be used with a command-prompt available under Linux or in Mac OS X and Windows when using boot2docker.

Server Edition (for Kitematic)

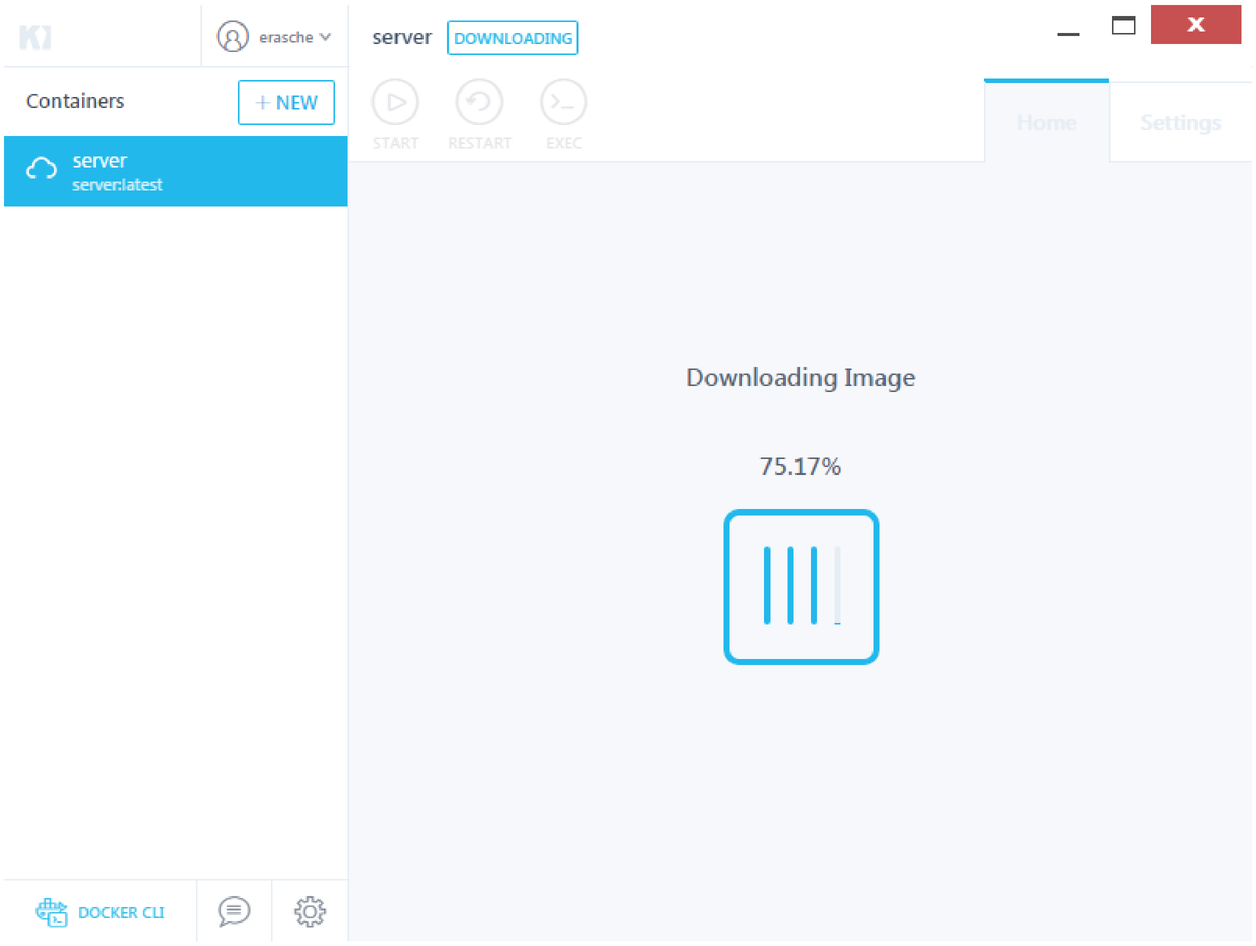
To get started with Kitematic, please download it if it hasn't been previously installed and then launch it.

Wait for Kitematic to load and search for *planemo/server*.

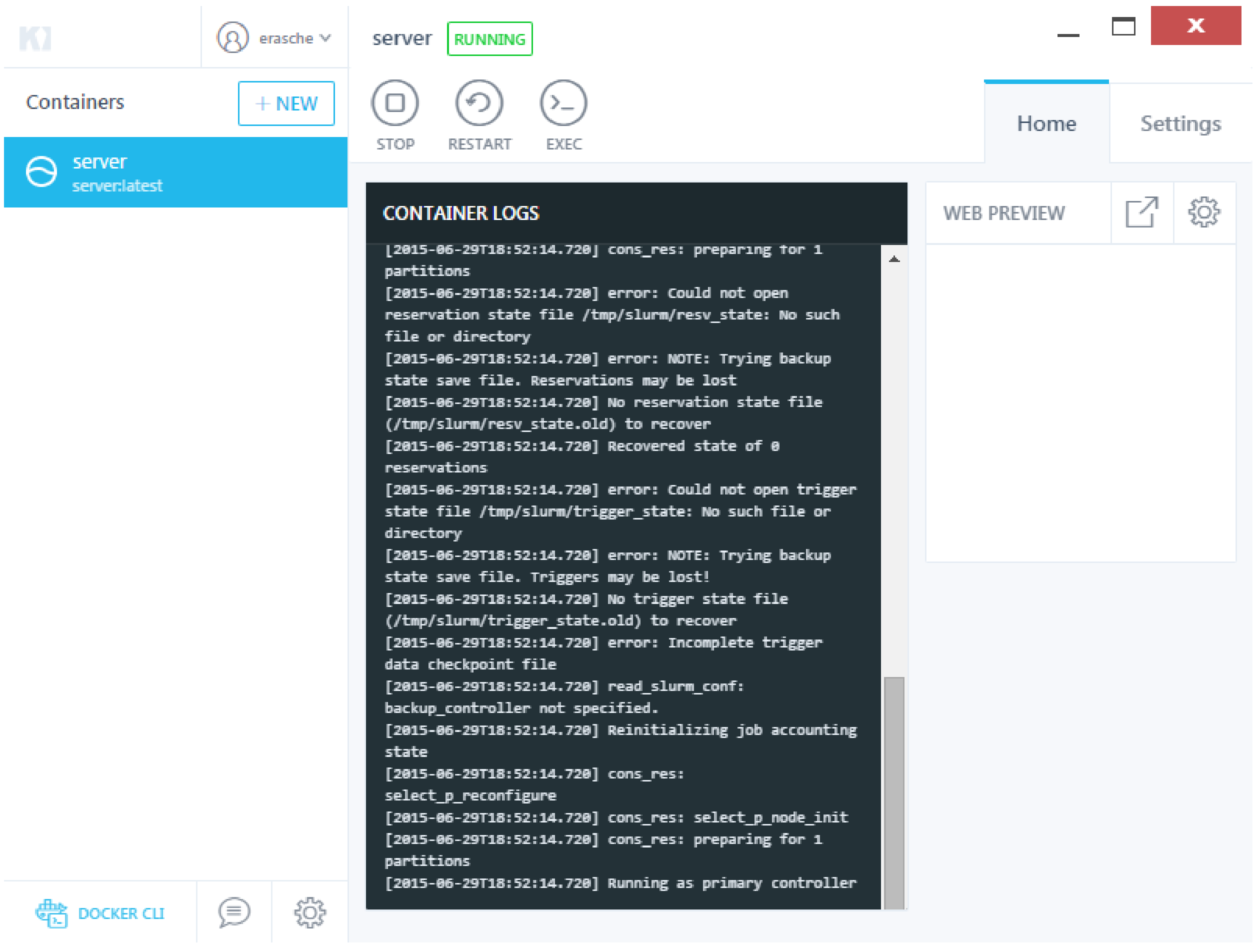
Once Kitematic has downloaded, you can use the search bar at the top to locate *planemo/server*



There may be several Planemo containers discovered - be sure to pick the *planemo/server* one with the experience optimized for Kitematic. Choose to create this image and it will download.



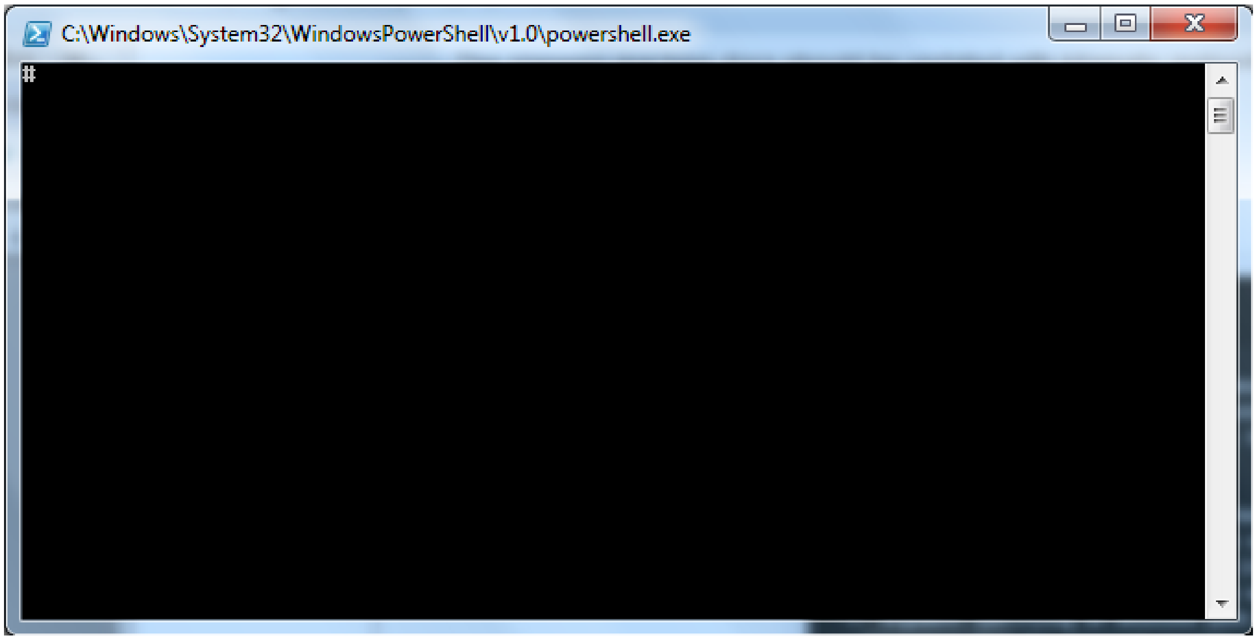
After a minute or so, you should see logs for the running container appear in the main window.



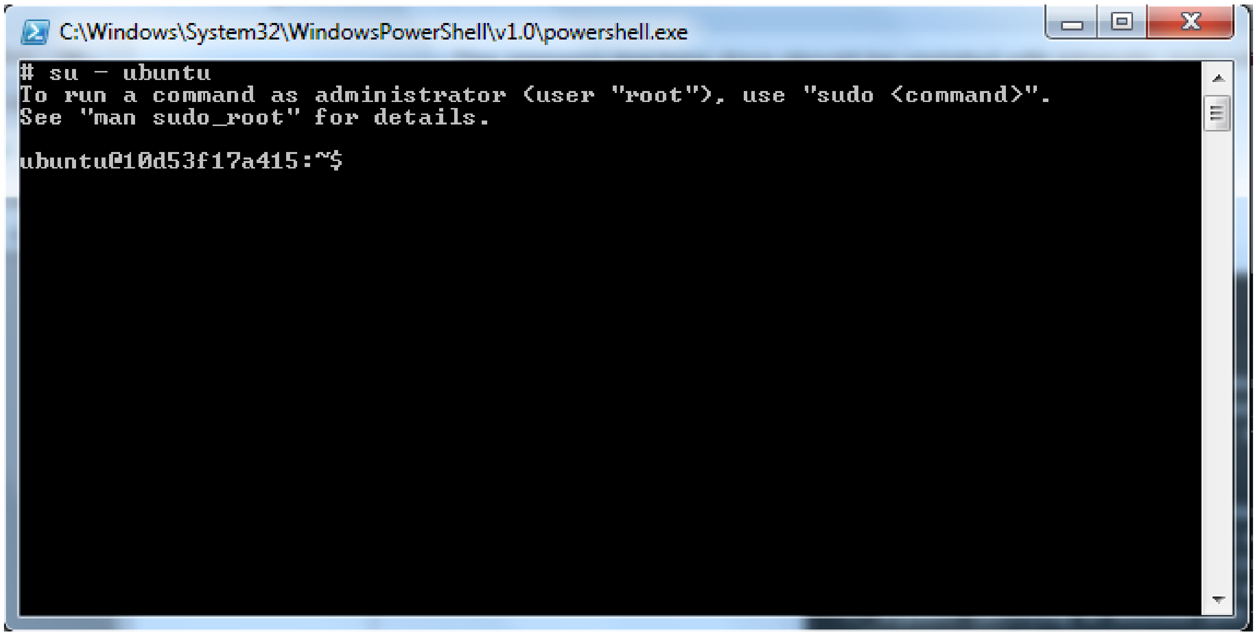
```
[2015-06-29T18:52:14.720] cons_res: preparing for 1
partitions
[2015-06-29T18:52:14.720] error: Could not open
reservation state file /tmp/slurm/resv_state: No such
file or directory
[2015-06-29T18:52:14.720] error: NOTE: Trying backup
state save file. Reservations may be lost
[2015-06-29T18:52:14.720] No reservation state file
(/tmp/slurm/resv_state.old) to recover
[2015-06-29T18:52:14.720] Recovered state of 0
reservations
[2015-06-29T18:52:14.720] error: Could not open trigger
state file /tmp/slurm/trigger_state: No such file or
directory
[2015-06-29T18:52:14.720] error: NOTE: Trying backup
state save file. Triggers may be lost!
[2015-06-29T18:52:14.720] No trigger state file
(/tmp/slurm/trigger_state.old) to recover
[2015-06-29T18:52:14.720] error: Incomplete trigger
data checkpoint file
[2015-06-29T18:52:14.720] read_slurm_conf:
backup_controller not specified.
[2015-06-29T18:52:14.720] Reinitializing job accounting
state
[2015-06-29T18:52:14.720] cons_res:
select_p_reconfigure
[2015-06-29T18:52:14.720] cons_res: select_p_node_init
[2015-06-29T18:52:14.720] cons_res: preparing for 1
partitions
[2015-06-29T18:52:14.720] Running as primary controller
```

Galaxy will now be available by clicking the link in the *Web Preview* section of the GUI.

Clicking the *Exec* button in the container's tool bar (at the top, middle of the screen) will launch a root command-prompt. Planemo is configured for the `ubuntu` user - so the first thing you should do is launch an `ubuntu` login session by entering the command `su - ubuntu`.



```
#
```



```
# su - ubuntu
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@10d53f17a415:~$
```

Interactive Edition

The interactive edition of the Planemo Docker image is designed for environments where the `docker` command-line tool is available. This can easily be installed via package managers under Linux - but for Windows and Mac OS X - boot2docker should be installed and launched in order to run these commands.

The Docker version of the planemo appliance can be launched using the following command (which will pull the appliance down from Docker Hub).

```
$ docker run -p 8010:80 -p 9009:9009 -v `pwd`:/opt/galaxy/tools -i -t planemo/interactive
```

This command will start Galaxy and various other services and then open a bash shell with Planemo available. This assumes your tools are in your current working directory (just replace *`pwd`* with a path to your tools if this is not the case).

Docker commands such as `ps` and `kill` can be used to manage this Docker container.

This Docker environment will contain your tools and modifications made to them will be made directly to your filesystem - so they are persistent. Data loaded into the Galaxy instance (history data for instance) will be lost when the Docker container is stopped. Check out the docker-galaxy-stable project for information on running persistent Galaxy processes in Docker.

4.3.3 Launching the Appliance (Vagrant)

The image for this way of launching the appliance is outdated. Please use a different one.

The latest Vagrant version of the planemo appliance can be found here. Once you have installed Vagrant (download now), the appliance can be enabled by first creating a `Vagrantfile` in your tool directory - the following demonstrates an example of such file.

```
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "planemo"
  config.vm.box_url = "https://images.galaxyproject.org/planemo/latest.box"
  config.ssh.username = "ubuntu"

  # Forward nginx and tool shed.
  config.vm.network "forwarded_port", guest: 80, host: 8010
  config.vm.network "forwarded_port", guest: 9009, host: 9009

  # Disable default mount and mount pwd to /opt/galaxy/tools instead.
  config.vm.synced_folder ".", "/vagrant", disabled: true
  config.vm.synced_folder ".", "/opt/galaxy/tools", owner: "ubuntu"

  # config.vm.provider "virtualbox" do |vb|
  #   # Don't boot with headless mode
  #   vb.gui = true
  #
  #   # Use VBoxManage to customize the VM. For example to change memory:
  #   vb.customize ["modifyvm", :id, "--memory", "1024"]
  # end
end
```

This file must literally be named `Vagrantfile`. Next you will need to startup the appliance. This is as easy as

```
$ vagrant up
```

Once the virtual server has booted up completely, Galaxy will be available at http://localhost:8010, the Codebox IDE will be available http://localhost:8010/ide/, and the local Tool Shed at http://localhost:9009.

4.3.4 Launching the Appliance (Google Compute Engine)

The image for this way of launching the appliance is outdated. Please use a different one.

The GCE version of the appliance is different in that it doesn't run locally on your computer, but on a remote 'cloud' machine. Using this variant of the appliance requires a Google Cloud Platform account with an active payment method.

The first thing you'll want to do is get the gcloud administration utility installed and configured. Once you've installed gcloud, you can authenticate and (optionally) set your default project, zone, and region (example below, but you should choose whatever region and zone are appropriate for your location). If you set these defaults, you will not have to supply them to all subsequent commands.

```
$ gcloud auth login
$ gcloud config set project YOUR-PROJECT-NAME
$ gcloud config set compute/region us-central1    (replace us-central1 with another␣
↪region if desired)
$ gcloud config set compute/zone us-central1-f    (same for the zone us-central1-f)
```

Import the image to your account with the following statement. This will only need to be done one time, unless you delete the image from your account.

```
$ gcloud compute images create planemo-machine --source-uri=http://storage.googleapis.
↪com/galaxyproject_images/planemo_machine.image.tar.gz
```

To launch the image as a fresh instance, use the following command. This command will, upon completion, display an external ip address that you can navigate to in your web browser.

```
$ gcloud compute instances create planemo --machine-type n1-standard-2 --image planemo-
↪machine --tags http-server
```

If you'd like to SSH in to the instance at this point, it's easy to do with:

```
$ gcloud compute ssh planemo
```

4.4 Building the Appliance

These appliances are built using the planemo-machine project which can be used to build customized recipes of this nature or even appliance for cloud environments such as Amazon Web Services and Google Compute Engine.

CHAPTER

FIVE

BUILDING GALAXY TOOLS

The following links are for the same tutorial describing the basics of how to build Galaxy tools. The first variant is tailored to local development environments (e.g. if Planemo has been installed with `brew` or `pip`) and the second is for developers using a dedicated Planemo virtual appliance (available as OVA, Docker, Vagrant, etc.).

5.1 Building Galaxy Tools Using Planemo

This tutorial is a gentle introduction to writing Galaxy tools using Planemo. Please read the installation instructions for Planemo if you have not already installed it.

5.1.1 The Basics

This guide is going to demonstrate building up tools for commands from Heng Li's Seqtk package - a package for processing sequence data in FASTA and FASTQ files.

To get started let's install Seqtk. Here we are going to use `conda` to install Seqtk - but however you obtain it should be fine.

```
$ conda install --force --yes -c conda-forge -c bioconda seqtk=1.2
    ... seqtk installation ...
$ seqtk seq
        Usage:   seqtk seq [options] <in.fq>|<in.fa>
        Options: -q INT    mask bases with quality lower than INT [0]
                 -X INT    mask bases with quality higher than INT [255]
                 -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
                 -l INT    number of residues per line; 0 for 2^32-1 [0]
                 -Q INT    quality shift: ASCII-INT gives base quality [33]
                 -s INT    random seed (effective with -f) [11]
                 -f FLOAT  sample FLOAT fraction of sequences [1]
                 -M FILE   mask regions in BED or name list FILE [null]
                 -L INT    drop sequences with length shorter than INT [0]
                 -c        mask complement region (effective with -M)
                 -r        reverse complement
                 -A        force FASTA output (discard quality)
                 -C        drop comments at the header lines
                 -N        drop sequences containing ambiguous bases
                 -1        output the 2n-1 reads only
                 -2        output the 2n reads only
                 -V        shift quality by '(-Q) - 33'
```

Next we will download an example FASTQ file and test out the a simple Seqtk command - `seq` which converts FASTQ files into FASTA.

```
$ wget https://raw.githubusercontent.com/galaxyproject/galaxy-test-data/master/2.fastq
$ seqtk seq -A 2.fastq > 2.fasta
$ cat 2.fasta
>EAS54_6_R1_2_1_413_324
CCCTTCTTGTCTTCAGCGTTTCTCC
>EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
>EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
```

For fully featured Seqtk wrappers check out Helena Rasche's wrappers on GitHub.

Galaxy tool files are just XML files, so at this point one could open a text editor and start writing the tool. Planemo has a command `tool_init` to quickly generate some of the boilerplate XML, so let's start by doing that.

```
$ planemo tool_init --id 'seqtk_seq' --name 'Convert to FASTA (seqtk)'
```

The `tool_init` command can take various complex arguments - but the two most basic ones are shown above `--id` and `--name`. Every Galaxy tool needs an `id` (this is a short identifier used by Galaxy itself to identify the tool) and a `name` (this is displayed to the Galaxy user and should be a short description of the tool). A tool's `name` can have whitespace but its `id` must not.

The above command will generate the file `seqtk_seq.xml` - which looks like this.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        TODO: Fill in command template.
    ]]></command>
    <inputs>
    </inputs>
    <outputs>
    </outputs>
    <help><![CDATA[
        TODO: Fill in help.
    ]]></help>
</tool>
```

This tool file has the common sections required for a Galaxy tool but you will still need to open up the editor and fill out the command template, describe input parameters, tool outputs, write a help section, etc.

The `tool_init` command can do a little bit better than this as well. We can use the test command we tried above `seqtk seq -A 2.fastq > 2.fasta` as an example to generate a command block by specifing the inputs and the outputs as follows.

```
$ planemo tool_init --force \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --requirement seqtk@1.2 \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta
```

This will generate the following XML file - which now has correct definitions for the input and output as well as an actual command template.

```xml
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq -A '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <help><![CDATA[
        TODO: Fill in help.
    ]]></help>
</tool>
```

As shown at the beginning of this section, the command `seqtk seq` generates a help message for the `seq` command. `tool_init` can take that help message and stick it right in the generated tool file using the `help_from_command` option.

Generally command help messages aren't exactly appropriate for tools since they mention argument names and simillar details that are abstracted away by the tool - but they can be an excellent place to start.

The following Planemo's `tool_init` call has been enhanced to use `--help_from_command`.

```
$ planemo tool_init --force \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --requirement seqtk@1.2 \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --test_case \
                    --cite_url 'https://github.com/lh3/seqtk' \
                    --help_from_command 'seqtk seq'
```

In addition to demonstrating `--help_from_command`, this demonstrates generating a test case from our example with `--test_case` and adding a citation for the underlying tool. The resulting tool XML file is:

```xml
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq -A '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
    </inputs>
    <outputs>
```

(continues on next page)

(continued from previous page)

```
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases
         -S        strip of white spaces in sequences


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

At this point we have a fairly a functional Galaxy tool with test and help. This was a pretty simple example - usually you will need to put more work into the tool to get to this point - `tool_init` is really just designed to get you started.

Now lets lint and test the tool we have developed. The Planemo's `lint` (or just `l`) command will review tool for XML validity, obvious mistakes, and compliance with IUC best practices.

```
$ planemo l
Linting tool /opt/galaxy/tools/seqtk_seq.xml
Applying linter tests... CHECK
.. CHECK: 1 test(s) found.
Applying linter output... CHECK
.. INFO: 1 outputs found.
Applying linter inputs... CHECK
.. INFO: Found 1 input parameters.
Applying linter help... CHECK
.. CHECK: Tool contains help section.
.. CHECK: Help contains valid reStructuredText.
Applying linter general... CHECK
.. CHECK: Tool defines a version [0.1.0].
.. CHECK: Tool defines a name [Convert to FASTA (seqtk)].
.. CHECK: Tool defines an id [seqtk_seq].
Applying linter command... CHECK
.. INFO: Tool contains a command.
Applying linter citations... CHECK
.. CHECK: Found 1 likely valid citations.
```

By default `lint` will find all the tools in your current working directory, but we could have specified a particular tool with `planemo lint seqtk_seq.xml`.

Next we can run our tool's functional test with the `test` (or just `t`) command. This will print a lot of output (as it starts a Galaxy instance) but should ultimately reveal our one test passed.

```
$ planemo t
...
All 1 test(s) executed passed.
seqtk_seq[0]: passed
```

In addition to the in console display of test results as red (failing) or green (passing), Planemo also creates an HTML report for the test results by default. Many more test report options are available such `--test_output_xunit` which is useful in certain continuous integration environments. See `planemo test --help` for more options, as well as the `test_reports` command.

Now we can open Galaxy with the `serve` (or just `s`).

```
$ planemo s
...
serving on http://127.0.0.1:9090
```

Open up http://127.0.0.1:9090 in a web browser to view your new tool.

Serve and test can be passed various command line arguments such as `--galaxy_root` to specify a Galaxy instance to use (by default planemo will download and manage a instance just for planemo).

5.1.2 Simple Parameters

We have built a tool wrapper for the `seqtk seq` command - but this tool actually has additional options that we may wish to expose the Galaxy user.

Lets take a few of the parameters from the help command and build Galaxy `param` blocks to stick in the tool's `inputs` block.

```
-V        shift quality by '(-Q) - 33'
```

In the previous section we saw `param` block of type `data` for input files, but there are many different kinds of parameters one can use. Flag parameters such as the above `-V` parameter are frequently represented by `boolean` parameters in Galaxy tool XML.

```
<param name="shift_quality" type="boolean" label="Shift quality"
       truevalue="-V" falsevalue=""
       help="shift quality by '(-Q) - 33' (-V)" />
```

We can then stick `$shift_quality` in our `command` block and if the user has selected this option it will be expanded as `-V` (since we have defined this as the `truevalue`). If the user hasn't selected this option `$shift_quality` will just expand as an empty string and not affect the generated command line.

Now consider the following `seqtk seq` parameters:

```
-q INT    mask bases with quality lower than INT [0]
-X INT    mask bases with quality higher than INT [255]
```

These can be translated into Galaxy parameters as:

```
<param name="quality_min" type="integer" label="Mask bases with quality lower than"
       value="0" min="0" max="255" help="(-q)" />
<param name="quality_max" type="integer" label="Mask bases with quality higher than"
       value="255" min="0" max="255" help="(-X)" />
```

These can be add to the command tag as `-q $quality_min -X $quality_max`.

At this point the tool would look like:

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq
              $shift_quality
              -q $quality_min
              -X $quality_max
              -a '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
        <param name="shift_quality" type="boolean" label="Shift quality"
               truevalue="-V" falsevalue=""
               help="shift quality by '(-Q) - 33' (-V)" />
        <param name="quality_min" type="integer" label="Mask bases with quality lower␣
↪than"
```

(continues on next page)

(continued from previous page)

```
               value="0" min="0" max="255" help="(-q)" />
        <param name="quality_max" type="integer" label="Mask bases with quality higher␣
↪than"
               value="255" min="0" max="255" help="(-X)" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

5.1.3 Conditional Parameters

The previous parameters were simple because they always appeared, now consider.

```
-M FILE   mask regions in BED or name list FILE [null]
```

We can mark this `data` type `param` as optional by adding the attribute `optional="true"`.

```
<param name="mask_regions" type="data" label="Mask regions in BED"
       format="bed" help="(-M)" optional="true" />
```

Then instead of just using `$mask_regions` directly in the `command` block, one can wrap it in an `if` statement (because tool XML files support Cheetah).

```
#if $mask_regions
-M '$mask_regions'
#end if
```

Next consider the parameters:

```
-s INT    random seed (effective with -f) [11]
-f FLOAT  sample FLOAT fraction of sequences [1]
```

In this case, the `-s` random seed parameter should only be seen or used if the sample parameter is set. We can express this using a `conditional` block.

```
<conditional name="sample">
    <param name="sample_selector" type="boolean" label="Sample fraction of sequences" />
    <when value="true">
        <param name="fraction" label="Fraction" type="float" value="1.0"
               help="(-f)" />
        <param name="seed" label="Random seed" type="integer" value="11"
               help="(-s)" />
    </when>
    <when value="false">
    </when>
</conditional>
```

In our command block, we can again use an `if` statement to include these parameters.

```
#if $sample.sample_selector
-f $sample.fraction -s $sample.seed
#end if
```

Notice we must reference the parameters using the `sample.` prefix since they are defined within the `sample` conditional block.

The newest version of this tool is now

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq
```

(continues on next page)

(continued from previous page)

```
              $shift_quality
              -q $quality_min
              -X $quality_max
              #if $mask_regions
                  -M '$mask_regions'
              #end if
              #if $sample.sample
                  -f $sample.fraction
                  -s $sample.seed
              #end if
              -a '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
        <param name="shift_quality" type="boolean" label="Shift quality"
               truevalue="-V" falsevalue=""
               help="shift quality by '(-Q) - 33' (-V)" />
        <param name="quality_min" type="integer" label="Mask bases with quality lower␣
↪than"
               value="0" min="0" max="255" help="(-q)" />
        <param name="quality_max" type="integer" label="Mask bases with quality higher␣
↪than"
               value="255" min="0" max="255" help="(-X)" />
        <param name="mask_regions" type="data" label="Mask regions in BED"
               format="bed" help="(-M)" optional="true" />
        <conditional name="sample">
            <param name="sample" type="boolean" label="Sample fraction of sequences" />
            <when value="true">
                <param name="fraction" label="Fraction" type="float" value="1.0"
                       help="(-f)" />
                <param name="seed" label="Random seed" type="integer" value="11"
                       help="(-s)" />
            </when>
            <when value="false">
            </when>
        </conditional>
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
```

(continues on next page)

(continued from previous page)

```
        -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
        -l INT    number of residues per line; 0 for 2^32-1 [0]
        -Q INT    quality shift: ASCII-INT gives base quality [33]
        -s INT    random seed (effective with -f) [11]
        -f FLOAT  sample FLOAT fraction of sequences [1]
        -M FILE   mask regions in BED or name list FILE [null]
        -L INT    drop sequences with length shorter than INT [0]
        -c        mask complement region (effective with -M)
        -r        reverse complement
        -A        force FASTA output (discard quality)
        -C        drop comments at the header lines
        -N        drop sequences containing ambiguous bases
        -1        output the 2n-1 reads only
        -2        output the 2n reads only
        -V        shift quality by '(-Q) - 33'
        -U        convert all bases to uppercases


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

For tools like this where there are many options but in the most uses the defaults are preferred - a common idiom is to break the parameters into simple and advanced sections using a `conditional`.

Updating this tool to use that idiom might look as follows.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq
              #if $settings.advanced == "advanced"
                  $settings.shift_quality
                  -q $settings.quality_min
                  -X $settings.quality_max
                  #if $settings.mask_regions
                      -M '$settings.mask_regions'
                  #end if
                  #if $settings.sample.sample
                      -f $settings.sample.fraction
                      -s $settings.sample.seed
```

(continues on next page)

(continued from previous page)

```
                  #end if
              #end if
              -a '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
        <conditional name="settings">
            <param name="advanced" type="select" label="Specify advanced parameters">
                <option value="simple" selected="true">No, use program defaults.</option>
                <option value="advanced">Yes, see full parameter list.</option>
            </param>
            <when value="simple">
            </when>
            <when value="advanced">
                <param name="shift_quality" type="boolean" label="Shift quality"
                       truevalue="-V" falsevalue=""
                       help="shift quality by '(-Q) - 33' (-V)" />
                <param name="quality_min" type="integer" label="Mask bases with quality␣
↪lower than"
                       value="0" min="0" max="255" help="(-q)" />
                <param name="quality_max" type="integer" label="Mask bases with quality␣
↪higher than"
                       value="255" min="0" max="255" help="(-X)" />
                <param name="mask_regions" type="data" label="Mask regions in BED"
                       format="bed" help="(-M)" optional="true" />
                <conditional name="sample">
                    <param name="sample" type="boolean" label="Sample fraction of␣
↪sequences" />
                    <when value="true">
                        <param name="fraction" label="Fraction" type="float" value="1.0"
                               help="(-f)" />
                        <param name="seed" label="Random seed" type="integer" value="11"
                               help="(-s)" />
                    </when>
                    <when value="false">
                    </when>
                </conditional>
            </when>
        </conditional>
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>
```

(continues on next page)

(continued from previous page)

```
Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

5.1.4 Wrapping a Script

Many common bioinformatics applications are available on the Tool Shed already and so a common development task is to integrate scripts of various complexity into Galaxy.

Consider the following small Perl script.

```
#!/usr/bin/perl -w

# usage : perl toolExample.pl <FASTA file> <output file>

open (IN, "<$ARGV[0]");
open (OUT, ">$ARGV[1]");
while (<IN>) {
    chop;
    if (m/^>/) {
```

(continues on next page)

(continued from previous page)

```
        s/^>//;
        if ($. > 1) {
            print OUT sprintf("%.3f", $gc/$length) . "\n";
        }
        $gc = 0;
        $length = 0;
    } else {
        ++$gc while m/[gc]/ig;
        $length += length $_;
    }
}
print OUT sprintf("%.3f", $gc/$length) . "\n";
close( IN );
close( OUT );
```

One can build a tool for this script as follows and place the script in the same directory as the tool XML file itself. The special value `$__tool_directory__` here refers to the directory your tool lives in.

```
<tool id="gc_content" name="Compute GC content">
  <description>for each sequence in a file</description>
  <command>perl '$__tool_directory__/gc_content.pl' '$input' output.tsv</command>
  <inputs>
    <param name="input" type="data" format="fasta" label="Source file"/>
  </inputs>
  <outputs>
    <data name="output" format="tabular" from_work_dir="output.tsv" />
  </outputs>
  <help>
This tool computes GC content from a FASTA file.
  </help>
</tool>
```

5.1.5 Macros

If your desire is to write a tool for a single relatively simple application or script - this section should be skipped. If you hope to maintain a collection of related tools - experience suggests you will realize there is a lot of duplicated XML to do this well. Galaxy tool XML macros can help reduce this duplication.

Planemo's `tool_init` command can be used to generate a macro file appropriate for suites of tools by using the `--macros` flag. Consider the following variant of the previous `tool_init` command (the only difference is now we are adding the `--macros` flag).

```
$ planemo tool_init --force \
                    --macros \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --requirement seqtk@1.2 \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --test_case \
                    --help_from_command 'seqtk seq'
```

This will produce the two files in your current directory instead of just one - `seqtk_seq.xml` and `macros.xml`.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <macros>
        <import>macros.xml</import>
    </macros>
    <expand macro="requirements" />
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq -A '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'


    ]]></help>
    <expand macro="citations" />
</tool>
```

```
<macros>
    <xml name="requirements">
        <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
```

(continues on next page)

(continued from previous page)

```
            <yield/>
        </requirements>
    </xml>
    <xml name="citations">
        <citations>
            <yield />
        </citations>
    </xml>
</macros>
```

As you can see in the above code macros are reusable chunks of XML that make it easier to avoid duplication and keep your XML concise.

Further reading:

- Macros syntax on the Galaxy Wiki.
- GATK tools (example tools making extensive use of macros)
- gemini tools (example tools making extensive use of macros)
- bedtools tools (example tools making extensive use of macros)
- Macros implementation details - Pull Request #129 and Pull Request #140

5.1.6 Creating Galaxy tools from python scripts using argparse

If a python script should be wrapped that creates its command line interface using the argparse library, then planemo can generate a good starting point.

All you need to do is to pass the path to the python source file containing the argparse definition. Lets use `tests/data/autopygen/autopygen_end_to_end_sub.py` from the planemo tests as an example:

```
$ planemo tool_init -i e2e -n e2e
                    --autopygen tests/data/autopygen/autopygen_end_to_end_sub.py
                    -- tool tool.xml
```

Here we only provided the `id` and `name` in addition to the python sources and planemo creates a pretty tool xml files that can serve as a good starting point to create a functional Galaxy tool in `tool.xml`. Additional information, e.g. requirements, help, etc, can and should be given with additional parameters to planemo.

There are a few points that need to be edited in any case. Most importantly since in most cases `argparse` does not distinguish between input and output files all file parameters will be rendered as input parameters of the tool.

Please open an issue if you have ideas on how to improve the generated tools: https://github.com/galaxyproject/planemo/issues

5.1.7 More Information

- Galaxy's Tool XML Syntax
- Big List of Tool Development Resources
- Cheetah templating

5.2 Building Galaxy Tools (Using the Planemo Appliance)

This tutorial is a gentle introduction to writing Galaxy tools using the Planemo virtual appliance (available as OVA, Docker and Vagrant). Check out these instructions for obtaining the virtual appliance if you have not done so already.

Note: Please note that you can leverage the clipboard for sharing text between the virtual image environment and your host system. To copy in the VM **terminal** use `ctrl` + `shift` + `C` and to paste use `ctrl` + `shift` + `V`. To copy in the VM **Firefox browser** use `ctrl` + `C`. Use the corresponding commands on your host system (e.g. `Command` + `C` on MacOS).

5.2.1 The Basics

This guide is going to demonstrate building up tools for commands from Heng Li's Seqtk package - a package for processing sequence data in FASTA and FASTQ files.

To get started let's install Seqtk. Here we are going to use `conda` to install Seqtk - but however you obtain it should be fine.

```
$ conda install --force --yes -c conda-forge -c bioconda seqtk=1.2
    ... seqtk installation ...
$ seqtk seq
        Usage:   seqtk seq [options] <in.fq>|<in.fa>
        Options: -q INT    mask bases with quality lower than INT [0]
                 -X INT    mask bases with quality higher than INT [255]
                 -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
                 -l INT    number of residues per line; 0 for 2^32-1 [0]
                 -Q INT    quality shift: ASCII-INT gives base quality [33]
                 -s INT    random seed (effective with -f) [11]
                 -f FLOAT  sample FLOAT fraction of sequences [1]
                 -M FILE   mask regions in BED or name list FILE [null]
                 -L INT    drop sequences with length shorter than INT [0]
                 -c        mask complement region (effective with -M)
                 -r        reverse complement
                 -A        force FASTA output (discard quality)
                 -C        drop comments at the header lines
                 -N        drop sequences containing ambiguous bases
                 -1        output the 2n-1 reads only
                 -2        output the 2n reads only
                 -V        shift quality by '(-Q) - 33'
```

Next we will download an example FASTQ file and test out the a simple Seqtk command - `seq` which converts FASTQ files into FASTA.

```
$ wget https://raw.githubusercontent.com/galaxyproject/galaxy-test-data/master/2.fastq
$ seqtk seq -A 2.fastq > 2.fasta
$ cat 2.fasta
>EAS54_6_R1_2_1_413_324
CCCTTCTTGTCTTCAGCGTTTCTCC
>EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
>EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
```

For fully featured Seqtk wrappers check out Helena Rasche's wrappers on GitHub.

Galaxy tool files are just XML files, so at this point one could open a text editor and start writing the tool. Planemo has a command `tool_init` to quickly generate some of the boilerplate XML, so let's start by doing that.

```
$ planemo tool_init --id 'seqtk_seq' --name 'Convert to FASTA (seqtk)'
```

The `tool_init` command can take various complex arguments - but the two most basic ones are shown above `--id` and `--name`. Every Galaxy tool needs an `id` (this is a short identifier used by Galaxy itself to identify the tool) and a `name` (this is displayed to the Galaxy user and should be a short description of the tool). A tool's `name` can have whitespace but its `id` must not.

The above command will generate the file `seqtk_seq.xml` - which looks like this.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        TODO: Fill in command template.
    ]]></command>
    <inputs>
    </inputs>
    <outputs>
    </outputs>
    <help><![CDATA[
        TODO: Fill in help.
    ]]></help>
</tool>
```

This tool file has the common sections required for a Galaxy tool but you will still need to open up the editor and fill out the command template, describe input parameters, tool outputs, write a help section, etc.

The `tool_init` command can do a little bit better than this as well. We can use the test command we tried above `seqtk seq -A 2.fastq > 2.fasta` as an example to generate a command block by specifing the inputs and the outputs as follows.

```
$ planemo tool_init --force \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --requirement seqtk@1.2 \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta
```

This will generate the following XML file - which now has correct definitions for the input and output as well as an actual command template.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq -A '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <help><![CDATA[
        TODO: Fill in help.
    ]]></help>
</tool>
```

As shown at the beginning of this section, the command `seqtk seq` generates a help message for the `seq` command. `tool_init` can take that help message and stick it right in the generated tool file using the `help_from_command` option.

Generally command help messages aren't exactly appropriate for tools since they mention argument names and simillar details that are abstracted away by the tool - but they can be an excellent place to start.

The following Planemo's `tool_init` call has been enhanced to use `--help_from_command`.

```
$ planemo tool_init --force \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --requirement seqtk@1.2 \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --test_case \
                    --cite_url 'https://github.com/lh3/seqtk' \
                    --help_from_command 'seqtk seq'
```

In addition to demonstrating `--help_from_command`, this demonstrates generating a test case from our example with `--test_case` and adding a citation for the underlying tool. The resulting tool XML file is:

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq -A '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
```

(continues on next page)

(continued from previous page)

```
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases
         -S        strip of white spaces in sequences


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

At this point we have a fairly a functional Galaxy tool with test and help. This was a pretty simple example - usually you will need to put more work into the tool to get to this point - `tool_init` is really just designed to get you started.

Now lets lint and test the tool we have developed. The Planemo's `lint` (or just `l`) command will review tool for XML validity, obvious mistakes, and compliance with IUC best practices.

```
$ planemo l
Linting tool /opt/galaxy/tools/seqtk_seq.xml
```

(continues on next page)

(continued from previous page)

```
Applying linter tests... CHECK
.. CHECK: 1 test(s) found.
Applying linter output... CHECK
.. INFO: 1 outputs found.
Applying linter inputs... CHECK
.. INFO: Found 1 input parameters.
Applying linter help... CHECK
.. CHECK: Tool contains help section.
.. CHECK: Help contains valid reStructuredText.
Applying linter general... CHECK
.. CHECK: Tool defines a version [0.1.0].
.. CHECK: Tool defines a name [Convert to FASTA (seqtk)].
.. CHECK: Tool defines an id [seqtk_seq].
Applying linter command... CHECK
.. INFO: Tool contains a command.
Applying linter citations... CHECK
.. CHECK: Found 1 likely valid citations.
```

By default `lint` will find all the tools in your current working directory, but we could have specified a particular tool with `planemo lint seqtk_seq.xml`.

Next we can run our tool's functional test with the `test` (or just `t`) command. This will print a lot of output (as it starts a Galaxy instance) but should ultimately reveal our one test passed.

```
$ planemo t
... Galaxy starts and runs the test ...
All 1 test(s) executed passed.
seqtk_seq[0]: passed
```

You can use the following command to open up the test results in your browser.

```
$ firefox /opt/galaxy/tools/tool_test_output.html
```

Normally `planemo` requires an existing Galaxy instance to point at to run the `t` (or `test` command) - but the virtual appliance has a Galaxy instance preconfigured and registered with `planemo`.

5.2.2 Simple Parameters

We have built a tool wrapper for the `seqtk seq` command - but this tool actually has additional options that we may wish to expose the Galaxy user.

Lets take a few of the parameters from the help command and build Galaxy `param` blocks to stick in the tool's `inputs` block.

```
-V        shift quality by '(-Q) - 33'
```

In the previous section we saw `param` block of type `data` for input files, but there are many different kinds of parameters one can use. Flag parameters such as the above `-V` parameter are frequently represented by `boolean` parameters in Galaxy tool XML.

```
<param name="shift_quality" type="boolean" label="Shift quality"
       truevalue="-V" falsevalue=""
       help="shift quality by '(-Q) - 33' (-V)" />
```

We can then stick `$shift_quality` in our `command` block and if the user has selected this option it will be expanded as `-V` (since we have defined this as the `truevalue`). If the user hasn't selected this option `$shift_quality` will just expand as an empty string and not affect the generated command line.

Now consider the following `seqtk seq` parameters:

```
-q INT    mask bases with quality lower than INT [0]
-X INT    mask bases with quality higher than INT [255]
```

These can be translated into Galaxy parameters as:

```
<param name="quality_min" type="integer" label="Mask bases with quality lower than"
       value="0" min="0" max="255" help="(-q)" />
<param name="quality_max" type="integer" label="Mask bases with quality higher than"
       value="255" min="0" max="255" help="(-X)" />
```

These can be add to the command tag as `-q $quality_min -X $quality_max`.

At this point the tool would look like:

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq
              $shift_quality
              -q $quality_min
              -X $quality_max
              -a '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
        <param name="shift_quality" type="boolean" label="Shift quality"
               truevalue="-V" falsevalue=""
               help="shift quality by '(-Q) - 33' (-V)" />
        <param name="quality_min" type="integer" label="Mask bases with quality lower␣
↪than"
               value="0" min="0" max="255" help="(-q)" />
        <param name="quality_max" type="integer" label="Mask bases with quality higher␣
↪than"
               value="255" min="0" max="255" help="(-X)" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>
```

(continues on next page)

(continued from previous page)

```

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

5.2.3 Conditional Parameters

The previous parameters were simple because they always appeared, now consider.

```
-M FILE   mask regions in BED or name list FILE [null]
```

We can mark this `data` type `param` as optional by adding the attribute `optional="true"`.

```
<param name="mask_regions" type="data" label="Mask regions in BED"
       format="bed" help="(-M)" optional="true" />
```

Then instead of just using `$mask_regions` directly in the `command` block, one can wrap it in an `if` statement (because tool XML files support Cheetah).

```
#if $mask_regions
-M '$mask_regions'
#end if
```

Next consider the parameters:

```
-s INT    random seed (effective with -f) [11]
-f FLOAT  sample FLOAT fraction of sequences [1]
```

In this case, the `-s` random seed parameter should only be seen or used if the sample parameter is set. We can express this using a `conditional` block.

```
<conditional name="sample">
    <param name="sample_selector" type="boolean" label="Sample fraction of sequences" />
    <when value="true">
        <param name="fraction" label="Fraction" type="float" value="1.0"
               help="(-f)" />
        <param name="seed" label="Random seed" type="integer" value="11"
               help="(-s)" />
    </when>
    <when value="false">
    </when>
</conditional>
```

In our command block, we can again use an `if` statement to include these parameters.

```
#if $sample.sample_selector
-f $sample.fraction -s $sample.seed
#end if
```

Notice we must reference the parameters using the `sample.` prefix since they are defined within the `sample` conditional block.

The newest version of this tool is now

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq
              $shift_quality
              -q $quality_min
              -X $quality_max
              #if $mask_regions
                  -M '$mask_regions'
              #end if
              #if $sample.sample
                  -f $sample.fraction
                  -s $sample.seed
              #end if
              -a '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
        <param name="shift_quality" type="boolean" label="Shift quality"
               truevalue="-V" falsevalue=""
               help="shift quality by '(-Q) - 33' (-V)" />
        <param name="quality_min" type="integer" label="Mask bases with quality lower␣
```

(continues on next page)

(continued from previous page)

```
↪than"
              value="0" min="0" max="255" help="(-q)" />
        <param name="quality_max" type="integer" label="Mask bases with quality higher␣
↪than"
              value="255" min="0" max="255" help="(-X)" />
        <param name="mask_regions" type="data" label="Mask regions in BED"
              format="bed" help="(-M)" optional="true" />
        <conditional name="sample">
            <param name="sample" type="boolean" label="Sample fraction of sequences" />
            <when value="true">
                <param name="fraction" label="Fraction" type="float" value="1.0"
                       help="(-f)" />
                <param name="seed" label="Random seed" type="integer" value="11"
                       help="(-s)" />
            </when>
            <when value="false">
            </when>
        </conditional>
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases
```

(continues on next page)

(continued from previous page)

```
    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

For tools like this where there are many options but in the most uses the defaults are preferred - a common idiom is to break the parameters into simple and advanced sections using a `conditional`.

Updating this tool to use that idiom might look as follows.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq
              #if $settings.advanced == "advanced"
                  $settings.shift_quality
                  -q $settings.quality_min
                  -X $settings.quality_max
                  #if $settings.mask_regions
                      -M '$settings.mask_regions'
                  #end if
                  #if $settings.sample.sample
                      -f $settings.sample.fraction
                      -s $settings.sample.seed
                  #end if
              #end if
              -a '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
        <conditional name="settings">
            <param name="advanced" type="select" label="Specify advanced parameters">
                <option value="simple" selected="true">No, use program defaults.</option>
                <option value="advanced">Yes, see full parameter list.</option>
            </param>
            <when value="simple">
            </when>
            <when value="advanced">
                <param name="shift_quality" type="boolean" label="Shift quality"
                       truevalue="-V" falsevalue=""
                       help="shift quality by '(-Q) - 33' (-V)" />
                <param name="quality_min" type="integer" label="Mask bases with quality␣
```

(continues on next page)

(continued from previous page)

```
↪lower than"
                     value="0" min="0" max="255" help="(-q)" />
              <param name="quality_max" type="integer" label="Mask bases with quality␣
↪higher than"
                     value="255" min="0" max="255" help="(-X)" />
              <param name="mask_regions" type="data" label="Mask regions in BED"
                     format="bed" help="(-M)" optional="true" />
              <conditional name="sample">
                  <param name="sample" type="boolean" label="Sample fraction of␣
↪sequences" />
                  <when value="true">
                      <param name="fraction" label="Fraction" type="float" value="1.0"
                             help="(-f)" />
                      <param name="seed" label="Random seed" type="integer" value="11"
                             help="(-s)" />
                  </when>
                  <when value="false">
                  </when>
              </conditional>
          </when>
      </conditional>
  </inputs>
  <outputs>
      <data name="output1" format="fasta" />
  </outputs>
  <tests>
      <test>
          <param name="input1" value="2.fastq"/>
          <output name="output1" file="2.fasta"/>
      </test>
  </tests>
  <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
```

(continues on next page)

(continued from previous page)

```
        -U        convert all bases to uppercases


    ]]></help>
    <citations>
        <citation type="bibtex">
@misc{githubseqtk,
  author = {LastTODO, FirstTODO},
  year = {TODO},
  title = {seqtk},
  publisher = {GitHub},
  journal = {GitHub repository},
  url = {https://github.com/lh3/seqtk},
}</citation>
    </citations>
</tool>
```

5.2.4 Publishing to the Tool Shed

Now that the tool is working and useful - it is time to publish it to the Tool Shed. The Galaxy Tool Shed (referred to colloquially in Planemo as the "shed") can store Galaxy tools, dependency definitions, and workflows among other Galaxy artifacts. Shed's goal is to make it easy for any Galaxy to install these.

Configuring a Tool Shed Account

The planemo appliance comes pre-configured with a local Tool Shed and Planemo is configured to talk to it via `~/.planemo.yml` configuration file. Check out the publishing docs for information on setting up this file on your development environment.

Creating a Repository

Planemo can be used to publish "repositories" to the Tool Shed. A single GitHub repository or locally managed directory of tools may correspond to any number of Tool Shed repositories. Planemo maps files to Tool Shed repositories using a special file called `.shed.yml`.

From a directory containing tools the `shed_init` command can be used to bootstrap a new `.shed.yml` file.

```
$ planemo shed_init --name=seqtk_seq \
                    --owner=planemo \
                    --description=seqtk_seq \
                    --long_description="Tool that converts FASTQ to FASTA files using␣
↪seqtk" \
                    --category="Fastq Manipulation"
```

The resulting `.shed.yml` file will look something like this:

```
categories: [Fastq Manipulation]
description: seqtk_seq
long_description: Tool that converts FASTQ to FASTA files using seqtk
name: seqtk_seq
owner: planemo
```

There is not a lot of magic happening here, this file could easily be created directly with a text editor - but the command has a `--help` to assist you and does some very basic validation. More information on `.shed.yml` can be found as part of the IUC's best practice documentation.

This configuration file and shed artifacts can be quickly linted using the following command.

```
$ planemo shed_lint --tools
```

Once the details in the `.shed.yml` are set and it is time to create the remote repository and upload artifacts to it - the following two commands can be used - the first only needs to be run once and creates the repository based on the metadata in `.shed.yml` and the second uploads your actual artifacts to it.

```
$ planemo shed_create --shed_target local
Repository created
cd '/opt/galaxy/tools' && git rev-parse HEAD
Repository seqtk_seq updated successfully.
```

You can now navigate to the local shed (likely at http://localhost:9009/) and see the repository there. Optionally you can login with username `planemo@test.com` and password `planemo` but it is not necessary.

Updating a Repository

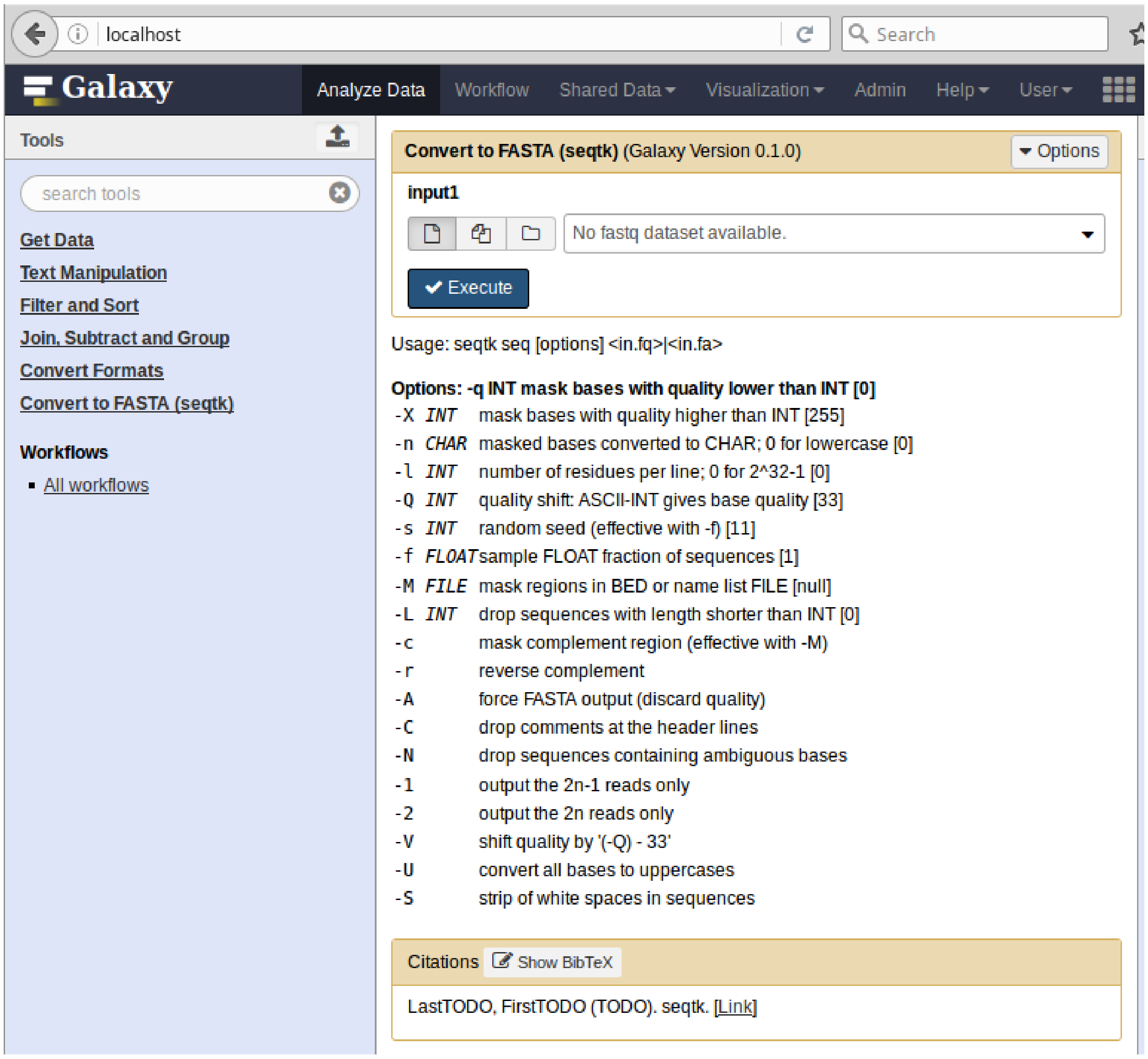
In order to push further changes in your local tool development directory to the shed you would run the `shed_update` command as follows.

```
$ planemo shed_update --shed_target local
```

Serving a Tool from Shed

Once tools (and possible required dependency files) have been published, the whole thing can be automatically installed and the tool served in local Galaxy using this command.

```
$ planemo shed_serve --shed_target local
```



Convert to FASTA (seqtk) (Galaxy Version 0.1.0)

input1

No fastq dataset available.

Usage: seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT mask bases with quality lower than INT [0]

-X	*INT*	mask bases with quality higher than INT [255]
-n	*CHAR*	masked bases converted to CHAR; 0 for lowercase [0]
-l	*INT*	number of residues per line; 0 for 2^32-1 [0]
-Q	*INT*	quality shift: ASCII-INT gives base quality [33]
-s	*INT*	random seed (effective with -f) [11]
-f	*FLOAT*	sample FLOAT fraction of sequences [1]
-M	*FILE*	mask regions in BED or name list FILE [null]
-L	*INT*	drop sequences with length shorter than INT [0]
-c		mask complement region (effective with -M)
-r		reverse complement
-A		force FASTA output (discard quality)
-C		drop comments at the header lines
-N		drop sequences containing ambiguous bases
-1		output the 2n-1 reads only
-2		output the 2n reads only
-V		shift quality by '(-Q) - 33'
-U		convert all bases to uppercases
-S		strip of white spaces in sequences

Citations

LastTODO, FirstTODO (TODO). seqtk. [Link]

Note: During this tutorial we did not "teach" Galaxy how to obtain the seqtk software so our tool (and thus Galaxy) just expects the command `seqtk` to be available. The seqtk software here is a so called `dependency` of our tool and in order for our tool to be fully installable we need to create a "recipe" for Galaxy so it knows how to obtain it. This is covered in other sections of this documentation as well as on the wiki.

Main Tool Shed

Once your artifacts are ready for publication to the Main Tool Shed, the following command creates a repository there and populates it with your contents.

```
$ planemo shed_create --shed_target toolshed
```

The planemo machine isn't preconfigured to allow publishing to the Main Tool Shed so this command will not work here. See the more complete publishing docs for full details about how to setup Planemo to publish to the Main and Test Tool Shed - the process is very similar.

5.2.5 Wrapping a Script

Many common bioinformatics applications are available on the Tool Shed already and so a common development task is to integrate scripts of various complexity into Galaxy.

Consider the following small Perl script.

```
#!/usr/bin/perl -w

# usage : perl toolExample.pl <FASTA file> <output file>

open (IN, "<$ARGV[0]");
open (OUT, ">$ARGV[1]");
while (<IN>) {
    chop;
    if (m/^>/) {
        s/^>//;
        if ($. > 1) {
            print OUT sprintf("%.3f", $gc/$length) . "\n";
        }
        $gc = 0;
        $length = 0;
    } else {
        ++$gc while m/[gc]/ig;
        $length += length $_;
    }
}
print OUT sprintf("%.3f", $gc/$length) . "\n";
close( IN );
close( OUT );
```

One can build a tool for this script as follows and place the script in the same directory as the tool XML file itself. The special value `$__tool_directory__` here refers to the directory your tool lives in.

```
<tool id="gc_content" name="Compute GC content">
  <description>for each sequence in a file</description>
  <command>perl '$__tool_directory__/gc_content.pl' '$input' output.tsv</command>
  <inputs>
    <param name="input" type="data" format="fasta" label="Source file"/>
  </inputs>
  <outputs>
    <data name="output" format="tabular" from_work_dir="output.tsv" />
```

(continues on next page)

(continued from previous page)

```
  </outputs>
  <help>
This tool computes GC content from a FASTA file.
  </help>
</tool>
```

5.2.6 Macros

If your desire is to write a tool for a single relatively simple application or script - this section should be skipped. If you hope to maintain a collection of related tools - experience suggests you will realize there is a lot of duplicated XML to do this well. Galaxy tool XML macros can help reduce this duplication.

Planemo's `tool_init` command can be used to generate a macro file appropriate for suites of tools by using the `--macros` flag. Consider the following variant of the previous `tool_init` command (the only difference is now we are adding the `--macros` flag).

```
$ planemo tool_init --force \
                    --macros \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --requirement seqtk@1.2 \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --test_case \
                    --help_from_command 'seqtk seq'
```

This will produce the two files in your current directory instead of just one - `seqtk_seq.xml` and `macros.xml`.

```
<tool id="seqtk_seq" name="Convert to FASTA (seqtk)" version="0.1.0">
    <macros>
        <import>macros.xml</import>
    </macros>
    <expand macro="requirements" />
    <command detect_errors="exit_code"><![CDATA[
        seqtk seq -A '$input1' > '$output1'
    ]]></command>
    <inputs>
        <param type="data" name="input1" format="fastq" />
    </inputs>
    <outputs>
        <data name="output1" format="fasta" />
    </outputs>
    <tests>
        <test>
            <param name="input1" value="2.fastq"/>
            <output name="output1" file="2.fasta"/>
        </test>
    </tests>
    <help><![CDATA[

Usage:   seqtk seq [options] <in.fq>|<in.fa>
```

(continues on next page)

(continued from previous page)

```

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'


    ]]></help>
    <expand macro="citations" />
</tool>
```

```
<macros>
    <xml name="requirements">
        <requirements>
        <requirement type="package" version="1.2">seqtk</requirement>
            <yield/>
        </requirements>
    </xml>
    <xml name="citations">
        <citations>
            <yield />
        </citations>
    </xml>
</macros>
```

As you can see in the above code macros are reusable chunks of XML that make it easier to avoid duplication and keep your XML concise.

Further reading:

- Macros syntax on the Galaxy Wiki.
- GATK tools (example tools making extensive use of macros)
- gemini tools (example tools making extensive use of macros)
- bedtools tools (example tools making extensive use of macros)
- Macros implementation details - Pull Request #129 and Pull Request #140

5.2.7 More Information

- Galaxy's Tool XML Syntax
- Big List of Tool Development Resources
- Cheetah templating

Additional tutorials include

5.3 Advanced Tool Development Topics

This tutorial covers some more advanced tool development topics - such as testing and collections. It assumes some basic knowledge about wrapping Galaxy tools and that you have an environment with Planemo available - check out tutorial if you have never developed a Galaxy tool.

5.3.1 Test-Driven Development

An Example Tool - BWA

To get started let's install BWA. Here we are going to use conda to install BWA - but however you obtain it should be fine.

```
$ conda install --force -c conda-forge -c bioconda bwa
    ... bwa installation ...
$ bwa
Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.13-r1126
Contact: Heng Li <lh3@sanger.ac.uk>

Usage:   bwa <command> [options]

Command: index         index sequences in the FASTA format
         mem           BWA-MEM algorithm
         fastmap       identify super-maximal exact matches
         pemerge       merge overlapping paired ends (EXPERIMENTAL)
         aln           gapped/ungapped alignment
         samse         generate alignment (single ended)
         sampe         generate alignment (paired ended)
         bwasw         BWA-SW for long queries

         shm           manage indices in shared memory
         fa2pac        convert FASTA to PAC format
         pac2bwt       generate BWT from PAC
         pac2bwtgen    alternative algorithm for generating BWT
         bwtupdate     update .bwt to the new format
         bwt2sa        generate SA from BWT and Occ

Note: To use BWA, you need to first index the genome with `bwa index'.
      There are three alignment algorithms in BWA: `mem', `bwasw', and
      `aln/samse/sampe'. If you are not sure which to use, try `bwa mem'
      first. Please `man ./bwa.1' for the manual.
```

Alternatively you can use Homebrew/linuxbrew to install it:

```
$ brew install homebrew/science/bwa
```

Lets start with a simple wrapper for the BWA application (`bwa mem` in particular). You can create a new mini-project with a minimal bwa-mem tool using Planemo's `project_init` command.

```
$ planemo project_init --template bwa bwa
$ cd bwa
```

This will create a folder with a `bwa-mem.xml` as follows:

```
<?xml version="1.0"?>
<tool id="bwa_mem_test" name="Map with BWA-MEM" version="0.0.1">
    <description>- map medium and long reads</description>
    <requirements>
        <requirement type="package" version="0.7.15">bwa</requirement>
        <requirement type="package" version="1.3">samtools</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
      ## Build reference
      #set $reference_fasta_filename = "localref.fa"
      ln -s "${ref_file}" "${reference_fasta_filename}" &&
      bwa index -a is "${reference_fasta_filename}" &&

      ## Begin BWA-MEM command line
      bwa mem
      -t "\${GALAXY_SLOTS:-4}"
      -v 1                                                                              ##␣
↪Verbosity is set to 1 (errors only)

      "${reference_fasta_filename}"
      "${fastq_input1}"

      | samtools view -Sb - > temporary_bam_file.bam &&
      samtools sort -o ${bam_output} temporary_bam_file.bam
    ]]></command>

    <inputs>
        <param name="ref_file" type="data" format="fasta" label="Use the following␣
↪dataset as the reference sequence" help="You can upload a FASTA sequence to the␣
↪history and use it as reference" />
        <param name="fastq_input1" type="data" format="fastqsanger" label="Select fastq␣
↪dataset" help="Specify dataset with single reads"/>
    </inputs>

    <outputs>
        <data format="bam" name="bam_output" label="${tool.name} on ${on_string} (mapped␣
↪reads in BAM format)"/>
    </outputs>

    <tests>
        <!-- header describing command-line will always be different -
             hence lines_diff="2" on output tag. -->
```

(continues on next page)

(continued from previous page)

```
        <test>
            <param name="fastq_input1" value="bwa-mem-fastq1.fq" />
            <param name="ref_file" value="bwa-mem-mt-genome.fa" />
            <output name="bam_output" file="bwa-aln-test1.bam" ftype="bam" lines_diff="2
↪" />
        </test>
    </tests>
    <help>

**BWA MEM options**

Algorithm options::

       -k INT        minimum seed length [19]
       -w INT        band width for banded alignment [100]
       -d INT        off-diagonal X-dropoff [100]
       -r FLOAT      look for internal seeds inside a seed longer than {-k} * FLOAT [1.5]
       -y INT        find MEMs longer than {-k} * {-r} with size less than INT [0]
       -c INT        skip seeds with more than INT occurrences [500]
       -D FLOAT      drop chains shorter than FLOAT fraction of the longest overlapping␣
↪chain [0.50]
       -W INT        discard a chain if seeded bases shorter than INT [0]
       -m INT        perform at most INT rounds of mate rescues for each read [50]
       -S            skip mate rescue
       -P            skip pairing; mate rescue performed unless -S also in use
       -e            discard full-length exact matches

Scoring options::

       -A INT        score for a sequence match, which scales options -TdBOELU unless␣
↪overridden [1]
       -B INT        penalty for a mismatch [4]
       -O INT[,INT]  gap open penalties for deletions and insertions [6,6]
       -E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
       -L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
       -U INT        penalty for an unpaired read pair [17]

Input/output options::

       -p            first query file consists of interleaved paired-end sequences
       -R STR        read group header line such as '@RG\tID:foo\tSM:bar' [null]

       -v INT        verbose level: 1=error, 2=warning, 3=message, 4+=debugging [3]
       -T INT        minimum score to output [30]
       -h INT        if there are &lt;INT hits with score &gt;80% of the max score,␣
↪output all in XA [5]
       -a            output all alignments for SE or unpaired PE
       -C            append FASTA/FASTQ comment to SAM output
       -V            output the reference FASTA header in the XR tag
       -Y            use soft clipping for supplementary alignments
       -M            mark shorter split hits as secondary
```

(continues on next page)

(continued from previous page)

```
     -I FLOAT[,FLOAT[,INT[,INT]]]
                       specify the mean, standard deviation (10% of the mean if absent),␣
↪max
                       (4 sigma from the mean if absent) and min of the insert size␣
↪distribution.
                       FR orientation only. [inferred]
    </help>
    <citations>
      <citation type="doi">10.1093/bioinformatics/btp698</citation>
    </citations>
</tool>
```

Highlighted are two features of Galaxy's tool XML syntax. The `detect_errors="exit_code"` on the `command` block will cause Galaxy to use the actual process exit code to determine failure - in most cases this is superior to the default Galaxy behavior of checking for the presence of standard error output.

The `<citations>` block at the bottom will cause Galaxy to generate exportable citations in the tool form and history UIs.

Improved Input Handling via Test-Driven Development

In this form, the tool only accepts a single input. The first thing we will do is to expand the tool to also allow paired datasets.

Note: Two big ideas behind test-driven development are:

- Write a failing test first.
- Run the test before you implement the feature. Seeing the initial test failing ensures that your feature is actually being tested.

So let's start by generating a test output for the two input files (the bootstrapped example includes two fastq input files to work with `bwa-mem-fastq1.fq` and `bwa-mem-fastq2.fq`). The following commands will create a bwa index on the fly, map two input files against it, and build and sort a bam output from the result - all following the pattern from the command block in the tool.

```
$ cd test-data
$ bwa index -a is bwa-mem-mt-genome.fa
$ bwa mem bwa-mem-mt-genome.fa bwa-mem-fastq1.fq bwa-mem-fastq2.fq | \
  samtools view -Sb - > temporary_bam_file.bam && \
  (samtools sort -f temporary_bam_file.bam bwa-aln-test2.bam || samtools sort -o bwa-aln-
↪test2.bam temporary_bam_file.bam)
```

Warning: In many ways this magic is the hardest part of wrapping Galaxy tools and is something this tutorial cannot really teach. The command line magic required for each tool is going to be different. Developing a Galaxy wrapper requires a lot of knowledge of the underlying applications.

Note: Sort appears twice in this odd command because two different versions of samtools with conflicting syntaxes may happen to be on your machine when running this command. Galaxy manages versioned dependencies and so the tool itself does not reflect this complexity.

The primary result of this is the file `test-data/bwa-aln-test2.bam`. We will now copy and paste the existing test case to add a new test case that specifies both fastq inputs as a collection and expects this new output.

```
<test>
    <param name="fastq_input">
        <collection type="paired">
            <element name="forward" value="bwa-mem-fastq1.fq" />
            <element name="reverse" value="bwa-mem-fastq2.fq" />
        </collection>
    </param>
    <param name="ref_file" value="bwa-mem-mt-genome.fa" />
    <param name="input_type" value="paired_collection" />
    <output name="bam_output" file="bwa-aln-test2.bam" ftype="bam" lines_diff="2" />
</test>
```

We want to specify the input datasets as a paired collection (see the collections documentation in this document for more information) and we need to have a way to allow the user to specify they are submitting a paired collection instead of a single input. This is where the `fastq_input` and `input_type` variables above came from.

Next run `planemo l` to verify the tool doesn't have any obvious defects. Once the XML is valid - use `planemo t` to verify the new test is failing.

```
$ planemo t
... bunch of output ...
bwa_mem_test[0]: passed
bwa_mem_test[1]: failed
```

Note: You can run `$ firefox tool_test_output.html` to see full details of all executed tests.

Here you can see this second new test is failing - that is good! The fix is to create a conditional allowing the user to specify an input type. When modifying the tool and retesting - try passing the `--failed` flag to `planemo t` - it will speed things up by only rerunning tests that have already failed.

```
$ planemo t --failed
```

If you are comfortable with Galaxy tool development - try modifying the tool to make the failing test pass.

Hint:

- You will need to use the `data_collection` `param` type. It accepts many of the same attributes as `data` parameters (e.g. see `input_fastq1`) but you will need to specify a `collection_type` of `paired`.
- To access the `data_collection` parameter parts in the `command` block - use `$collection_param.forward` and `$collection_param.reverse`.

Once you get the new test case passing with the `--failed` parameter - try running all the tests again to ensure you didn't break the original test.

```
$ planemo t
... bunch of output ...
bwa_mem_test[0]: passed
bwa_mem_test[1]: passed
```

One possible implementation for tests is as follows (sections with changes highlighted).

```
<?xml version="1.0"?>
<tool id="bwa_mem_test" name="Map with BWA-MEM" version="0.0.1">
    <description>- map medium and long reads</description>
    <requirements>
        <requirement type="package" version="0.7.15">bwa</requirement>
        <requirement type="package" version="1.3">samtools</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
      ## Build reference
      #set $reference_fasta_filename = "localref.fa"
      ln -s "${ref_file}" "${reference_fasta_filename}" &&
      bwa index -a is "${reference_fasta_filename}" &&

      ## Begin BWA-MEM command line
      bwa mem
      -t "\${GALAXY_SLOTS:-4}"
      -v 1                                                                              ##␣
↪Verbosity is set to 1 (errors only)

      "${reference_fasta_filename}"
      #set $input_type = $input_type_conditional.input_type
      #if $input_type == "single"
      "${input_type_conditional.fastq_input1}"
      #elif $input_type == "paired_collection"
      "${input_type_conditional.fastq_input.forward}" "${input_type_conditional.fastq_
↪input.reverse}"
      #end if
      | samtools view -Sb - > temporary_bam_file.bam &&
      samtools sort -o ${bam_output} temporary_bam_file.bam
    ]]></command>

    <inputs>
        <param name="ref_file" type="data" format="fasta" label="Use the following␣
↪dataset as the reference sequence" help="You can upload a FASTA sequence to the␣
↪history and use it as reference" />
        <conditional name="input_type_conditional">
            <param name="input_type" type="select" label="Input Type">
                <option value="single" selected="true">Single Dataset</option>
                <option value="paired_collection">Paired Collection</option>
            </param>
            <when value="single">
                 <param name="fastq_input1" type="data" format="fastqsanger" label=
↪"Select fastq dataset" help="Specify dataset with single reads"/>
            </when>
            <when value="paired_collection">
                 <param name="fastq_input" format="fastqsanger" type="data_collection"␣
↪collection_type="paired" label="Select dataset pair" help="Specify paired dataset␣
↪collection containing paired reads"/>
            </when>
        </conditional>
    </inputs>
```

(continues on next page)

(continued from previous page)

```
    <outputs>
        <data format="bam" name="bam_output" label="${tool.name} on ${on_string} (mapped␣
↪reads in BAM format)"/>
    </outputs>

    <tests>
        <!-- header describing command-line will always be different -
             hence lines_diff="2" on output tag. -->
        <test>
            <param name="fastq_input1" value="bwa-mem-fastq1.fq" />
            <param name="ref_file" value="bwa-mem-mt-genome.fa" />
            <output name="bam_output" file="bwa-aln-test1.bam" ftype="bam" lines_diff="2
↪" />
        </test>
        <test>
            <param name="fastq_input">
              <collection type="paired">
                <element name="forward" value="bwa-mem-fastq1.fq" />
                <element name="reverse" value="bwa-mem-fastq2.fq" />
              </collection>
            </param>
            <param name="ref_file" value="bwa-mem-mt-genome.fa" />
            <param name="input_type" value="paired_collection" />
            <output name="bam_output" file="bwa-aln-test2.bam" ftype="bam" lines_diff="2
↪" />
        </test>
    </tests>
    <help>

**BWA MEM options**

Algorithm options::

       -k INT        minimum seed length [19]
       -w INT        band width for banded alignment [100]
       -d INT        off-diagonal X-dropoff [100]
       -r FLOAT      look for internal seeds inside a seed longer than {-k} * FLOAT [1.5]
       -y INT        find MEMs longer than {-k} * {-r} with size less than INT [0]
       -c INT        skip seeds with more than INT occurrences [500]
       -D FLOAT      drop chains shorter than FLOAT fraction of the longest overlapping␣
↪chain [0.50]
       -W INT        discard a chain if seeded bases shorter than INT [0]
       -m INT        perform at most INT rounds of mate rescues for each read [50]
       -S            skip mate rescue
       -P            skip pairing; mate rescue performed unless -S also in use
       -e            discard full-length exact matches

Scoring options::

       -A INT        score for a sequence match, which scales options -TdBOELU unless␣
↪overridden [1]
```

(continues on next page)

(continued from previous page)

```
       -B INT        penalty for a mismatch [4]
       -O INT[,INT]  gap open penalties for deletions and insertions [6,6]
       -E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
       -L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
       -U INT        penalty for an unpaired read pair [17]

Input/output options::

       -p            first query file consists of interleaved paired-end sequences
       -R STR        read group header line such as '@RG\tID:foo\tSM:bar' [null]

       -v INT        verbose level: 1=error, 2=warning, 3=message, 4+=debugging [3]
       -T INT        minimum score to output [30]
       -h INT        if there are &lt;INT hits with score &gt;80% of the max score,␣
↪output all in XA [5]
       -a            output all alignments for SE or unpaired PE
       -C            append FASTA/FASTQ comment to SAM output
       -V            output the reference FASTA header in the XR tag
       -Y            use soft clipping for supplementary alignments
       -M            mark shorter split hits as secondary

       -I FLOAT[,FLOAT[,INT[,INT]]]
                     specify the mean, standard deviation (10% of the mean if absent),␣
↪max
                     (4 sigma from the mean if absent) and min of the insert size␣
↪distribution.
                     FR orientation only. [inferred]
    </help>
    <citations>
      <citation type="doi">10.1093/bioinformatics/btp698</citation>
    </citations>
</tool>
```

Note: **Exercise:** The devteam mappers allow users to specify both a paired collection or individual datasets (i.e. using two `data` parameters). Extend the above `conditional` to allow that. Remember to write your test case first and make sure it fails.

Hint: You should not require additional inputs or outputs to do this.

Adding More Parameters

Next up, let's add some of BWA's optional parameters to our tool - these parameters are outlined in the example tool's `help` section. To speed this up and demonstrate another feature of Galaxy - the next test will test the command-line generated by Galaxy instead of the exact outputs. Not requiring a complete set of outputs for each test case is convenient because it can speed development and allows testing more parameter combinations. There are certain tools and certain parameters where exact outputs are impossible to pre-determine though.

Lets start with `algorithm` parameter``-k INT minimum seed length [19]``. Again, lets do a test first!

```
<test>
    <param name="fastq_input1" value="bwa-mem-fastq1.fq" />
    <param name="ref_file" value="bwa-mem-mt-genome.fa" />
    <param name="set_algorithm_params" value="true" />
    <param name="k" value="20" />
    <assert_command>
        <has_text text="-k 20" />
    </assert_command>
</test>
```

Continuing our pattern - let's ensure this new test fails before implementing the `k` parameter.

```
$ planemo t
... bunch of output ...
bwa_mem_test[0]: passed
bwa_mem_test[1]: passed
bwa_mem_test[2]: failed
```

Reviewing the output - indeed this new test failed as expected (*did not contain expected text '-k 20'*). Now let's implement the `k` parameter and use `planemo t --failed` to ensure our implementation is correct.

An example tool with this test and passing.

```
<?xml version="1.0"?>
<tool id="bwa_mem_test" name="Map with BWA-MEM" version="0.0.1">
    <description>- map medium and long reads</description>
    <requirements>
        <requirement type="package" version="0.7.15">bwa</requirement>
        <requirement type="package" version="1.3">samtools</requirement>
    </requirements>
    <command detect_errors="exit_code"><![CDATA[
      ## Build reference
      #set $reference_fasta_filename = "localref.fa"
      ln -s "${ref_file}" "${reference_fasta_filename}" &&
      bwa index -a is "${reference_fasta_filename}" &&

      ## Begin BWA-MEM command line
      bwa mem
      -t "\${GALAXY_SLOTS:-4}"
      -v 1                                                                        ##␣
↪Verbosity is set to 1 (errors only)

      #if $algorithm.set_algorithm_params
      -k ${algorithm.k}
      #end if

      "${reference_fasta_filename}"
      #set $input_type = $input_type_conditional.input_type
      #if $input_type == "single"
      "${input_type_conditional.fastq_input1}"
      #elif $input_type == "paired_collection"
      "${input_type_conditional.fastq_input.forward}" "${input_type_conditional.fastq_
↪input.reverse}"
      #end if
```

(continues on next page)

(continued from previous page)

```
      | samtools view -Sb - > temporary_bam_file.bam &&
      samtools sort -o ${bam_output} temporary_bam_file.bam
    ]]></command>

    <inputs>
        <param name="ref_file" type="data" format="fasta" label="Use the following␣
↪dataset as the reference sequence" help="You can upload a FASTA sequence to the␣
↪history and use it as reference" />
        <conditional name="input_type_conditional">
            <param name="input_type" type="select" label="Input Type">
                <option value="single" selected="true">Single Dataset</option>
                <option value="paired_collection">Paired Collection</option>
            </param>
            <when value="single">
                 <param name="fastq_input1" type="data" format="fastqsanger" label=
↪"Select fastq dataset" help="Specify dataset with single reads"/>
            </when>
            <when value="paired_collection">
                 <param name="fastq_input" format="fastqsanger" type="data_collection"␣
↪collection_type="paired" label="Select a paired collection" help="Specify paired␣
↪dataset collection containing paired reads"/>
            </when>
        </conditional>
        <conditional name="algorithm">
            <param name="set_algorithm_params" type="boolean" label="Set Algorithm␣
↪Parameters">
            </param>
            <when value="true">
                <param argument="-k" label="minimum seed length" type="integer" value="19
↪" />
            </when>
            <when value="false">
            </when>
        </conditional>
    </inputs>

    <outputs>
        <data format="bam" name="bam_output" label="${tool.name} on ${on_string} (mapped␣
↪reads in BAM format)"/>
    </outputs>

    <tests>
        <!-- header describing command-line will always be different -
             hence lines_diff="2" on output tag. -->
        <test>
            <param name="fastq_input1" value="bwa-mem-fastq1.fq" />
            <param name="ref_file" value="bwa-mem-mt-genome.fa" />
            <output name="bam_output" file="bwa-aln-test1.bam" ftype="bam" lines_diff="2
↪" />
        </test>
        <test>
            <param name="fastq_input">
```

(continues on next page)

(continued from previous page)

```
              <collection type="paired">
                <element name="forward" value="bwa-mem-fastq1.fq" />
                <element name="reverse" value="bwa-mem-fastq2.fq" />
              </collection>
            </param>
            <param name="ref_file" value="bwa-mem-mt-genome.fa" />
            <param name="input_type" value="paired_collection" />
            <output name="bam_output" file="bwa-aln-test2.bam" ftype="bam" lines_diff="2
↪" />
        </test>
        <test>
            <param name="fastq_input1" value="bwa-mem-fastq1.fq" />
            <param name="ref_file" value="bwa-mem-mt-genome.fa" />
            <param name="set_algorithm_params" value="true" />
            <param name="k" value="20" />
            <assert_command>
                <has_text text="-k 20" />
            </assert_command>
         </test>
    </tests>
    <help>

**BWA MEM options**

Algorithm options::

       -k INT        minimum seed length [19]
       -w INT        band width for banded alignment [100]
       -d INT        off-diagonal X-dropoff [100]
       -r FLOAT      look for internal seeds inside a seed longer than {-k} * FLOAT [1.5]
       -y INT        find MEMs longer than {-k} * {-r} with size less than INT [0]
       -c INT        skip seeds with more than INT occurrences [500]
       -D FLOAT      drop chains shorter than FLOAT fraction of the longest overlapping␣
↪chain [0.50]
       -W INT        discard a chain if seeded bases shorter than INT [0]
       -m INT        perform at most INT rounds of mate rescues for each read [50]
       -S            skip mate rescue
       -P            skip pairing; mate rescue performed unless -S also in use
       -e            discard full-length exact matches

Scoring options::

       -A INT        score for a sequence match, which scales options -TdBOELU unless␣
↪overridden [1]
       -B INT        penalty for a mismatch [4]
       -O INT[,INT]  gap open penalties for deletions and insertions [6,6]
       -E INT[,INT]  gap extension penalty; a gap of size k cost '{-O} + {-E}*k' [1,1]
       -L INT[,INT]  penalty for 5'- and 3'-end clipping [5,5]
       -U INT        penalty for an unpaired read pair [17]

Input/output options::
```

(continues on next page)

(continued from previous page)

```
       -p            first query file consists of interleaved paired-end sequences
       -R STR        read group header line such as '@RG\tID:foo\tSM:bar' [null]

       -v INT        verbose level: 1=error, 2=warning, 3=message, 4+=debugging [3]
       -T INT        minimum score to output [30]
       -h INT        if there are &lt;INT hits with score &gt;80% of the max score,␣
↪output all in XA [5]
       -a            output all alignments for SE or unpaired PE
       -C            append FASTA/FASTQ comment to SAM output
       -V            output the reference FASTA header in the XR tag
       -Y            use soft clipping for supplementary alignments
       -M            mark shorter split hits as secondary

       -I FLOAT[,FLOAT[,INT[,INT]]]
                     specify the mean, standard deviation (10% of the mean if absent),␣
↪max
                     (4 sigma from the mean if absent) and min of the insert size␣
↪distribution.
                     FR orientation only. [inferred]
    </help>
    <citations>
      <citation type="doi">10.1093/bioinformatics/btp698</citation>
    </citations>
</tool>
```

The tool also demonstrates the new `argument` option on `param` tag. These work a lot like specifying a parameter `name` argument - but Galaxy will describe the underlying application argument in the GUI and API - which may be helpful for power users and external applications.

Exercise 1: Implement a few more algorithm parameters and start another `Scoring` section. Extend the above test case as you go.

Exercise 2: Extend the first test case to verify by default none of these parameters are present in the command. Use the `not_has_text` tag to do this (e.g. `<not_has_text text="-k 20">`).

Exercise 3: Publish the bwa-mem to the local Tool Shed following the procedure described in the tutorial. (Don't forget to alter the commands from the used `seqtk` example to `bwa-mem`.)

Hint:

```
$ planemo shed_init --name=bwa-bwa \
                    --owner=planemo \
                    --description=bwa-mem \
                    --long_description="BWA MEM: Long and medium read mapper" \
                    --category="Next Gen Mappers"
```

Note: A full list of the current assertion elements like these that are allowed can be found on the tool syntax page.

In additon to the assertion-based testing of the command, the jobs standard output and standard error can be checked using `assert_stdout` and `assert_stderr` respectively - paralleling the `assert_command` tag.

See the sample tool job_properties.xml for an example of this.

5.3.2 Multiple Output Files

Tools which create more than one output file are common. There are several different methods to accommodate this need. Each one of these has their advantages and weaknesses; careful thought should be employed to determine the best method for a particular tool.

Static Multiple Outputs

Handling cases when tools create a static number of outputs is simple. Simply include an `<output>` tag for each output desired within the tool XML file:

```xml
<tool id="example_tool" name="Multiple output" description="example">
    <command>example_tool.sh '$input1' $tool_option1 '$output1' '$output2'</command>
    <inputs>
        ...
    </inputs>
    ...
    <outputs>
        <data format="interval" name="output1" metadata_source="input1" />
        <data format="pdf" name="output2" />
    </outputs>
</tool>
```

Static Outputs Determinable from Inputs

In cases when the number of output files varies, but can be determined based upon a user's parameter selection, the filter tag can be used. The text contents of the `<filter>` tag are `eval``ed and if the expression is ``True` a dataset will be created as normal. If the expression is `False`, the output dataset will not be created; instead a `NoneDataset` object will be created and made available. When used on the command line the text `None` will appear instead of a file path. The local namespace of the filter has been populated with the tool parameters.

```xml
<tool id="example_tool" name="Multiple output" description="example">
    <command>example_tool.sh '$input1' $tool_option1 '$output1' '$output2'</command>
    <inputs>
       ...
       <param name="tool_option1" type="select" label="Type of output">
           <option value="1">Single File</option>
           <option value="2">Two Files</option>
       </param>
       <conditional name="condition1">
           <param name="tool_option2" type="select" label="Conditional output">
               <option value="yes">Yes</option>
               <option value="no">No</option>
           </param>
           ...
       </condition>
       ...
    </inputs>
    ...
    <outputs>
        <data format="interval" name="output1" metadata_source="input1" />
        <data format="pdf" name="output2" >
```

(continues on next page)

(continued from previous page)

```
            <filter>tool_option1 == "2"</filter>
        </data>
        <data format="txt" name="output3" >
            <filter>condition1['tool_option2'] == "yes"</filter>
        </data>
    </outputs>
</tool>
```

The command line generated when `tool_option1` is set to `Single File` is:

```
example_tool.sh input1_FILE_PATH 1 output1_FILE_PATH None
```

The command line generated when `tool_option1` is set to `Two Files` is:

```
example_tool.sh input1_FILE_PATH 2 output1_FILE_PATH output2_FILE_PATH
```

The datatype of an output can be determined by conditional parameter settings as in tools/filter/pasteWrapper.xml

```
<outputs>
    <data format="input" name="out_file1" metadata_source="input1">
        <change_format>
            <when input_dataset="input1" attribute="ext" value="bed" format="interval"/>
        </change_format>
    </data>
</outputs>
```

Single HTML Output

There are times when a single history item is desired, but this history item is composed of multiple files which are only useful when considered together. This is done by having a single (`primary`) output and storing additional files in a directory (single-level) associated with the primary dataset.

A common usage of this strategy is to have the primary dataset be an HTML file and then store additional content (reports, pdfs, images, etc) in the dataset extra files directory. The content of this directory can be referenced using relative links within the primary (HTML) file, clicking on the eye icon to view the dataset will display the HTML page.

If you want to wrap or create a tool that generates an HTML history item that shows the user links to a number of related output objects (files, images..), you need to know where to write the objects and how to reference them when your tool generates HTML which gets written to the HTML file. Galaxy will not write that HTML for you at present.

The FastQC wrapper is an existing tool example where the Java application generates HTML and image outputs but these need to be massaged to make them Galaxy friendly. In other cases, the application or your wrapper must take care of all the fiddly detailed work of writing valid html to display to the user. In either situation, the `html` datatype offers a flexible way to display very complex collections of related outputs inside a single history item or to present a complex html page generated by an application. There are some things you need to take care of for this to work:

The following example demonstrates declaring an output of type `html`.

```
<outputs>
    <data format="html" name="html_file" label="myToolOutput_${tool_name}.html">
</outputs>
```

The application or script must be set up to write all the output files and/or images to a new special subdirectory passed as a command line parameter from Galaxy every time the tool is run. The paths for images and other files will end up looking something like `$GALAXY_ROOT/database/files/000/dataset_56/img1.jpg` when you prepend the Galaxy provided path to the filenames you want to use. The command line must pass that path to your script and it is specified using the `extra_files_path` property of the HTML file output.

For example:

```
<command>myscript.pl '$input1' '$html_file' '$html_file.extra_files_path' </command>
```

The application must create and write valid html to setup the page `$html_file` seen by the user when they view (eye icon) the file. It must create and write that new file at the path passed by Galaxy as the `$html_file` command line parameter. All application outputs that will be included as links in that html code should be placed in the specific output directory `$html_file.extra_files_path` passed on the command line. The external application is responsible for creating that directory before writing images and files into it. When generating the html, The files written by the application to `$html_file.extra_files_path` are referenced in links directly by their name, without any other path decoration - eg:

```
<a href="file1.xls">Some special output</a>
<br/>
<img src="image1.jpg" >
```

The (now unmaintained) Galaxy Tool Factory includes code to gather all output files and create a page with links and clickable PDF thumbnail images which may be useful as a starting point (e.g. see rgToolFactory2.py.

`galaxy.datatypes.text.Html` (the `html` datatype) is a subclass of composite datasets so new subclasses of composite can be used to implement even more specific structured outputs but this requires adding the new definition to Galaxy - whereas Html files require no extension of the core framework. For more information visit Composite Datatypes.

Dynamic Numbers of Outputs

This section discusses the case where the number of output datasets cannot be determined until the tool run is complete. If the outputs can be broken into groups or collections of similar/homogenous datasets - this is possibly a case for using dataset collections. If instead the outputs should be treated individually and Galaxy's concept of dataset collections doesn't map cleanly to the outputs - Galaxy can "discover" individual output datasets dynamically after the job is complete.

Collections

See the Planemo documentation on creating collections for more details on this topic.

A blog post on generating dataset collections from tools can be found here.

Individual Datasets

There are times when the number of output datasets varies entirely based upon the content of an input dataset and the user needs to see all of these outputs as new individual history items rather than as a collection of datasets or a group of related files linked in a single new HTML page in the history. Tools can optionally describe how to "discover" an arbitrary number of files that will be added after the job's completion to the user's history as new datasets. Whenever possible, one of the above strategies should be used instead since these discovered datasets cannot be used with workflows and require the user to refresh their history before they are shown.

Discovering datasets (arbitrarily) requires a fixed "parent" output dataset to key on - this dataset will act as the reference for our additional datasets. Sometimes the parent dataset that should be used makes sense from context but in instances where one does not readily make sense tool authors can just create an arbitrary text output (like a report of the dataset generation).

Each discovered dataset requires a unique "designation" (used to describe functional tests, the default output name, etc. . .) and should be located in the job's working direcotry or a sub-directory thereof. Regular expressions are used to describe how to discover the datasets and (though not required) a unique such pattern should be specified for each homogeneous group of such files.

Examples

Consider a tool that creates a bunch of text files or bam files and writes them (with extension that matches the Galaxy datatype - e.g. `txt` or `bam` to the `split` sub-directory of the working directory. Such outputs can be discovered by adding the following block of XML to your tool description:

```
<outputs>
    <data name="report" format="txt">
        <discover_datasets pattern="__designation_and_ext__" directory="split" visible=
↪"true" />
    </data>
</outputs>
```

So for instance, if the tool creates 4 files (in addition to the report) such as `split/samp1.bam`, `split/samp2.bam`, `split/samp3.bam`, and `split/samp4.bam` - then 4 discovered datasets will be created of type `bam` with designations of `samp1`, `samp2`, `samp3`, and `samp4`.

If the tool doesn't create the files in `split` with extensions or does but with extensions that do not match Galaxy's datatypes - a slightly different pattern can be used and the extension/format can be statically specified (here either `ext` or `format` may be used as the attribute name):

```
<outputs>
    <data name="report" format="txt">
        <discover_datasets pattern="__designation__" format="tabular" directory="tables"␣
↪visible="true" />
    </data>
</outputs>
```

So in this example, if the tool creates 3 tabular files such as `tables/part1.tsv`, `tables/part2.tsv`, and `tables/part3.tsv` - then 3 discovered datasets will be created of type `tabular` with designations of `part1.tsv`, `part2.tsv`, and `part3.tsv`.

It may not be desirable for the extension/format (`.tsv`) to appear in the `designation` this way. These patterns `__designation__` and `__designation_and_ext__` are replaced with regular expressions that capture metadata from the file name using named groups. A tool author can explicitly define these regular expressions instead of using these shortcuts - for instance `__designation__` is just `(?P<designation>.*)` and `__designation_and_ext__` is `(?P<designation>.*)\.(?P<ext>[^\._]+)?`. So the above example can be modified as:

```
<outputs>
    <data name="report" format="txt">
        <discover_datasets pattern="(?P&lt;designation&gt;.+)\.tsv" format="tabular"␣
↪directory="tables" visible="true" />
    </data>
</outputs>
```

As a result - three datasets are still be captured - but this time with designations of `part1`, `part2`, and `part3`.

Notice here the < and > in the tool pattern had to be replaced with `\&lt;` and `&gt;` to be properly embedded in XML (this is very ugly - apologies).

The metadata elements that can be captured via regular expression named groups this way include `ext`, `designation`, `name`, `dbkey`, and `visible`. Each pattern must declare at least either a `designation` or a `name` - the other metadata parts `ext`, `dbkey`, and `visible` are all optional and may also be declared explicitly in via attributes on the `discover_datasets` element (as shown in the above examples).

For tools which do not define a `profile` version or define one before 16.04, if no `discover_datasets` element is nested with a tool output - Galaxy will still look for datasets using the named pattern `__default__` which expands to `primary_DATASET_ID_(?P<designation>[^_]+)_(?P<visible>[^_]+)_(?P<ext>[^_]+)(_(?P<dbkey>[^_]+))?`. Many tools use this mechanism as it traditionally was the only way to discover datasets and has the nice advantage of not requiring an explicit declaration and encoding everything (including the output to map to) right in the name of the file itself.

For instance consider the following output declaration:

```
<outputs>
    <data format="interval" name="output1" metadata_source="input1" />
</outputs>
```

If `$output1.id` (accessible in the tool `command` block) is `546` and the tool (likely a wrapper) produces the files `primary_546_output2_visible_bed` and `primary_546_output3_visible_pdf` in the job's working directory - then after execution is complete these two additional datasets (a `bed` file and a `pdf` file) are added to the user's history.

Newer tool profile versions disable this and require the tool author to be more explicit about what files are discovered.

More information

- Example tools which demonstrate discovered datasets:
  - multi_output.xml
  - multi_output_assign_primary.xml
  - multi_output_configured.xml
- Original pull request for discovered dataset enhancements with implementation details
- Implementation of output collection code in galaxy

Legacy information

In the past, it would be necessary to set the attribute `force_history_refresh` to `True` to force the user's history to fully refresh after the tool run has completed. This functionality is now broken and `force_history_refresh` is ignored by Galaxy. Users now **MUST** manually refresh their history to see these files. A Trello card used to track the progress on fixing this and eliminating the need to refresh histories in this manner can be found [[https://trello.com/c/f5Ddv4CS/1993-history-api-determine-history-state-running-from-join-on-running-jobs|here]].

Discovered datasets are available via post job hooks (a deprecated feature) by using the designation - e.g. `__collected_datasets__['primary'][designation]`.

In the past these datasets were typically written to `$__new_file_path__` instead of the working directory. This is not very scalable and `$__new_file_path__` should generally not be used. If you set the option `collect_outputs_from` in `galaxy.ini` ensure `job_working_directory` is listed as an option (if not the only option).

5.3.3 Collections

Galaxy has a concept of dataset collections to group together datasets and operate over them as a single unit.

Galaxy collections are hierarchical and composed from two collection types - `list` and `paired`.

- A **list** is a collection of datasets (or other collections) where each element has an `identifier`. Unlike Galaxy dataset names which are transformed throughout complex analyses - the `identifier` is generally preserved and can be used for concepts such as `sample` name that one wants to preserve in the earlier mapping steps of a workflow and use it during reduction steps and reporting later.
- The **paired** collection type is much simpler and more specific to sequencing applications. Each `paired` collection consists of a `forward` and `reverse` dataset.

Note: Read more about creating and managing collections.

Composite types include for instance the `list:paired` collection type - which represents a list of dataset pairs. In this case, instead of each dataset having a list idenifier, each pair of datasets does.

Consuming Collections

Many Galaxy tools can be used without modification in conjunction with collections. Galaxy users can take a collection and *map over* any tool that consumes individual datasets. For instance, early in typical bioinformatics workflows you may have steps that filter raw data, convert to standard formats, perform QC on individual files - users can take lists, pairs, or lists of paired datasets and map over such tools that consume individual dataset (files). Galaxy will then run the tool once for each dataset in the collection and for each output of that tool Galaxy will rebuild a new collection.

Collection elements have the concept an *identifier* and an *index* when the collection is created. Both of these are preserved during these mapping steps. As Galaxy builds output collections from these mapping steps, the identifier and index for the output entries match those of the supplied input.

If a tool's functionality can be applied to individual files in isolation, the implicit mapping described above should be sufficient and no knowledge of collections by tools should be needed. However, tools may need to process multiple files at once - in this case explicit collection consumption is required. This document outlines three cases:

- consuming pairs of datasets
- consuming lists
- consuming arbitrary collections.

Note: If you find yourself consuming a collection of files and calling the underlying application multiple times within the tool command block, you are likely doing something wrong. Just process a pair or a single dataset and allow the user to map over the collection.

Processing Pairs

Dataset collections are not extensively used by typical Galaxy users yet - so for tools which process paired datasets the recommended best practice is to allow users to either supply paired collections or two individual datasets. Furthermore, many tools which process pairs of datasets can also process single datasets. The following `conditional` captures this idiom.

```
<conditional name="fastq_input">
  <param name="fastq_input_selector" type="select" label="Single or Paired-end reads"␣
↪help="Select between paired and single end data">
    <option value="paired">Paired</option>
    <option value="single">Single</option>
    <option value="paired_collection">Paired Collection</option>
    <option value="paired_iv">Paired Interleaved</option>
  </param>
  <when value="paired">
    <param name="fastq_input1" type="data" format="fastqsanger" label="Select first set␣
↪of reads" help="Specify dataset with forward reads"/>
    <param name="fastq_input2" type="data" format="fastqsanger" label="Select second set␣
↪of reads" help="Specify dataset with reverse reads"/>
  </when>
  <when value="single">
    <param name="fastq_input1" type="data" format="fastqsanger" label="Select fastq␣
↪dataset" help="Specify dataset with single reads"/>
  </when>
  <when value="paired_collection">
    <param name="fastq_input" format="fastqsanger" type="data_collection" collection_
↪type="paired" label="Select a paired collection" label="Select dataset pair" help=
↪"Specify paired dataset collection containing paired reads"/>
  </when>
</conditional>
```

This introduces a new `param` type - `data_collection`. The optional attribute `collection_type` can specify which kinds of collections are appropriate for this input. Additional `data` attributes such as `format` can further restrict valid collections. Here we defined that both items of the paired collection must be of datatype `fastqsanger`.

In Galaxy's `command` block, the individual datasets can be accessed using `$fastq_input1.forward` and `$fastq_input1.reverse`. If processing arbitrary collection types an array syntax can also be used (e.g. `$fastq_input['forward']`).

Note: Mirroring the ability of Galaxy users to map tools that consume individual datasets over lists (and other collection types), users may also map lists of pairs over tools which explicitly consume dataset pair.

If the output of the tool is datasets, the output of this mapping operation (sometimes referred to as subcollection mapping) will be lists. The element identifier and index of the top level of the list will be preserved.

Some example tools which consume paired datasets include:

- collection_paired_test (minimal test tool in Galaxy test suite)
- Bowtie 2
- BWA MEM
- Tophat

Processing Lists (Reductions)

The data_collection parameter type can specify a collection_type or list but whenever possible, it is recommended to not explicitly consume lists as a tool author. Parameters of type data can include a multiple="True" attribute to allow many datasets to be selected simultaneously. While the default UI will then have Galaxy users pick individual datasets, they can choose a collections as the tool can process both. This has the benefit of allowing tools to process either individual datasets or collections. A noteworthy difference is that if a parameter of type data with multiple="true" is used, the elements of the collection are passed to the tool as a (Python) list, i.e. it is not possible:

- to find out if a collection was passed,
- to access properties of the collection (name,. . .), or
- to write tests that pass a collection to the parameter (which would allow to name the elements explicitly).

Another drawback is that the *${on_string}* of the label contains the list of data sets in the collection (which can be confusing, since these data sets are in most cases hidden) and not the name of the collection.

```
<param type="data" name="inputs" label="Input BAM(s)" format="bam" multiple="true" />
```

The command tag can use for loops to build command lines using these parameters.

For instance:

```
#for $input in $inputs
--input "$input"
#end for
```

or using the single-line form of this expression:

```
#for $input in $inputs# $input #end for#
```

Will produce command strings with an argument for each input (e.g. --input "/path/to/input1" --input "/path/to/input2"). Other programs may require all inputs to be supplied in a single parameter. This can be accomplished using the idiom:

```
--input "${",".join(map(str, $inputs))}"
```

Some example tools which consume multiple datasets (including lists) include:

- multi_data_param (small test tool in Galaxy test suite)
- cuffmerge tool macros
- unionBedGraphs

Also see the tools-devteam repository Pull Request #20 modifying the cufflinks suite of tools for collection compatible reductions.

Processing Identifiers

Collection elements have identifiers that can be used for various kinds of sample tracking. These identifiers are set when the collection is first created - either explicitly in the UI (or API), through mapping over collections that preserves input identifiers, or as the `identifier` when dynamically discovering collection outputs described below.

During reduction steps one may likely want to use these - for reporting, comparisons, etc. When using these multiple `data` parameters the dataset objects expose a field called `element_identifier`. When these parameters are used with individual datasets - this will just default to being the dataset's name, but when used with collections this parameter will be the `element_identifier` (i.e. the preserved sample name).

For instance, imagine merging a collection of tabular datasets into a single table with a new column indicating the sample name the corresponding rows were derived from using a little fictitious program called `merge_rows`.

```
#import re
#for $input in $inputs
merge_rows --name "${re.sub('[^\w\-_]', '_', $input.element_identifier)}" --file "$input
↪" --to $output;
#end for
```

Note: Here we are rewriting the element identifiers to assure everything is safe to put on the command-line. In the future, collections will not be able to contain keys that are potentially harmful and this won't be necessary.

Some example tools which utilize `element_identifier` include:

- identifier_multiple
- identifier_single
- vcftools_merge
- jbrowse
- kraken-mpa-report

More on `data_collection` parameters

The above three cases (users mapping over single tools, consuming pairs, and consuming lists using *multiple* `data` parameters) are hopefully the most common ways to consume collections for a tool author - but the `data_collection` parameter type allows one to handle more cases than just these.

We have already seen that in `command` blocks `data_collection` parameters can be accessed as arrays by element identifier (e.g. `$input_collection["left"]`). This applies for lists and higher-order structures as well as pairs. The valid element identifiers can be iterated over using the `keys` method.

```
#for $key in $input_collection.keys()
--input_name $key
--input $input_collection[$key]
#end for
```

```
#for $input in $input_collection
--input $input
#end for
```

Importantly, the `keys` method and direct iteration are both strongly ordered. If you take a list of files, do a bunch of processing on them to produce another list, and then consume both collections in a tools - the elements will match up if iterated over simultaneously.

Finally, if processing arbitrarily nested collections - one can access the `is_collection` attribute to determine if a given element is another collection or just a dataset.

```
#for $input in $input_collection
--nested ${input.is_collection}
#end for
```

Some example tools which consume nested collections include:

- collection_nested_test (small test tool demonstrating consumption of nested collections)

Creating Collections

Whenever possible simpler operations that produce datasets should be implicitly "mapped over" to produce collections as described above - but there are a variety of situations for which this idiom is insufficient.

Progressively more complex syntax elements exist for the increasingly complex scenarios. Broadly speaking - the three scenarios covered are when the tool produces…

1. a collection with a static number of elements (mostly for `paired` collections, but if a tool has fixed binding it might make sense to create a list this way as well)
2. a `list` with the same number of elements as an input list (this would be a common pattern for normalization applications for instance).
3. a `list` where the number of elements is not knowable until the job is complete.

1. Static Element Count

For this first case - the tool can declare standard data elements below an output collection element in the outputs tag of the tool definition.

```
<collection name="paired_output" type="paired" label="Split Pair">
    <data name="forward" format="txt" />
    <data name="reverse" format_source="input1" from_work_dir="reverse.txt" />
</collection>
```

Templates (e.g. the `command` tag) can then reference `$forward` and `$reverse` or whatever `name` the corresponding `data` elements are given as demonstrated in `test/functional/tools/collection_creates_pair.xml`.

The tool should describe the collection type via the type attribute on the collection element. Data elements can define `format`, `format_source`, `metadata_source`, `from_work_dir`, and `name`.

The above syntax would also work for the corner case of static lists. For paired collections specifically however, the type plugin system now knows how to prototype a pair so the following even easier (though less configurable) syntax works.

```
<collection name="paired_output" type="paired" label="Split Pair" format_source="input1">
</collection>
```

In this case the command template could then just reference `${paried_output.forward}` and `${paired_output.reverse}` as demonstrated in `test/functional/tools/collection_creates_pair_from_type.xml`.

2. Computable Element Count

For the second case - where the structure of the output is based on the structure of an input - a structured_like attribute can be defined on the collection tag.

```
<collection name="list_output" type="list" label="Duplicate List" structured_like="input1
↪" inherit_format="true" />
```

Templates can then loop over `input1` or `list_output` when building up command-line expressions. See `test/functional/tools/collection_creates_list.xml` for an example.

`format`, `format_source`, and `metadata_source` can be defined for such collections if the format and metadata are fixed or based on a single input dataset. If instead the format or metadata depends on the formats of the collection it is structured like - `inherit_format="true"` and/or `inherit_metadata="true"` should be used instead - which will handle corner cases where there are for instance subtle format or metadata differences between the elements of the incoming list.

3. Dynamic Element Count

The third and most general case is when the number of elements in a list cannot be determined until runtime. For instance, when splitting up files by various dynamic criteria.

In this case a collection may define one of more discover_dataset elements. As an example of one such tool that splits a tabular file out into multiple tabular files based on the first column see `test/functional/tools/collection_split_on_column.xml` - which includes the following output definition:

```
<collection name="split_output" type="list" label="Table split on first column">
    <discover_datasets pattern="__name_and_ext__" directory="outputs" />
</collection>
```

Nested Collections

Galaxy Pull Request #538 implemented the ability to define nested output collections. See the pull request and included example tools for more details.

Further Reading

- Galaxy Community Conference Talk by John Chilton [Slides][Video].
- Creating and Managing Collections
- Pull Request #386 (the initial implementation)
- Pull Request #634 (implementing ability for tools to explicitly output collections)

5.3.4 Macros - Reusable Elements

Frequently, tools may require the same XML fragments be repeated in a file (for instance similar conditional branches, repeated options, etc…) or between tools in the same repository (for instance, nearly all of the GATK tools contain the same standard options). Galaxy tools have a macroing system to address this problem.

Direct XML Macros

The following examples are taken from Pull Request 129 the initial implementation of macros. Prior to to the inclusion of macros, the tophat2 wrapper defined several outputs each which had the following identical actions block associated with them:

```
<actions>
  <conditional name="refGenomeSource.genomeSource">
    <when value="indexed">
      <action type="metadata" name="dbkey">
        <option type="from_data_table" name="tophat2_indexes" column="1" offset="0">
          <filter type="param_value" column="0" value="#" compare="startswith" keep=
↪"False"/>
          <filter type="param_value" ref="refGenomeSource.index" column="0"/>
        </option>
      </action>
    </when>
    <when value="history">
      <action type="metadata" name="dbkey">
        <option type="from_param" name="refGenomeSource.ownFile" param_attribute="dbkey"␣
↪/>
      </action>
    </when>
  </conditional>
</actions>
```

To reuse this action definition, first a macros section has been defined in the tophat2_wrpper.xml file.

```
<tool>
   ...
   <macros>
     <xml name="dbKeyActions">
       <action><!-- Whole big example above. -->
         ....
       </action>
     </xml>
   </macros>
```

With this in place, each output data element can include this block using the expand XML element as follows:

```
<outputs>
    <data format="bed" name="insertions" label="${tool.name} on ${on_string}: insertions
↪" from_work_dir="tophat_out/insertions.bed">
        <expand macro="dbKeyActions" />
    </data>
    <data format="bed" name="deletions" label="${tool.name} on ${on_string}: deletions"␣
↪from_work_dir="tophat_out/deletions.bed">
```

(continues on next page)

(continued from previous page)

```
        <expand macro="dbKeyActions" />
    </data>
    <data format="bed" name="junctions" label="${tool.name} on ${on_string}: splice␣
↪junctions" from_work_dir="tophat_out/junctions.bed">
        <expand macro="dbKeyActions" />
    </data>
    <data format="bam" name="accepted_hits" label="${tool.name} on ${on_string}:␣
↪accepted_hits" from_work_dir="tophat_out/accepted_hits.bam">
        <expand macro="dbKeyActions" />
    </data>
</outputs>
```

This has reduced the size of the XML file by dozens of lines and reduces the long term maintenance associated with copied and pasted code.

Imported Macros

The `macros` element described above, can also contain any number of `import` elements. This allows a directory/repository of tool XML files to contain shared macro definitions that can be used by any number of actual tool files in that directory/repository.

Revisiting the tophat example, all three tophat wrappers (`tophat_wrapper.xml`, `tophat_color_wrapper.xml`, and `tophat2_wrapper.xml`) shared some common functionality. To reuse XML elements between these files, a `tophat_macros.xml` file was added to that directory.

The following block is a simplified version of that macros file's contents:

```
<macros>
  <xml name="own_junctionsConditional">
    <conditional name="own_junctions">
      <param name="use_junctions" type="select" label="Use Own Junctions">
        <option value="No">No</option>
        <option value="Yes">Yes</option>
      </param>
      <when value="Yes">
        <conditional name="gene_model_ann">
          <param name="use_annotations" type="select" label="Use Gene Annotation Model">
            <option value="No">No</option>
            <option value="Yes">Yes</option>
          </param>
          <when value="No" />
          <when value="Yes">
            <param format="gtf,gff3" name="gene_annotation_model" type="data" label=
↪"Gene Model Annotations" help="TopHat will use the exon records in this file to build␣
↪a set of known splice junctions for each gene, and will attempt to align reads to␣
↪these junctions even if they would not normally be covered by the initial mapping."/>
          </when>
        </conditional>
        <expand macro="raw_juncsConditional" />
        <expand macro="no_novel_juncsParam" />
      </when>
      <when value="No" />
```

(continues on next page)

(continued from previous page)

```
    </conditional> <!-- /own_junctions -->
  </xml>
  <xml name="raw_juncsConditional">
    <conditional name="raw_juncs">
      <param name="use_juncs" type="select" label="Use Raw Junctions">
        <option value="No">No</option>
        <option value="Yes">Yes</option>
      </param>
      <when value="No" />
      <when value="Yes">
        <param format="interval" name="raw_juncs" type="data" label="Raw Junctions" help=
↪"Supply TopHat with a list of raw junctions. Junctions are specified one per line, in␣
↪a tab-delimited format. Records look like: [chrom] [left] [right] [+/-] left and right␣
↪are zero-based coordinates, and specify the last character of the left sequenced to be␣
↪spliced to the first character of the right sequence, inclusive."/>
      </when>
    </conditional>
  </xml>
  <xml name="no_novel_juncsParam">
    <param name="no_novel_juncs" type="select" label="Only look for supplied junctions">
      <option value="No">No</option>
      <option value="Yes">Yes</option>
    </param>
  </xml>
</macros>
```

Any tool definition in that directory can use the macros contained therein once imported as shown below.

```
<tool>
  ...
  <macros>
    <import>tophat_macros.xml</import>
  </macros>
  ...
  <inputs>
    <expand macro="own_junctionsConditional" />
    ...
  </inputs>
  ...
</tool>
```

This example also demonstrates that macros may themselves expand macros.

Parameterizing XML Macros (with `yield`)

In some cases, tools may contain similar though not exact same definitions. Some parameterization can be performed by declaring expand elements with child elements and expanding them in the macro definition with a yield element.

For instance, previously the tophat wrapper contained the following definition:

```
<conditional name="refGenomeSource">
  <param name="genomeSource" type="select" label="Will you select a reference genome␣
↪from your history or use a built-in index?" help="Built-ins were indexed using default␣
↪options">
    <option value="indexed">Use a built-in index</option>
    <option value="history">Use one from the history</option>
  </param>
  <when value="indexed">
    <param name="index" type="select" label="Select a reference genome" help="If your␣
↪genome of interest is not listed, contact the Galaxy team">
      <options from_data_table="tophat_indexes_color">
        <filter type="sort_by" column="2"/>
        <validator type="no_options" message="No indexes are available for the selected␣
↪input dataset"/>
      </options>
    </param>
  </when>
  <when value="history">
    <param name="ownFile" type="data" format="fasta" metadata_name="dbkey" label="Select␣
↪the reference genome" />
  </when>  <!-- history -->
</conditional>  <!-- refGenomeSource -->
```

and the tophat2 wrapper contained the highly analogous definition:

```
<conditional name="refGenomeSource">
  <param name="genomeSource" type="select" label="Will you select a reference genome␣
↪from your history or use a built-in index?" help="Built-ins were indexed using default␣
↪options">
    <option value="indexed">Use a built-in index</option>
    <option value="history">Use one from the history</option>
  </param>
  <when value="indexed">
    <param name="index" type="select" label="Select a reference genome" help="If your␣
↪genome of interest is not listed, contact the Galaxy team">
      <options from_data_table="tophat2_indexes_color">
        <filter type="sort_by" column="2"/>
        <validator type="no_options" message="No indexes are available for the selected␣
↪input dataset"/>
      </options>
    </param>
  </when>
  <when value="history">
    <param name="ownFile" type="data" format="fasta" metadata_name="dbkey" label="Select␣
↪the reference genome" />
  </when>  <!-- history -->
</conditional>  <!-- refGenomeSource -->
```

These blocks differ only in the from_data_table attribute on the options element. To capture this pattern, tophat_macros.xml contains the following macro definition:

```
<xml name="refGenomeSourceConditional">
  <conditional name="refGenomeSource">
    <param name="genomeSource" type="select" label="Use a built in reference genome or␣
↪own from your history" help="Built-ins genomes were created using default options">
      <option value="indexed" selected="True">Use a built-in genome</option>
      <option value="history">Use a genome from history</option>
    </param>
    <when value="indexed">
      <param name="index" type="select" label="Select a reference genome" help="If your␣
↪genome of interest is not listed, contact the Galaxy team">
        <yield />
      </param>
    </when>
    <when value="history">
      <param name="ownFile" type="data" format="fasta" metadata_name="dbkey" label=
↪"Select the reference genome" />
    </when>  <!-- history -->
  </conditional>  <!-- refGenomeSource -->
</xml>
```

Notice the yield statement in lieu of an options declaration. This allows the nested options element to be declared when expanding the macro:

The following expand declarations have replaced the original conditional elements.

```
<expand macro="refGenomeSourceConditional">
  <options from_data_table="tophat_indexes">
    <filter type="sort_by" column="2"/>
    <validator type="no_options" message="No genomes are available for the selected␣
↪input dataset"/>
  </options>
</expand>
```

```
<expand macro="refGenomeSourceConditional">
  <options from_data_table="tophat2_indexes">
    <filter type="sort_by" column="2"/>
    <validator type="no_options" message="No genomes are available for the selected␣
↪input dataset"/>
  </options>
</expand>
```

From Galaxy release 22.01 named yields are supported. That is, if the macro contains `<yield name="xyz"/>` it is replaced by the content of the `token` element with the same name. Token elements need to be direct children of the `expand` element. This is useful if different parts of the macro should be parametrized.

In the following example two named yield and one unnamed yield are used to parametrize the options of the select of a conditional, the options of another select, and additional when block(s). Given the following macro:

```
<xml name="named_yields_example">
  <conditional>
    <param type="select">
      <option value="a">A</option>
```

(continues on next page)

(continued from previous page)

```
      <option value="b">B</option>
      <yield name="more_options"/>
    </param>
    <when value="a">
      <param name="aselect" type="select">
        <yield />
      </param>
    </when>
    <when value="b"/>
    <yield name="more_whens">
  </conditional>
</xml>
```

and expanding the macro in the following way:

```
<expand macro="named_yields_example">
  <token name="more_options">
    <option value="c">C</option>
  </token>
  <token name="more_whens">
    <when value="c">
      <param type="select">
        <yield />
      </param>
    </when>
  </token>
  <options from_data_table="tophat2_indexes">
    <filter type="sort_by" column="2"/>
    <validator type="no_options" message="No genomes are available for the selected␣
↪input dataset"/>
  </options>
</expand>
```

we get the following expanded definition:

```
<xml name="named_yields_example">
  <conditional>
    <param type="select">
      <option value="a">A</option>
      <option value="b">B</option>
      <option value="c">C</option>
    </param>
    <when value="a">
      <param name="aselect" type="select">
        <options from_data_table="tophat2_indexes">
          <filter type="sort_by" column="2"/>
          <validator type="no_options" message="No genomes are available for the␣
↪selected input dataset"/>
        </options>
      </param>
    </when>
    <when value="b"/>
```

(continues on next page)

(continued from previous page)

```
    <when value="c">
      <param type="select">
        <options from_data_table="tophat2_indexes">
          <filter type="sort_by" column="2"/>
          <validator type="no_options" message="No genomes are available for the␣
↪selected input dataset"/>
        </options>
      </param>
    </when>
  </conditional>
</xml>
```

Named yields are replaced in the order of the tokens defined in the `expand` tag. Unnamed yields are replaced after all named tokens have been replaced (by the non-token child elements of the expand tag). If there are named yields that have no corresponding token, then they are treated like unnamed yields. Note that unnamed and named tokens can be used multiple times in a macro, then each occurrence is replaced by the corresponding content defined in the `expand`.

Further, note that the order of the replacements offers some possibilities to achieve recursion-like replacements, since a token may contain further named or unnamed `yield` tags (see for instance the `yield` tag contained in the named token `more_whens`).

Parameterizing XML Macros (with tokens)

In addition to using `yield` blocks, there is another way to parametrize macros by specifying:

- required parameters as comma-separated list of parameter names using the `tokens` attribute (e.g. `tokens="foo, bar"`) of the `xml` element and then using `@FOO@` and `@BAR@` in the macro definition;
- optional parameters as `token_xyz="default_value"` attributes of the `xml` element, and then using `@XYZ@` in the macro definition.

```
<macros>
  <xml name="color" tokens="varname" token_default_color="#00ff00" token_label="Pick a␣
↪color">
      <param name="@VARNAME@" type="color" label="@LABEL@" value="@DEFAULT_COLOR@" />
  </xml>
</macros>
```

This defines a macro with a required parameter `varname` and two optional parameters `default_color` and `label`. When invoking this macro, you can pass values for those parameters and produce varying results.

```
<inputs>
    <expand macro="color" varname="myvar" default_color="#ff0000" />
    <expand macro="color" varname="c2" default_color="#0000ff" label="Choose a different␣
↪color" />
</inputs>
```

The attributes passed to the macro definition will be filled in (or defaults used if not provided). Effectively this yields:

```
<inputs>
    <param name="myvar" type="color" label="Pick a color" value="#ff0000" />
    <param name="c2" type="color" label="Choose a different color" value="#0000ff" />
</inputs>
```

Macro tokens can be used in the text content of tags, attribute values, and (with a little trick also in attribute names). The problem is that the default delimiting character of macro tokens is `@` and the XML must still be valid before processing the macros (and `@` is invalid in attribute names). Luckily the delimiting character(s) can be changed by adding `token_quote` to the macro definition:

```
<macros>
  <xml name="color" tokens="attr,attr_value" token_quote="__" token_label="label">
      <param __ATTR__="__ATTR_VALUE__" label="__LABEL__"/>
  </xml>
</macros>
```

Note that, this forbids to use tokens with the name `quote`.

Macro Tokens

You can use

```
<token name="@IS_PART_OF_VCFLIB@">is a part of VCFlib toolkit developed by Erik Garrison␣
↪(https://github.com/ekg/vcflib).</token>
```

and then call the token within any element of the file like this:

```
Vcfallelicprimitives @IS_PART_OF_VCFLIB@
```

5.3.5 Tool Provided Metadata

This stub of a section provides some initial documentation on tool provided metadata. Galaxy allows datasets to be discovered after a tool has been executed and allows tools to specify metadata for these datasets. Whenever possible, Galaxy's datatypes and more structured outputs should be used to collect metadata.

If an arbitrary number of outputs is needed but no special metadata must be set, file name patterns can be used to allow Galaxy to discover these datasets. More information on this can be found in the *dedicated section*.

The file name patterns described in the above link are nice because they don't need special instrumenting in the tool wrapper to adapt to Galaxy in general and can adapt to many existing application's output. When more metadata must be supplied or when implementing a custom tool wrapper anyway - it may be beneficial to build a manifest file.

A tool may also produce a file called `galaxy.json` during execution. If upon a job's completion this file is populated, Galaxy will expect to find metadata about outputs in it.

The format of this file is a bit quirky - each line of this file should be a JSON dictionary. Each such dictionary should contain a `type` attribute - this type may be either `new_primary_dataset` or `dataset`.

If the `type` is `new_primary_dataset`, the dictionary should contain a `filename` entry with a path to a "discovered dataset". In this case the dictionary may contain any of the following entries `name`, `dbkey`, `info`, `ext`, `metadata`.

- `name` will be used as the output dataset's name
- `ext` allows specification of the format of the output (e.g. `txt`, `tabular`, `fastqsanger`, etc...)
- `dbkey` allows specifying a genome build for the discovered dataset
- `info` is a short text description for each dataset that appears in the history panel
- `metadata` this should be a dictionary of key-value pairs for metadata registered with the datatype for this output

Examples of tools using `new_primary_dataset` entries:

- tool_provided_metadata_2.xml demonstrating using the simpler attributes described here.
- tool_provided_metadata_3.xml demonstrates overridding datatype specified metadata.

The `type` of an entry may also be `dataset`. In this case the metadata descriptions describe an explicit output (one with its own corresponding `output` element definition). In this case, an entry called `dataset` should appear in the dictionary (in lieu of `filename` above) and should be the database id of the output dataset. Such entries may contain all of the other fields described above except `metadata`.

Example tool using a `dataset` entry:

- tool_provided_metadata_1.xml

5.3.6 Cluster Usage

Developing for Clusters - `GALAXY_SLOTS`, `GALAXY_MEMORY_MB`, and `GALAXY_MEMORY_MB_PER_SLOT`

`GALAXY_SLOTS` is a special environment variable that is set in a Galaxy tool's runtime environment. If the tool you are working on allows configuring the number of processes or threads that should be spawned, this variable should be used.

For example, for the StringTie tool the binary `stringtie` can take an argument `-p` that allows specification of the number of threads to be used. The Galaxy tool sets this up as follows

```
stringtie "$input_bam" -o "$output_gtf" -p "\${GALAXY_SLOTS:-1}" ...
```

Here we use `\${GALAXY_SLOTS:-Z}` instead of a fixed value (Z being an integer representing a default value in non-Galaxy contexts). The backslash here is because this value is interpreted at runtime as environment variable - not during command building time as a templated value. Now server administrators can configure how many processes the tool should be allowed to use.

For information on how server administrators can configure this value for a particular tool, check out the Galaxy admin documentation.

Analogously `GALAXY_MEMORY_MB` and `GALAXY_MEMORY_MB_PER_SLOT` are special environment variables in a Galaxy tool's runtime environment that can be used to specify the amount of memory that a tool can use overall and per slot, respectively.

For an example see the samtools sort tool which allows to specify the total memory with the -m parameter.

Test Against Clusters - `--job_config_file`

The various commands that start Galaxy servers (`serve`, `test`, `shed_serve`, `shed_test`, etc…) allow specification of a Galaxy job configuration XML file (e.g. `job_conf.xml`).

For instance, Slurm is a popular distributed reource manager (DRM) in the Galaxy community. The following `job_conf.xml` tells Galaxy to run all jobs using Slurm and allocate 2 cores for each job.

```
<?xml version="1.0"?>
<job_conf>
    <plugins>
        <plugin id="drmaa" type="runner" load="galaxy.jobs.runners.drmaa:DRMAAJobRunner"␣
↪/>
    </plugins>
    <handlers>
        <handler id="main"/>
```

(continues on next page)

(continued from previous page)

```
    </handlers>
    <destinations default="drmaa">
        <destination id="drmaa" runner="drmaa">
          <param id="nativeSpecification">--time=00:05:00 --nodes=1 --ntasks=2</param>
        </destination>
    </destinations>
</job_conf>
```

If this file is named `planemo_job_conf.xml` and resides in one's home directory (~), Planemo can `test` or `serve` using this configuration with the following commands.

```
$ planemo test --job_config_file ~/planemo_job_conf.xml .
$ planemo serve --job_config_file ~/planemo_job_conf.xml .
```

For general information on configuring Galaxy to communicate with clusters check out this page on the Galaxy wiki and for information regarding configuring job configuration XML files in particular check out the example distributed with Galaxy.

5.3.7 Dependencies and Conda

Specifying and Using Tool Requirements

Note: This document discusses using Conda to satisfy tool dependencies from a tool developer perspective. An in depth discussion of using Conda to satisfy dependencies from an admistrator's perspective can be found here. That document also serves as good background for this discussion.

Note: Planemo requires a Conda installation to target with its various Conda related commands. A properly configured Conda installation can be initialized with the `conda_init` command. This should only need to be executed once per development machine.

```
$ planemo conda_init
galaxy.tools.deps.conda_util INFO: Installing conda, this may take several minutes.
wget -q --recursive -O /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/conda_
↪installLW5zn1.sh https://repo.continuum.io/miniconda/Miniconda3-4.3.31-MacOSX-x86_64.sh
bash /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/conda_installLW5zn1.sh -b -p /
↪Users/john/miniconda3
PREFIX=/Users/john/miniconda3
installing: python-3.6.3-h47c878a_7 ...
Python 3.6.3 :: Anaconda, Inc.
installing: ca-certificates-2017.08.26-ha1e5d58_0 ...
installing: conda-env-2.6.0-h36134e3_0 ...
installing: libcxxabi-4.0.1-hebd6815_0 ...
installing: tk-8.6.7-h35a86e2_3 ...
installing: xz-5.2.3-h0278029_2 ...
installing: yaml-0.1.7-hc338f04_2 ...
installing: zlib-1.2.11-hf3cbc9b_2 ...
installing: libcxx-4.0.1-h579ed51_0 ...
installing: openssl-1.0.2n-hdbc3d79_0 ...
installing: libffi-3.2.1-h475c297_4 ...
```

(continues on next page)

(continued from previous page)

```
installing: ncurses-6.0-hd04f020_2 ...
installing: libedit-3.1-hb4e282d_0 ...
installing: readline-7.0-hc1231fa_4 ...
installing: sqlite-3.20.1-h7e4c145_2 ...
installing: asn1crypto-0.23.0-py36h782d450_0 ...
installing: certifi-2017.11.5-py36ha569be9_0 ...
installing: chardet-3.0.4-py36h96c241c_1 ...
installing: idna-2.6-py36h8628d0a_1 ...
installing: pycosat-0.6.3-py36hee92d8f_0 ...
installing: pycparser-2.18-py36h724b2fc_1 ...
installing: pysocks-1.6.7-py36hfa33cec_1 ...
installing: python.app-2-py36h54569d5_7 ...
installing: ruamel_yaml-0.11.14-py36h9d7ade0_2 ...
installing: six-1.11.0-py36h0e22d5e_1 ...
installing: cffi-1.11.2-py36hd3e6348_0 ...
installing: setuptools-36.5.0-py36h2134326_0 ...
installing: cryptography-2.1.4-py36h842514c_0 ...
installing: wheel-0.30.0-py36h5eb2c71_1 ...
installing: pip-9.0.1-py36h1555ced_4 ...
installing: pyopenssl-17.5.0-py36h51e4350_0 ...
installing: urllib3-1.22-py36h68b9469_0 ...
installing: requests-2.18.4-py36h4516966_1 ...
installing: conda-4.3.31-py36_0 ...
installation finished.
/Users/john/miniconda3/bin/conda install -y --override-channels --channel iuc --channel␣
↪conda-forge --channel bioconda --channel defaults conda=4.3.33 conda-build=2.1.18
Fetching package metadata ...................
Solving package specifications: .

Package plan for installation in environment /Users/john/miniconda3:

The following NEW packages will be INSTALLED:

    beautifulsoup4: 4.6.0-py36_0   conda-forge
    conda-build:    2.1.18-py36_0 conda-forge
    conda-verify:   2.0.0-py36_0   conda-forge
    filelock:       3.0.4-py36_0   conda-forge
    jinja2:         2.10-py36_0    conda-forge
    markupsafe:     1.0-py36_0     conda-forge
    pkginfo:        1.4.2-py36_0   conda-forge
    pycrypto:       2.6.1-py36_1   conda-forge
    pyyaml:         3.12-py36_1    conda-forge

The following packages will be UPDATED:

    conda:          4.3.31-py36_0             --> 4.3.33-py36_0 conda-forge

beautifulsoup4 100% |###################################################################
↪#| Time: 0:00:00 782.08 kB/s
filelock-3.0.4 100% |###################################################################
↪#| Time: 0:00:00   7.95 MB/s
markupsafe-1.0 100% |###################################################################
```

(continues on next page)

(continued from previous page)

```
↪#| Time: 0:00:00   5.82 MB/s
pkginfo-1.4.2- 100% |#######################################################################
↪#| Time: 0:00:00   1.18 MB/s
pycrypto-2.6.1 100% |#######################################################################
↪#| Time: 0:00:00   1.69 MB/s
pyyaml-3.12-py 100% |#######################################################################
↪#| Time: 0:00:00   3.31 MB/s
conda-verify-2 100% |#######################################################################
↪#| Time: 0:00:00   6.91 MB/s
jinja2-2.10-py 100% |#######################################################################
↪#| Time: 0:00:00   2.81 MB/s
conda-4.3.33-p 100% |#######################################################################
↪#| Time: 0:00:00 621.27 kB/s
conda-build-2. 100% |#######################################################################
↪#| Time: 0:00:00   2.16 MB/s
Conda installation succeeded - Conda is available at '/Users/john/miniconda3/bin/conda'
```

While Galaxy can be configured to resolve dependencies various ways, Planemo is configured with opinionated defaults geared at making building tools that target Conda as easy as possible.

During the introductory tool development tutorial, we called `planemo tool_init` with the argument `--requirement seqtk@1.2` and the resulting tool contained the XML:

```
<requirements>
    <requirement type="package" version="1.2">seqtk</requirement>
</requirements>
```

As configured by Planemo, when Galaxy encounters these `requirement` tags it will attempt to install Conda, check for referenced packages (such as `seqtk`), and install them as needed for tool testing.

Note: *Why Conda?*

Many different package managers could potentially be targeted here, but we focus on Conda for a few key reasons.

- No compilation at install time - binaries with their dependencies and libraries
- Support for all operating systems
- Easy to manage multiple versions of the same recipe
- HPC-ready: no root privileges needed
- Easy-to-write YAML recipes
- Viberant communities

Note: **Conda Terminology**

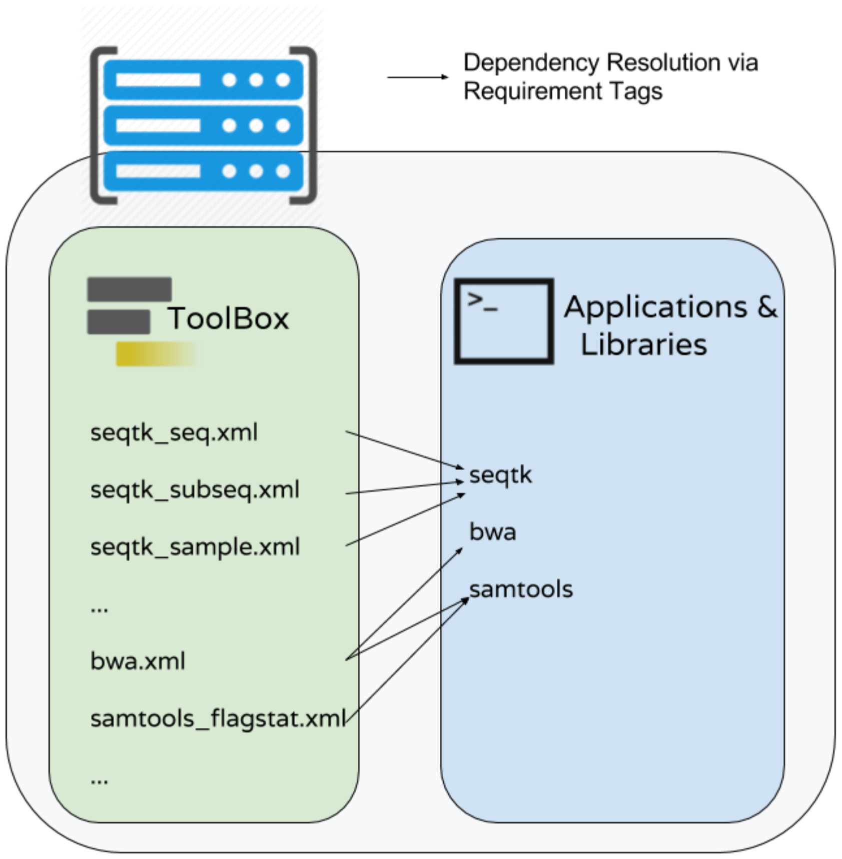
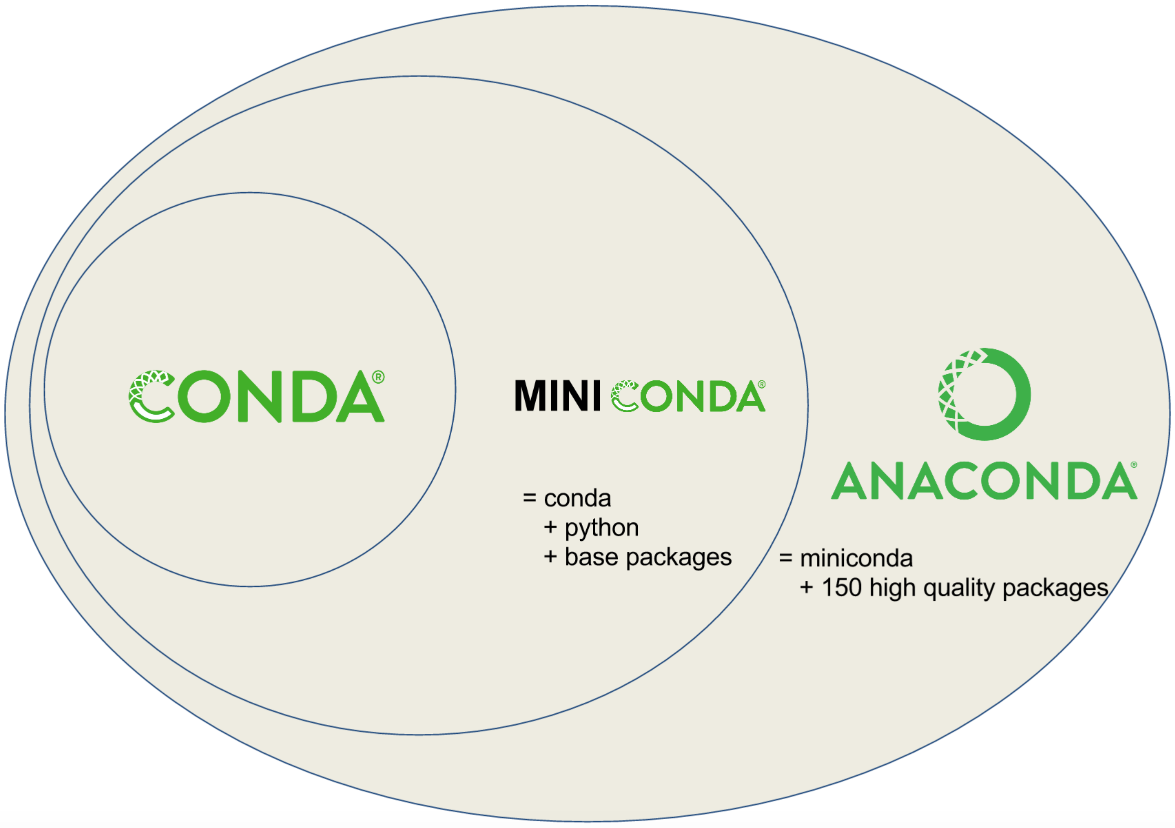


Fig. 1: Galaxy's dependency resolution maps tool requirement tags to concrete applications and libraries setup by the Galaxy deployer (or Planemo). As the above diagram indicates the same requirements may be used by multiple Galaxy tools and a single Galaxy tool may depend on multiple requirements. The document describes working with Conda dependencies from a developer perspective but other dependency resolution techniques are covered in the Galaxy docs.



Conda *recipes* build *packages* that are published to *channels*.

Planemo is setup to target a few channels by default, these include `iuc`, `bioconda`, `conda_forge`, `defaults` - the whole dependency management scheme outlined here works a lot better if packages can be found in one of these "best practice" channels.

We can check if the requirements on a tool are available in best practice Conda channels using an extended form of the `planemo lint` command. Passing `--conda_requirements` flag will ensure all listed requirements are found.

```
$ planemo lint --conda_requirements seqtk_seq.xml
Linting tool /Users/john/workspace/planemo/docs/writing/seqtk_seq.xml
  ...
Applying linter requirements_in_conda... CHECK
.. INFO: Requirement [seqtk@1.2] matches target in best practice Conda channel␣
↪[bioconda].
```

Note: You can download the final version of the seqtk seq wrapper from the Planemo tutorial using the command:

```
$ planemo project_init --template=seqtk_complete seqtk_example
$ cd seqtk_example
```

We can verify these tool requirements install with the `conda_install` command. With its default parameters `conda_install` processes tools and creates isolated environments for their declared requirements.

```
$ planemo conda_install seqtk_seq.xml
Install conda target CondaTarget[seqtk,version=1.2]
/home/john/miniconda2/bin/conda create -y --name __seqtk@1.2 seqtk=1.2
Fetching package metadata ...............
Solving package specifications: ..........

Package plan for installation in environment /home/john/miniconda2/envs/__seqtk@1.2:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    seqtk-1.2                  |                0          29 KB  bioconda

The following NEW packages will be INSTALLED:

    seqtk: 1.2-0   bioconda
    zlib:  1.2.8-3

Fetching packages ...
seqtk-1.2-0.ta 100% |###############################################################|␣
↪Time: 0:00:00 444.71 kB/s
Extracting packages ...
[      COMPLETE      ]|###################################################################
↪##############| 100%
Linking packages ...
[      COMPLETE      ]|###################################################################
↪##############| 100%
#
# To activate this environment, use:
# > source activate __seqtk@1.2
#
# To deactivate this environment, use:
# > source deactivate __seqtk@1.2
#
$ which seqtk
seqtk not found
$
```

The above install worked properly, but `seqtk` is not on your `PATH` because this merely created an environment within the Conda directory for the seqtk installation. Planemo will configure Galaxy to exploit this installation. If you wish to interactively explore the resulting enviornment to explore the installed tool or produce test data the output of the conda_env command can be sourced.

```
$ . <(planemo conda_env seqtk_seq.xml)
Deactivate environment with conda_env_deactivate
(seqtk_seq) $ which seqtk
/home/planemo/miniconda2/envs/
↪jobdepsiJClEUfecc6d406196737781ff4456ec60975c137e04884e4f4b05dc68192f7cec4656/bin/seqtk
(seqtk_seq) $ seqtk seq

Usage:   seqtk seq [options] <in.fq>|<in.fa>
```

(continues on next page)

(continued from previous page)

```
Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases
         -S        strip of white spaces in sequences
(seqtk_seq) $ conda_env_deactivate
$
```

As shown above the `conda_env_deactivate` will be created in this environment and can be used to restore your initial shell configuration.

Confident the underlying application works, we can now use `planemo test` or `planemo serve` and it will reuse this environment and find our dependency (in this case `seqtk` as needed).

Here is a portion of the output from the testing command `planemo test seqtk_seq.xml` demonstrating using this tool.

```
$ planemo test seqtk_seq.xml
...
2017-02-22 10:13:28,902 INFO  [galaxy.tools.actions] Handled output named output1 for␣
↪tool seqtk_seq (20.136 ms)
2017-02-22 10:13:28,914 INFO  [galaxy.tools.actions] Added output datasets to history␣
↪(12.782 ms)
2017-02-22 10:13:28,935 INFO  [galaxy.tools.actions] Verified access to datasets for␣
↪Job[unflushed,tool_id=seqtk_seq] (10.954 ms)
2017-02-22 10:13:28,936 INFO  [galaxy.tools.actions] Setup for job Job[unflushed,tool_
↪id=seqtk_seq] complete, ready to flush (21.053 ms)
2017-02-22 10:13:28,962 INFO  [galaxy.tools.actions] Flushed transaction for job␣
↪Job[id=2,tool_id=seqtk_seq] (26.510 ms)
2017-02-22 10:13:29,064 INFO  [galaxy.jobs.handler] (2) Job dispatched
2017-02-22 10:13:29,281 DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of␣
↪type conda
2017-02-22 10:13:29,282 DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of␣
↪type conda
2017-02-22 10:13:29,317 INFO  [galaxy.jobs.command_factory] Built script [/tmp/tmpLvKwta/
↪job_working_directory/000/2/tool_script.sh] for tool command [[ "$CONDA_DEFAULT_ENV" =
↪"/Users/john/miniconda2/envs/__seqtk@1.2" ] || . /Users/john/miniconda2/bin/activate '/
↪Users/john/miniconda2/envs/__seqtk@1.2' >conda_activate.log 2>&1 ; seqtk seq -A '/tmp/
↪tmpLvKwta/files/000/dataset_1.dat' > '/tmp/tmpLvKwta/files/000/dataset_2.dat']
```

(continues on next page)

(continued from previous page)

```
2017-02-22 10:13:29,516 DEBUG [galaxy.tools.deps] Using dependency samtools version None␣
↪of type conda
2017-02-22 10:13:29,516 DEBUG [galaxy.tools.deps] Using dependency samtools version None␣
↪of type conda
ok

----------------------------------------------------------------------
XML: /private/tmp/tmpLvKwta/xunit.xml
----------------------------------------------------------------------
Ran 1 test in 15.936s

OK
2017-02-22 10:13:37,014 INFO  [test_driver] Shutting down
2017-02-22 10:13:37,014 INFO  [test_driver] Shutting down embedded galaxy web server
2017-02-22 10:13:37,016 INFO  [test_driver] Embedded web server galaxy stopped
2017-02-22 10:13:37,017 INFO  [test_driver] Stopping application galaxy
....
2017-02-22 10:13:37,018 INFO  [galaxy.jobs.handler] sending stop signal to worker thread
2017-02-22 10:13:37,018 INFO  [galaxy.jobs.handler] job handler stop queue stopped
Testing complete. HTML report is in "/Users/john/workspace/planemo/project_templates/
↪seqtk_complete/tool_test_output.html".
All 1 test(s) executed passed.
seqtk_seq[0]: passed
```

In this case the tests passed and the line containing `[galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda` indicates Galaxy dependency resolution was successful and it found the environment we previously installed with `conda_install`.

Finding Existing Conda Packages

How did we know what software name and software version to use? We found the existing packages available for Conda and referenced them. To do this yourself, you can simply use the planemo command `conda_search`. If we do a search for `seqt` it will show all the software and all the versions available matching that search term - including `seqtk`.

```
$ planemo conda_search seqt
/Users/john/miniconda3/bin/conda search --override-channels --channel iuc --channel␣
↪conda-forge --channel bioconda --channel defaults '*seqt*'
Loading channels: done
# Name                  Version           Build  Channel
bioconductor-htseqtools          1.26.0       r3.4.1_0  bioconda
bioconductor-seqtools         1.10.0       r3.3.2_0  bioconda
bioconductor-seqtools         1.10.0       r3.4.1_0  bioconda
bioconductor-seqtools         1.12.0       r3.4.1_0  bioconda
seqtk                         r75              0  bioconda
seqtk                         r82              0  bioconda
seqtk                         r82              1  bioconda
seqtk                         r93              0  bioconda
seqtk                         1.2              0  bioconda
seqtk                         1.2              1  bioconda
```

Note: The Planemo command `conda_search` is a light wrapper around the underlying `conda search` command but configured to use the same channels and other options as Planemo and Galaxy. The following Conda command would also work to search:

```
$ $HOME/miniconda3/bin/conda -c iuc -c conda-forge -c bioconda '*seqt*'
```

For Conda versions 4.3.X or less, the search invocation would be something a bit different:

```
$ $HOME/miniconda3/bin/conda -c iuc -c conda-forge -c bioconda seqt
```

Alternatively the Anaconda website can be used to search for packages. Typing `seqtk` into the search form on that page and clicking the top result will bring on to this page with information about the Bioconda package.

When using the website to search though, you need to aware of what channel you are using. By default, Planemo and Galaxy will search a few different Conda channels. While it is possible to configure a local Planemo or Galaxy to target different channels - the current best practice is to add tools to the existing channels.

The existing channels include:

- Bioconda (github | conda) - best practice channel for various bioinformatics packages.
- Conda-Forge (github | conda) - best practice channel for general purpose and widely useful computing packages and libraries.
- iuc (github | conda) - best practice channel for other more Galaxy specific packages.

Exercise - Leveraging Bioconda

Use the `project_init` command to download this exercise.

```
$ planemo project_init --template conda_exercises conda_exercises
$ cd conda_exercises/exercise_1
$ ls
pear.xml              test-data
```

This project template contains a few exercises. The first uses an adapted version of an IUC tool for PEAR - Paired-End reAd mergeR. This tool however has no `requirement` tags and so will not work properly.

1. Run `planemo test pear.xml` to verify the tool does not function without dependencies defined.
2. Use `--conda_requirements` flag with `planemo lint` to verify it does indeed lack requirements.
3. Use `planemo conda_search` or the Anaconda website to search for the correct package and version in a best practice channel.
4. Update `pear.xml` with the correct `requirement` tags.
5. Re-run the `lint` command from above to verify the tool now has the correct dependency definition.
6. Re-run the `test` command from above to verify the tool test now works properly.

Building New Conda Packages

Frequently packages your tool will require are not found in Bioconda or conda-forge yet. In these cases, it is likely best to contribute your package to one of these projects. Unless the tool is exceedingly general Bioconda is usually the correct starting point.

Note: Many things that are not strictly or even remotely "bio" have been accepted into Bioconda - including tools for image analysis, natural language processing, and cheminformatics.

To get quickly learn to write Conda recipes for typical Galaxy tools, please read the following pieces of external documentation.

- Contributing to Bioconda in particular focusing on
  - One time setup
  - Contributing a recipe (through "Write a Recipe")
- Building conda packages in particular
  - Building conda packages with conda skeleton (the best approach for common scripting languages such as R and Python)
  - Building conda packages from scratch
  - Building conda packages for general code projects
  - Using conda build
- Then return to the Bioconda documentation and read
  - The rest of "Contributing a recipe" continuing from Testing locally
  - And finally Guidelines for bioconda recipes

These guidelines in particular can be skimmed depending on your recipe type, for instance that document provides specific advice for:

- Python
- R (CRAN)
- R (Bioconductor)
- Perl
- C/C++

To go a little deeper, you may want to read:

- Specification for meta.yaml
- Environment variables
- Custom channels

And finally to debug problems the Bioconda troubleshooting documentation may prove useful.

Exercise - Build a Recipe

If you have just completed the exercise above - this exercise can be found in parent folder. Get there with `cd ../exercise_2`. If not, the exercise can be downloaded with

```
$ planemo project_init --template conda_exercises conda_exercises
$ cd conda_exercises/exercise_2
$ ls
fleeqtk_seq.xml               test-data
```

This is the skeleton of a tool wrapping the parody bioinformatics software package fleeqtk. fleeqtk is a fork of the project seqtk that many Planemo tutorials are built around and the example tool should hopefully be fairly familiar. fleeqtk version 1.3 can be downloaded from here and built using `make`. The result of `make` includes a single executable `fleeqtk`.

1. Clone and branch Bioconda.
2. Build a recipe for fleeqtk version 1.3. You may wish to start from scratch (`conda skeleton` is not available for C programs like fleeqtk), or copy the recipe of seqtk and modify it for fleeqtk.
3. Use `conda build` or Bioconda tooling to build the recipe.
4. Run `planemo test --conda_use_local fleeqtk_seq.xml` to verify the resulting package works as expected.

Congratulations on writing a Conda recipe and building a package! Upon succesfully building and testing such a Bioconda package, you would normally push your branch to Github and open a pull request. This step is skipped here as to not pollute Bioconda with unneeded software packages.

5.3.8 Dependencies and Containers

For years Galaxy has supported running tools inside containers. The details of how to run Galaxy tools inside of containers varies depending on the Galaxy job runner but details can be found in Galaxy's job_conf.xml sample file.

This document doesn't describe how to run the containers, it describes how Galaxy figures out which container to run for a given tool. There are currently two strategies for finding containers for a tool - and they are each discussed in detail in this document. The newer approach is a bit more experimental but should be considered the best practice approach - it is to allow Galaxy to find or build a BioContainers container using `requirement` tags that resolve to best-practice Conda channels. The older approach is to explicitly declare a container identifier in the tool XML.

While not as flexible as resolving arbitrary image IDs from URLs, the newer approach has a few key advantages that make them a best practice:

- They provide superior reproducibility across Galaxy instances because the same binary Conda packages will automatically be used for both bare metal dependencies and inside containers.
- They are constructed automatically from existing Conda packages so tool developers shouldn't need to write `Dockerfile` s or register projects on Docker Hub.
- They are produced using mulled which produce very small containers that make deployment easy.
- Annotating `requirement` tags reduces the opaqueness of the Docker process. With this method it is entirely traceable how the container was constructed from what sources were fetched, which exact build of every dependency was used, to how packages in the container were built. Beyond that metadata about the packages can be fetched from Bioconda (e.g. this).

Read more about this reproducibility stack in our preprint Practical computational reproducibility in the life sciences.

BioContainers

Note: This section is a continuation of *Dependencies and Conda*, please review that section for background information on resolving requirements with Conda.

Finding BioContainers

If a tool contains requirements in best practice Conda channels, a BioContainers-style container can be found or built for it.

As reminder, `planemo lint --conda_requirements <tool.xml>` can be used to check if a tool contains only best-practice `requirement` tags. The `lint` command can also be fed the `--biocontainers` flag to check if a BioContainers container has been registered that is compatible with that tool.

Below is an example of using this with the completed `seqtk_seq.xml` tool from the introductory tutorial.

```
$ planemo lint --biocontainers seqtk_seq.xml
Linting tool /home/planemo/workspace/planemo/project_templates/seqtk_complete/seqtk_seq.
↪xml
Applying linter tests... CHECK
.. CHECK: 1 test(s) found.
Applying linter output... CHECK
.. INFO: 1 outputs found.
Applying linter inputs... CHECK
.. INFO: Found 9 input parameters.
Applying linter help... CHECK
.. CHECK: Tool contains help section.
.. CHECK: Help contains valid reStructuredText.
Applying linter general... CHECK
.. CHECK: Tool defines a version [0.1.0].
.. CHECK: Tool defines a name [Convert to FASTA (seqtk)].
.. CHECK: Tool defines an id [seqtk_seq].
.. CHECK: Tool targets 16.01 Galaxy profile.
Applying linter command... CHECK
.. INFO: Tool contains a command.
Applying linter citations... CHECK
.. CHECK: Found 1 likely valid citations.
Applying linter tool_xsd... CHECK
.. INFO: File validates against XML schema.
Applying linter biocontainer_registered... CHECK
.. INFO: BioContainer best-practice container found [quay.io/biocontainers/seqtk:1.2--0].
```

This last linter indicates that indeed a container has been registered that is compatible with this tool – `quay.io/biocontainers/seqtk:1.2--1`. We didn't do any extra work to build this container for this tool, all Bioconda recipes are packaged into containers and registered on quay.io as part of the BioContainers project.

This tool can be tested using `planemo test` in its BioContainer Docker container using the flag `--biocontainers` as shown below.

```
$ planemo test --biocontainers seqtk_seq.xml
...
2017-03-01 08:18:19,669 INFO  [galaxy.tools.actions] Handled output named output1 for␣
```

(continues on next page)

(continued from previous page)

```
↪tool seqtk_seq (22.145 ms)
2017-03-01 08:18:19,683 INFO  [galaxy.tools.actions] Added output datasets to history␣
↪(14.604 ms)
2017-03-01 08:18:19,703 INFO  [galaxy.tools.actions] Verified access to datasets for␣
↪Job[unflushed,tool_id=seqtk_seq] (8.687 ms)
2017-03-01 08:18:19,704 INFO  [galaxy.tools.actions] Setup for job Job[unflushed,tool_
↪id=seqtk_seq] complete, ready to flush (20.380 ms)
2017-03-01 08:18:19,719 INFO  [galaxy.tools.actions] Flushed transaction for job␣
↪Job[id=2,tool_id=seqtk_seq] (15.191 ms)
2017-03-01 08:18:20,120 INFO  [galaxy.jobs.handler] (2) Job dispatched
2017-03-01 08:18:20,311 DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of␣
↪type conda
2017-03-01 08:18:20,312 DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of␣
↪type conda
2017-03-01 08:18:20,325 INFO  [galaxy.tools.deps.containers] Checking with container␣
↪resolver [ExplicitDockerContainerResolver[]] found description [None]
2017-03-01 08:18:20,468 INFO  [galaxy.tools.deps.containers] Checking with container␣
↪resolver [CachedMulledDockerContainerResolver[namespace=None]] found description [None]
2017-03-01 08:18:20,881 INFO  [galaxy.tools.deps.containers] Checking with container␣
↪resolver [MulledDockerContainerResolver[namespace=biocontainers]] found description␣
↪[ContainerDescription[identifier=quay.io/biocontainers/seqtk:1.2--0,type=docker]]
2017-03-01 08:18:20,904 INFO  [galaxy.jobs.command_factory] Built script [/tmp/tmpw8_UQm/
↪job_working_directory/000/2/tool_script.sh] for tool command [seqtk seq -A '/tmp/tmpw8_
↪UQm/files/000/dataset_1.dat' > '/tmp/tmpw8_UQm/files/000/dataset_2.dat']
2017-03-01 08:18:21,060 DEBUG [galaxy.tools.deps] Using dependency samtools version None␣
↪of type conda
2017-03-01 08:18:21,061 DEBUG [galaxy.tools.deps] Using dependency samtools version None␣
↪of type conda
ok

----------------------------------------------------------------------
XML: /private/tmp/tmpw8_UQm/xunit.xml
----------------------------------------------------------------------
Ran 1 test in 11.926s

OK
2017-03-01 08:18:26,726 INFO  [test_driver] Shutting down
...
2017-03-01 08:18:26,732 INFO  [galaxy.jobs.handler] job handler stop queue stopped
Testing complete. HTML report is in "/home/planemo/workspace/planemo/tool_test_output.
↪html".
All 1 test(s) executed passed.
seqtk_seq[0]: passed
$
```

A very important line here is:

```
2017-03-01 08:18:20,881 INFO  [galaxy.tools.deps.containers] Checking with container␣
↪resolver [MulledDockerContainerResolver[namespace=biocontainers]] found description␣
↪[ContainerDescription[identifier=quay.io/biocontainers/seqtk:1.2--0,type=docker]]
```

This line indicates that Galaxy was able to find a container for this tool and executed the tool in that container.

For interactive testing, the `planemo serve` command also implements the `--biocontainers` flag.

Building BioContainers

In this seqtk example the relevant BioContainer already existed on quay.io, this won't always be the case. For tools that contain multiple `requirement` tags an existing container likely won't exist. The mulled toolkit (distributed with planemo or available standalone) can be used to build containers for such tools. For such tools, if Galaxy is configured to use BioContainers it will attempt to build these containers on the fly by default (though this behavior can be disabled).

You can try it directly using the `mull` command in Planemo. The `conda_testing` Planemo project template has a toy example tool with two requirements for demonstrating this - bwa_and_samtools.xml.

```
$ planemo project_init --template=conda_testing conda_testing
$ cd conda_testing/
$ planemo mull bwa_and_samtools.xml
/Users/john/.planemo/involucro -v=3 -f /Users/john/workspace/planemo/.venv/lib/python2.7/
↪site-packages/galaxy_lib-17.9.0-py2.7.egg/galaxy/tools/deps/mulled/invfile.lua -set␣
↪CHANNELS='iuc,conda-forge,bioconda,defaults' -set TEST='true' -set TARGETS='samtools=1.
↪3.1,bwa=0.7.15' -set REPO='quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820' -
↪set BINDS='build/dist:/usr/local/' -set PREINSTALL='conda install --quiet --yes␣
↪conda=4.3' build
/Users/john/.planemo/involucro -v=3 -f /Users/john/workspace/planemo/.venv/lib/python2.7/
↪site-packages/galaxy_lib-17.9.0-py2.7.egg/galaxy/tools/deps/mulled/invfile.lua -set␣
↪CHANNELS='iuc,conda-forge,bioconda,defaults' -set TEST='true' -set TARGETS='samtools=1.
↪3.1,bwa=0.7.15' -set REPO='quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820' -
↪set BINDS='build/dist:/usr/local/' -set PREINSTALL='conda install --quiet --yes␣
↪conda=4.3' build
[Jun 19 11:28:35] DEBU Run file [/Users/john/workspace/planemo/.venv/lib/python2.7/site-
↪packages/galaxy_lib-17.9.0-py2.7.egg/galaxy/tools/deps/mulled/invfile.lua]
[Jun 19 11:28:35] STEP Run image [continuumio/miniconda:latest] with command [[rm -rf /
↪data/dist]]
[Jun 19 11:28:35] DEBU Creating container [step-730a02d79e]
[Jun 19 11:28:35] DEBU Created container [5e4b5f83c455 step-730a02d79e], starting it
[Jun 19 11:28:35] DEBU Container [5e4b5f83c455 step-730a02d79e] started, waiting for␣
↪completion
[Jun 19 11:28:36] DEBU Container [5e4b5f83c455 step-730a02d79e] completed with exit code␣
↪[0] as expected
[Jun 19 11:28:36] DEBU Container [5e4b5f83c455 step-730a02d79e] removed
[Jun 19 11:28:36] STEP Run image [continuumio/miniconda:latest] with command [[/bin/sh -
↪c conda install --quiet --yes conda=4.3 && conda install  -c iuc -c conda-forge -c␣
↪bioconda -c defaults samtools=1.3.1 bwa=0.7.15 -p /usr/local --copy --yes --quiet]]
[Jun 19 11:28:36] DEBU Creating container [step-e95bf001c8]
[Jun 19 11:28:36] DEBU Created container [72b9ca0e56f8 step-e95bf001c8], starting it
[Jun 19 11:28:37] DEBU Container [72b9ca0e56f8 step-e95bf001c8] started, waiting for␣
↪completion
[Jun 19 11:28:46] SOUT Fetching package metadata .........
[Jun 19 11:28:47] SOUT Solving package specifications: .
[Jun 19 11:28:50] SOUT
[Jun 19 11:28:50] SOUT Package plan for installation in environment /opt/conda:
[Jun 19 11:28:50] SOUT
[Jun 19 11:28:50] SOUT The following packages will be UPDATED:
```

(continues on next page)

(continued from previous page)

```
[Jun 19 11:28:50] SOUT
[Jun 19 11:28:50] SOUT conda: 4.3.11-py27_0 --> 4.3.22-py27_0
[Jun 19 11:28:50] SOUT
[Jun 19 11:29:04] SOUT Fetching package metadata .................
[Jun 19 11:29:06] SOUT Solving package specifications: .
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:56] SOUT Package plan for installation in environment /usr/local:
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:56] SOUT The following NEW packages will be INSTALLED:
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:56] SOUT bwa:        0.7.15-1      bioconda
[Jun 19 11:29:56] SOUT curl:       7.52.1-0
[Jun 19 11:29:56] SOUT libgcc:     5.2.0-0
[Jun 19 11:29:56] SOUT openssl:    1.0.2l-0
[Jun 19 11:29:56] SOUT pip:        9.0.1-py27_1
[Jun 19 11:29:56] SOUT python:     2.7.13-0
[Jun 19 11:29:56] SOUT readline:   6.2-2
[Jun 19 11:29:56] SOUT samtools:   1.3.1-5       bioconda
[Jun 19 11:29:56] SOUT setuptools: 27.2.0-py27_0
[Jun 19 11:29:56] SOUT sqlite:     3.13.0-0
[Jun 19 11:29:56] SOUT tk:         8.5.18-0
[Jun 19 11:29:56] SOUT wheel:      0.29.0-py27_0
[Jun 19 11:29:56] SOUT zlib:       1.2.8-3
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:57] DEBU Container [72b9ca0e56f8 step-e95bf001c8] completed with exit code␣
↪[0] as expected
[Jun 19 11:29:57] DEBU Container [72b9ca0e56f8 step-e95bf001c8] removed
[Jun 19 11:29:57] STEP Wrap [build/dist] as [quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0]
[Jun 19 11:29:57] DEBU Creating container [step-6f1c176372]
[Jun 19 11:29:58] DEBU Packing succeeded
```

As the output indicates, this command built the container named `quay.io/biocontainers/mulled-v2-fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0`. This is the same namespace / URL that would be used if or when published by the BioContainers project.

Note: The first part of this `mulled-v2` hash is a hash of the package names that went into it, the second the packages used and build number. Check out the Multi-package Containers web application to explore best practice channels and build such hashes.

We can see this new container when running the Docker command `images` and explore the new container interactively with `docker run`.

```
$ docker images
REPOSITORY                                                                  TAG          ␣
↪                                IMAGE ID           CREATED              SIZE
quay.io/biocontainers/mulled-v2-fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40  ␣
↪03dc1d2818d9de56938078b8b78b82d967c1f820-0   a740fe1e6a9e        16 hours ago          ␣
↪104 MB
quay.io/biocontainers/seqtk                                                 1.2--0       ␣
↪                                10bc359ebd30       2 days ago           7.34 MB
```

(continues on next page)

(continued from previous page)

```
continuumio/miniconda                                                           latest
↪                                 6965a4889098        3 weeks ago          437 MB
bgruening/busybox-bash                                                         0.1
↪                                 3d974f51245c        9 months ago         6.73 MB
$ docker run -i -t quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0 /
↪bin/bash
bash-4.2# which samtools
/usr/local/bin/samtools
bash-4.2# which bwa
/usr/local/bin/bwa
```

As before, we can test running the tool inside its container in Galaxy using the `--biocontainers` flag.

```
$ planemo test --biocontainers bwa_and_samtools.xml
...
2017-03-01 10:20:58,077 INFO  [galaxy.tools.actions] Handled output named output_2 for
↪tool bwa_and_samtools (17.443 ms)
2017-03-01 10:20:58,090 INFO  [galaxy.tools.actions] Added output datasets to history
↪(12.935 ms)
2017-03-01 10:20:58,095 INFO  [galaxy.tools.actions] Verified access to datasets for
↪Job[unflushed,tool_id=bwa_and_samtools] (0.021 ms)
2017-03-01 10:20:58,096 INFO  [galaxy.tools.actions] Setup for job Job[unflushed,tool_
↪id=bwa_and_samtools] complete, ready to flush (5.755 ms)
2017-03-01 10:20:58,116 INFO  [galaxy.tools.actions] Flushed transaction for job
↪Job[id=1,tool_id=bwa_and_samtools] (19.582 ms)
2017-03-01 10:20:58,869 INFO  [galaxy.jobs.handler] (1) Job dispatched
2017-03-01 10:20:59,067 DEBUG [galaxy.tools.deps] Using dependency bwa version 0.7.15 of
↪type conda
2017-03-01 10:20:59,067 DEBUG [galaxy.tools.deps] Using dependency samtools version 1.3.
↪1 of type conda
2017-03-01 10:20:59,067 DEBUG [galaxy.tools.deps] Using dependency bwa version 0.7.15 of
↪type conda
2017-03-01 10:20:59,068 DEBUG [galaxy.tools.deps] Using dependency samtools version 1.3.
↪1 of type conda
2017-03-01 10:20:59,083 INFO  [galaxy.tools.deps.containers] Checking with container
↪resolver [ExplicitContainerResolver[]] found description [None]
2017-03-01 10:20:59,142 INFO  [galaxy.tools.deps.containers] Checking with container
↪resolver [CachedMulledDockerContainerResolver[namespace=biocontainers]] found
↪description [ContainerDescription[identifier=quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0,
↪type=docker]]
2017-03-01 10:20:59,163 INFO  [galaxy.jobs.command_factory] Built script [/tmp/tmpQs0gyp/
↪job_working_directory/000/1/tool_script.sh] for tool command [bwa > /tmp/tmpQs0gyp/
↪files/000/dataset_1.dat 2>&1 ; samtools > /tmp/tmpQs0gyp/files/000/dataset_2.dat 2>&1]
2017-03-01 10:20:59,367 DEBUG [galaxy.tools.deps] Using dependency samtools version None
↪of type conda
2017-03-01 10:20:59,367 DEBUG [galaxy.tools.deps] Using dependency samtools version None
↪of type conda
ok

----------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
XML: /private/tmp/tmpQs0gyp/xunit.xml
----------------------------------------------------------------------
Ran 1 test in 7.553s

OK
2017-03-01 10:21:05,223 INFO  [test_driver] Shutting down
2017-03-01 10:21:05,224 INFO  [test_driver] Shutting down embedded galaxy web server
2017-03-01 10:21:05,226 INFO  [test_driver] Embedded web server galaxy stopped
2017-03-01 10:21:05,226 INFO  [test_driver] Stopping application galaxy
...
2017-03-01 10:21:05,228 INFO  [galaxy.jobs.handler] job handler stop queue stopped
Testing complete. HTML report is in "/home/planemo/workspace/planemo/tool_test_output.
↪html".
All 1 test(s) executed passed.
bwa_and_samtools[0]: passed
```

In particular take note of the line:

```
2017-03-01 10:20:59,142 INFO  [galaxy.tools.deps.containers] Checking with container␣
↪resolver [CachedMulledDockerContainerResolver[namespace=biocontainers]] found␣
↪description [ContainerDescription[identifier=quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0,
↪type=docker]]
```

Here we can see the container ID (`quay.io/biocontainers/mulled-v2-fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1` from earlier has been cached on our Docker host is picked up by Galaxy. This is used to run the simple tool tests and indeed they pass.

In our initial seqtk example, the container resolver that matched was of type `MulledDockerContainerResolver` indicating that the Docker image would be downloaded from the BioContainer repository and this time the resolve that matched was of type `CachedMulledDockerContainerResolver` meaning that Galaxy would just use the locally cached version from the Docker host (i.e. the one we built with `planemo mull` above).

Note: Planemo doesn't yet expose options that make it possible to build mulled containers for local packages that have yet to be published to anaconda.org but the mulled toolkit allows this. See mulled documentation for more information. However, once a container for a local package is built with `mulled-build-tool` the `--biocontainers` command should work to test it.

Publishing BioContainers

Building unpublished BioContainers on the fly is great for testing but for production use and to increase reproducibility such containers should ideally be published as well.

BioContainers maintains a registry of package combinations to be published using these long mulled hashes. This registry is represented as a Github repository named multi-package-containers. The Planemo command `container_register` will inspect a tool and open a Github pull request to add the tool's combination of packages to the registry. Once merged, this pull request will result in the corresponding BioContainers image to be published (with the correct mulled has as its name) - these can be subsequently be picked up by Galaxy.

Various Github related settings need to be configured in order for Planemo to be able to open pull requests on your behalf as part of the `container_register` command. To simplify all of this - the Planemo community maintains a list of Github repositories containing Galaxy and/or CWL tools that are scanned daily by Travis. For each such repository, the Travis job will run `container_register` across the repository on all tools resulting in new registry pull requests for all new combinations of tools. This list is maintained in a script named `monitor.sh` in the planemo-monitor repository. The easiest way to ensure new containers are built for your tools is simply to open open a pull request to add your tool repositories to this list.

Explicit Annotation

This section of documentation needs to be filled out but a detailed example is worked through this documentation from Aaron Petkau (@apetkau) built at the 2014 Galaxy Community Conference Hackathon.

5.4 How do I…

This section contains a number of smaller topics with links and examples meant to provide relatively concrete answers for specific tool development scenarios.

5.4.1 … deal with index/reference data?

Galaxy's concept of data tables are meant to provide tools with access reference datasets or index data not tied to particular histories or users. A common example would be FASTA files for various genomes or mapper-specific indices of those files (e.g. a BWA index for the hg19 genome).

Galaxy data managers are specialized tools designed to populate tool data tables.

5.4.2 … cite tools without an obvious DOI?

In the absence of an obvious DOI, tools may contain embedded BibTeX directly.

Futher reading:

- bibtex.xml (test tool with a bunch of random examples)
- bwa-mem.xml (BWA-MEM tool by Anton Nekrutenko demonstrating citation of an arXiv article)
- macros.xml (Macros for vcflib tool demonstrating citing a github repository)

5.4.3 … declare a Docker container for my tool?

Galaxy tools can be decorated to with `container` tags indicated Docker container ids that the tools can run inside of.

The longer term plan for the Tool Shed ecosystem is to be able to automatically build Docker containers for tool dependency descriptions and thereby obtain this Docker functionality for free and in a way that is completely backward compatible with non-Docker deployments.

Further reading:

- Complete tutorial on Github by Aaron Petkau. Covers installing Docker, building a Dockerfile, publishing to Docker Hub, annotating tools and configuring Galaxy.
- Another tutorial from the Galaxy User Group Grand Ouest.
- Landing page on the Galaxy Wiki
- Impementation details on Pull Request #401

5.4.4 ... do extra validation of parameters?

Tool parameters support a `validator` element (syntax) to perform validation of a single parameter. More complex validation across parameters can be performed using arbitrary Python functions using the `code` file syntax but this feature should be used sparingly.

Further reading:

- validator XML tag syntax on the Galaxy wiki.
- fastq_filter.xml (a FASTQ filtering tool demonstrating validator constructs)
- gffread.xml (a tool by Jim Johnson demonstrating using regular expressions with `validator` tags)
- code_file.xml, code_file.py (test files demonstrating defining a simple constraint in Python across two parameters)

5.4.5 ... check input type in command blocks?

Input data parameters may specify multiple formats. For example

```
<param name="input" type="data" format="fastq,fasta" label="Input" />
```

If the command-line under construction doesn't require changes based on the input type - this may just be referenced as `$input`. However, if the command-line under construction uses different argument names depending on type for instance - it becomes important to dispatch on the underlying type.

In this example `$input.ext` - would return the short code for the actual datatype of the input supplied - for instance the string `fasta` or `fastqsanger` would be valid responses for inputs to this parameter for the above definition.

While `.ext` may sometimes be useful - there are many cases where it is inappropriate because of subtypes - checking if `.ext` is equal to `fastq` in the above example would not catch `fastqsanger` inputs for instance. To check if an input matches a type or any subtype thereof - the `is_of_type` method can be used. For instance

```
$input.is_of_type('fastq')
```

would check if the input is of type `fastq` or any derivative types such as `fastqsanger`.

- Pull Request 457

5.4.6 ... handle arbitrary output data formats?

If the output format of a tool's output cannot be known ahead of time, Galaxy can be instructed to "sniff" the output and determine the data type using the same method used for uploads. Adding the `auto_format="true"` attribute to a tool's output enables this.

```
<output name="out1" auto_format="true" label="Auto Output" />
```

- output_auto_format.xml

5.4.7 ... determine the user submitting a job?

The variable `$__user_email__` (as well as `$__user_name__` and `$__user_id__`) is available when building up your command in the tool's `<command>` block. The following tool demonstrates the use of this and a few other special parameters available to all tools.

- special_params.xml

5.4.8 ... test with multiple value inputs?

To write tests that supply multiple values to a `multiple="true"` `select` or `data` parameter - simply specify the multiple values as a comma seperated list.

Here are examples of each:

- multi_data_param.xml
- muti_select.xml

5.4.9 ... test dataset collections?

Here are some examples of testing tools that consume collections with `type="data_collection"` parameters.

- collection_paired_test.xml
- collection_mixed_param.xml
- collection_nested_param.xml

Here are some examples of testing tools that produce collections with `output_collection` elements.

- collection_creates_list.xml
- collection_creates_list_2.xml
- collection_creates_pair.xml
- collection_creates_pair_from_type.xml

5.4.10 ... test discovered datasets?

Tools which dynamically discover datasets after the job is complete, either using the `<discovered_datasets>` element, the older default pattern approach (e.g. finding files with names like `primary_DATASET_ID_sample1_true_bam_hg18`), or the undocumented `galaxy.json` approach can be tested by placing `discovered_dataset` elements beneath the corresponding `output` element with the `designation` corresponding to the file to test.

```
<test>
  <param name="input" value="7" />
  <output name="report" file="example_output.html">
    <discovered_dataset designation="world1" file="world1.txt" />
    <discovered_dataset designation="world2">
      <assert_contents>
        <has_line line="World Contents" />
      </assert_contents>
    </discovered_dataset>
```

(continues on next page)

(continued from previous page)

```
  </output>
</test>
```

The test examples distributed with Galaxy demonstrating dynamic discovery and the testing thereof include:

- multi_output.xml
- multi_output_assign_primary.xml
- multi_output_configured.xml

5.4.11 ... test composite dataset contents?

Tools which consume Galaxy composite datatypes can generate test inputs using the `composite_data` element demonstrated by the following tool.

- composite.xml

Tools which produce Galaxy composite datatypes can specify tests for the individual output files using the `extra_files` element demonstrated by the following tool.

- composite_output.xml
- macs2_callpeak.xml

5.4.12 ... test index (.loc) data?

There is an idiom to supply test data for index during tests using Planemo.

To create this kind of test, one needs to provide a `tool_data_table_conf.xml.test` beside your tool's `tool_data_table_conf.xml.sample` file that specifies paths to test `.loc` files which in turn define paths to the test index data. Both the `.loc` files and the `tool_data_table_conf.xml.test` can use the value `${__HERE__}` which will be replaced with the path to the directory the file lives in. This allows using relative-like paths in these files which is needed for portable tests.

An example commit demonstrating the application of this approach to a Picard tool can be found here.

These tests can then be run with the Planemo test command.

5.4.13 ... test exit codes?

A `test` element can check the exit code of the underlying job using the `check_exit_code="n"` attribute.

- job_properties.xml

5.4.14 ... test failure states?

Normally, all tool test cases described by a `test` element are expected to pass - but on can assert a job should fail by adding `expect_failure="true"` to the `test` element.

- job_properties.xml

5.4.15 ... test output filters work?

If your tool contains `filter` elements, you can't verify properties of outputs that are filtered out and do not exist. The `test` element may contain an `expect_num_outputs` attribute to specify the expected number of outputs, this can be used to verify that outputs not listed are expected to be filtered out during tool execution.

- output_filter.xml

5.4.16 ... test metadata?

Output metadata can be checked using `metadata` elements in the XML description of the `output`.

- metadata.xml

5.4.17 ... test tools installed in an existing Galaxy instance?

Do not use planemo, Galaxy should be used to test its tools directly. The following two commands can be used to test Galaxy tools in an existing instance.

```
$ sh run_tests.sh --report_file tool_tests_shed.html --installed
```

This above command specifies the `--installed` flag when calling `run_tests.sh`, this tells the test framework to test Tool Shed installed tools and only those tools.

```
$ GALAXY_TEST_TOOL_CONF=config/tool_conf.xml sh run_tests.sh --report_file tool_tests_
↪tool_conf.html functional.test_toolbox
```

The second command sets `GALAXY_TEST_TOOL_CONF` environment variable, which will restrict the testing framework to considering a single tool conf file (such as the default tools that ship with Galaxy `config/tool_conf.xml.sample` and which must have their dependencies setup manually). The last argument to `run_tests.sh`, `functional.test_toolbox` tells the test framework to run all the tool tests in the configured tool conf file.

Note: *Tip:* To speed up tests you can use a pre-migrated database file the way Planemo does by setting the following environment variable before running `run_tests.sh`.

```
$ export GALAXY_TEST_DB_TEMPLATE="https://github.com/jmchilton/galaxy-downloads/raw/
↪master/db_gx_rev_0127.sqlite"
```

5.4.18 ... test tools against a package or container in a bioconda pull request?

First, obtain the artifacts of the PR by adding this comment: `@BiocondaBot please fetch artifacts`. In the reply one finds the links a zip file containing the built package and docker image. Download this zip and extract it. For the following let `PACKAGES_DIR` be the absolute path to the directory `packages` in the resulting unzipped directory and `IMAGE_ZIP` be the absolute path to the `tar.gz` file in the `images` directory in the unzipped directory.

In order to test the tool with the package add the following to the planemo call:

```
$ planemo test ... --conda_channels file://PACKAGES_DIR,conda-forge,bioconda,defaults ...
```

For containerized testing we need to differentiate two cases:

1. the tool has a single requirement (that is fulfilled by the container)

2. the tool has multiple requirements (in this case a docker image will be built on the fly using package)

For the former case the docker image that has been created by the bioconda CI needs to be loaded:

```
$ gzip -dc IMAGE_ZIP | docker load
```

and a planemo test can then simply use this image:

```
$ planemo test ... --biocontainers --no_dependency_resolution --no_conda_auto_init ...
```

For the later case it suffices to call planemo as follows:

```
$ planemo test ... --biocontainers --no_dependency_resolution --no_conda_auto_init --
↪conda_channels file://PACKAGES_DIR,conda-forge,bioconda,defaults ...
```

5.4.19 ... interactively debug tool tests?

It can be desirable to interactively debug a tool test. In order to do so, start `planemo test` with the option `--no_cleanup`. Inspect the output: After Galaxy starts up, the tests commence. At the start of each test one finds a message: `( <TOOL_ID> ) > Test-N`. After some upload jobs, the actual tool job is started (it is the last before the next test is executed). There you will find a message like `Built script [/tmp/tmp1zixgse3/job_working_directory/000/3/tool_script.sh]`

In this case `/tmp/tmp1zixgse3/job_working_directory/000/3/` is the job dir. It contains some files and directories of interest:

- `tool_script.sh`: the bash script generated from the tool's `command` and `version_command` tags plus some boiler plate code
- `galaxy_3.sh` (note that the number may be different): a shell script setting up the environment (e.g. paths and environment variables), starting the `tool_script.sh`, and postprocessing (e.g. error handling and setting metadata)
- `working`: the job working directory
- `outputs`: a directory containing the job stderr and stdout

For a tool test that uses a conda environment to resolve the requirements one can simply change into `working` and execute `../tool_script.sh` (works as long as no special environment variables are used; in this case `../galaxy_3.sh` needs to be executed after cleaning the job dir). By editing the tool script one may understand/fix problems in the `command` block faster than by rerunning `planemo test` over and over again.

Alternatively one can change into the `working` dir and load the conda environment (the code to do so can be found in `tool_script.sh`: `. PATH_TO_CONDA_ENV activate`). Afterwards one can execute individual commands, e.g. those found in `tool_script.sh` or variants.

For a tool test that uses Docker to to resolve the requirements one needs to execute `../galaxy_3.sh`, because it executes `docker run ... tool_script.sh` in order to rerun the job (with a possible edited version of the tool script). In order to run the docker container interactively execute the `docker run .... /bin/bash` that you find in `../galaxy_3.sh` (i.e. ommitting the call of the `tool_script.sh`) with added parameter `-it`. Note that the `docker run` command contains some shell variables (`-v "$_GALAXY_JOB_TMP_DIR:$_GALAXY_JOB_TMP_DIR:rw" -v "$_GALAXY_JOB_HOME_DIR:$_GALAXY_JOB_HOME_DIR:rw"`) which ensure that the job's temporary and home directory are available within docker. Ideally these shell variables are set to the same values as in `../galaxy_3.sh`, but often its sufficient to remove this part from the `docker run` call.

CHAPTER

SIX

BUILDING COMMON WORKFLOW LANGUAGE TOOLS

The following links are for the same tutorial describing the basics of how to build Common Workflow Language tools. The first variant is tailored to local development environments (e.g. if Planemo has been installed with `brew` or `pip`) and the second is for developers using a dedicated Planemo virtual appliance (available for Docker, Vagrant, etc…).

6.1 Building Common Workflow Language Tools Using Planemo

This tutorial is a gentle introduction to writing Common Workflow Language tools using Planemo. Please read the installation instructions for Planemo if you have not already installed it.

6.1.1 The Basics

This guide is going to demonstrate building up tools for commands from Heng Li's Seqtk package - a package for processing sequence data in FASTA and FASTQ files.

To get started let's install Seqtk. Here we are going to use `conda` to install Seqtk - but however you obtain it should be fine.

```
$ conda install --force --yes -c conda-forge -c bioconda seqtk=1.2
    ... seqtk installation ...
$ seqtk seq
        Usage:   seqtk seq [options] <in.fq>|<in.fa>
        Options: -q INT    mask bases with quality lower than INT [0]
                 -X INT    mask bases with quality higher than INT [255]
                 -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
                 -l INT    number of residues per line; 0 for 2^32-1 [0]
                 -Q INT    quality shift: ASCII-INT gives base quality [33]
                 -s INT    random seed (effective with -f) [11]
                 -f FLOAT  sample FLOAT fraction of sequences [1]
                 -M FILE   mask regions in BED or name list FILE [null]
                 -L INT    drop sequences with length shorter than INT [0]
                 -c        mask complement region (effective with -M)
                 -r        reverse complement
                 -A        force FASTA output (discard quality)
                 -C        drop comments at the header lines
                 -N        drop sequences containing ambiguous bases
                 -1        output the 2n-1 reads only
                 -2        output the 2n reads only
                 -V        shift quality by '(-Q) - 33'
```

Next we will download an example FASTQ file and test out the a simple Seqtk command - `seq` which converts FASTQ files into FASTA.

```
$ wget https://raw.githubusercontent.com/galaxyproject/galaxy-test-data/master/2.fastq
$ seqtk seq -A 2.fastq > 2.fasta
$ cat 2.fasta
>EAS54_6_R1_2_1_413_324
CCCTTCTTGTCTTCAGCGTTTCTCC
>EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
>EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
```

Common Workflow Language tool files are just simple YAML files, so at this point one could just open a text editor and start implementing the tool. Planemo has a command `tool_init` to quickly generate a skeleton to work from, so let's start by doing that.

```
$ planemo tool_init --cwl --id 'seqtk_seq' --name 'Convert to FASTA (seqtk)'
```

The `tool_init` command can take various complex arguments - but three two most basic ones are shown above `--cwl`, `--id` and `--name`. The `--cwl` flag tells Planemo to generate a Common Workflow Language tool. `--id` is a short identifier for this tool and it should be unique across all tools. `--name` is a short, human-readable name for the the tool - it corresponds to the `label` attribute in the CWL tool document.

The above command will generate the file `seqtk_seq.cwl` - which should look like this.

```
#!/usr/bin/env cwl-runner
cwlVersion: 'v1.0'
class: CommandLineTool
id: "seqtk_seq"
label: "Convert to FASTA (seqtk)"
inputs: [] # TODO
outputs: [] # TODO
baseCommand: []
arguments: []
doc: |
   TODO: Fill in description.
```

This tool file has the common fields required for a CWL tool with TODO notes, but you will still need to open up the editor and fill out the command, describe input parameters, tool outputs, writeup usage documentation (`doc`), etc..

The `tool_init` command can do a little bit better than this as well. We can use the test command we tried above `seqtk seq -A 2.fastq > 2.fasta` as an example to generate a command block by specifing the inputs and the outputs as follows.

```
$ planemo tool_init --force \
                    --cwl \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta
```

This will generate the following CWL tool definition - which now has correct definitions for the input, output, and command specified. These represent a best guess by planemo, and in most cases will need to be tweaked manually after the tool is generated.

```
#!/usr/bin/env cwl-runner
cwlVersion: 'v1.0'
class: CommandLineTool
id: "seqtk_seq"
label: "Convert to FASTA (seqtk)"
inputs:
  input1:
    type: File
    doc: |
      TODO
    inputBinding:
      position: 1
      prefix: "-a"
outputs:
  output1:
    type: File
    outputBinding:
      glob: out
baseCommand:
  - "seqtk"
  - "seq"
arguments: []
stdout: out
doc: |
   TODO: Fill in description.
```

As shown at the beginning of this section, the command `seqtk seq` generates a help message for the `seq` command. `tool_init` can take that help message and stick it right in the generated tool file using the `help_from_command` option.

Generally command help messages aren't exactly appropriate for tools since they mention argument names and simillar details that are abstracted away by the tool - but they can be an excellent place to start.

The following Planemo's `tool_init` call has been enhanced to use `--help_from_command`.

```
$ planemo tool_init --force \
                    --cwl \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --requirement seqtk@1.2 \
                    --container 'quay.io/biocontainers/seqtk:1.2--0' \
                    --test_case \
                    --help_from_command 'seqtk seq'
```

This command generates the following CWL YAML file.

```
#!/usr/bin/env cwl-runner
cwlVersion: 'v1.0'
class: CommandLineTool
id: "seqtk_seq"
label: "Convert to FASTA (seqtk)"
```

(continues on next page)

(continued from previous page)

```
hints:
  DockerRequirement:
    dockerPull: quay.io/biocontainers/seqtk:1.2--1
  SoftwareRequirement:
    packages:
    - package: seqtk
      version:
      - "1.2"
inputs:
  input1:
    type: File
    doc: |
      TODO
    inputBinding:
      position: 1
      prefix: "-a"
outputs:
  output1:
    type: File
    outputBinding:
      glob: out
baseCommand:
  - "seqtk"
  - "seq"
arguments: []
stdout: out
doc: |

  Usage:   seqtk seq [options] <in.fq>|<in.fa>

  Options: -q INT    mask bases with quality lower than INT [0]
           -X INT    mask bases with quality higher than INT [255]
           -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
           -l INT    number of residues per line; 0 for 2^32-1 [0]
           -Q INT    quality shift: ASCII-INT gives base quality [33]
           -s INT    random seed (effective with -f) [11]
           -f FLOAT  sample FLOAT fraction of sequences [1]
           -M FILE   mask regions in BED or name list FILE [null]
           -L INT    drop sequences with length shorter than INT [0]
           -c        mask complement region (effective with -M)
           -r        reverse complement
           -A        force FASTA output (discard quality)
           -C        drop comments at the header lines
           -N        drop sequences containing ambiguous bases
           -1        output the 2n-1 reads only
           -2        output the 2n reads only
           -V        shift quality by '(-Q) - 33'
           -U        convert all bases to uppercases
           -S        strip of white spaces in sequences
```

In addition to generating a CWL tool adding the `--test_case` flag generates from more files that are useful including `seqtk_seq_job.yml` as shown below:

```
input1:
  class: File
  path: test-data/2.fastq
```

This is a CWL job input document and should allow you to run the example command using any CWL implementation. For instance if you have cwltool (`cwltool`) or Toil (`cwltoil`) on your `PATH` the following examples should work.

```
$ cwltool seqtk_seq.cwl seqtk_seq_job.yml
/Users/john/workspace/planemo/.venv/bin/cwltool 1.0.20180508202931
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl'
[job seqtk_seq.cwl] /private/tmp/docker_tmpXgtSLt$ docker \
    run \
    -i \
    --volume=/private/tmp/docker_tmpXgtSLt:/private/var/spool/cwl:rw \
    --volume=/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpGG1thW:/tmp:rw \
    --volume=/Users/john/tool_init_exercise/test-data/2.fastq:/private/var/lib/cwl/
↪stg7db12d3a-2375-42ed-ba60-8a0ef69ffe80/2.fastq:ro \
    --workdir=/private/var/spool/cwl \
    --read-only=true \
    --log-driver=none \
    --user=502:20 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/private/var/spool/cwl \
    quay.io/biocontainers/seqtk:1.2--1 \
    seqtk \
    seq \
    -A \
    /private/var/lib/cwl/stg7db12d3a-2375-42ed-ba60-8a0ef69ffe80/2.fastq > /private/tmp/
↪docker_tmpXgtSLt/out
[job seqtk_seq.cwl] completed success
{
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "location": "file:///Users/john/tool_init_exercise/out",
        "path": "/Users/john/tool_init_exercise/out",
        "class": "File",
        "size": 150
    }
}
```

```
$ cwltoil seqtk_seq.cwl seqtk_seq_job.yml
jlaptop17.local 2018-05-21 15:25:30,630 MainThread INFO toil.lib.bioio: Root logger is␣
↪at level 'INFO', 'toil' logger at level 'INFO'.
jlaptop17.local 2018-05-21 15:25:30,648 MainThread INFO toil.jobStores.abstractJobStore:␣
↪The workflow ID is: '55a08d91-1852-4069-97a9-741abd2ea04e'
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl'
jlaptop17.local 2018-05-21 15:25:30,650 MainThread INFO cwltool: Resolved 'seqtk_seq.cwl
↪' to 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl'
jlaptop17.local 2018-05-21 15:25:31,793 MainThread INFO toil.common: Using the single␣
↪machine batch system
jlaptop17.local 2018-05-21 15:25:31,793 MainThread WARNING toil.batchSystems.
```

(continues on next page)

(continued from previous page)

```
↪singleMachine: Limiting maxCores to CPU count of system (8).
jlaptop17.local 2018-05-21 15:25:31,793 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxMemory to physically available memory (17179869184).
jlaptop17.local 2018-05-21 15:25:31,800 MainThread INFO toil.common: Created the␣
↪workflow directory at /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/toil-55a08d91-
↪1852-4069-97a9-741abd2ea04e-132281828025877
jlaptop17.local 2018-05-21 15:25:31,800 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxDisk to physically available disk (206962089984).
jlaptop17.local 2018-05-21 15:25:31,808 MainThread INFO toil.common: User script␣
↪ModuleDescriptor(dirPath='/Users/john/workspace/planemo/.venv/lib/python2.7/site-
↪packages', name='toil.cwl.cwltoil', fromVirtualEnv=True) belongs to Toil. No need to␣
↪auto-deploy it.
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: No user script to␣
↪auto-deploy.
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: Written the␣
↪environment for the jobs to the environment file
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: Caching all jobs in␣
↪job store
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: 0 jobs downloaded.
jlaptop17.local 2018-05-21 15:25:31,825 MainThread INFO toil: Running Toil version 3.15.
↪0-0e3a87e738f5e0e7cff64bfdad337d592bd92704.
jlaptop17.local 2018-05-21 15:25:31,825 MainThread INFO toil.realtimeLogger: Real-time␣
↪logging disabled
jlaptop17.local 2018-05-21 15:25:31,832 MainThread INFO toil.toilState: (Re)building␣
↪internal scheduler state
2018-05-21 15:25:31,832 - toil.toilState - INFO - (Re)building internal scheduler state
jlaptop17.local 2018-05-21 15:25:31,832 MainThread INFO toil.leader: Found 1 jobs to␣
↪start and 0 jobs with successors to run
2018-05-21 15:25:31,832 - toil.leader - INFO - Found 1 jobs to start and 0 jobs with␣
↪successors to run
jlaptop17.local 2018-05-21 15:25:31,832 MainThread INFO toil.leader: Checked batch␣
↪system has no running jobs and no updated jobs
2018-05-21 15:25:31,832 - toil.leader - INFO - Checked batch system has no running jobs␣
↪and no updated jobs
jlaptop17.local 2018-05-21 15:25:31,833 MainThread INFO toil.leader: Starting the main␣
↪loop
2018-05-21 15:25:31,833 - toil.leader - INFO - Starting the main loop
jlaptop17.local 2018-05-21 15:25:31,834 MainThread INFO toil.leader: Issued job 'file:///
↪Users/john/tool_init_exercise/seqtk_seq.cwl' seqtk seq u/c/jobzYKQ3V with job batch␣
↪system ID: 0 and cores: 1, disk: 3.0 G, and memory: 2.0 G
2018-05-21 15:25:31,834 - toil.leader - INFO - Issued job 'file:///Users/john/tool_init_
↪exercise/seqtk_seq.cwl' seqtk seq u/c/jobzYKQ3V with job batch system ID: 0 and cores:␣
↪1, disk: 3.0 G, and memory: 2.0 G
jlaptop17.local 2018-05-21 15:25:33,953 MainThread INFO toil.leader: Job ended␣
↪successfully: 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl' seqtk seq u/c/
↪jobzYKQ3V
2018-05-21 15:25:33,953 - toil.leader - INFO - Job ended successfully: 'file:///Users/
↪john/tool_init_exercise/seqtk_seq.cwl' seqtk seq u/c/jobzYKQ3V
jlaptop17.local 2018-05-21 15:25:33,955 MainThread INFO toil.leader: Finished the main␣
↪loop: no jobs left to run
2018-05-21 15:25:33,955 - toil.leader - INFO - Finished the main loop: no jobs left to␣
↪run
```

(continues on next page)

(continued from previous page)

```
jlaptop17.local 2018-05-21 15:25:33,955 MainThread INFO toil.serviceManager: Waiting for␣
↪service manager thread to finish ...
2018-05-21 15:25:33,955 - toil.serviceManager - INFO - Waiting for service manager␣
↪thread to finish ...
jlaptop17.local 2018-05-21 15:25:34,841 MainThread INFO toil.serviceManager: ...␣
↪finished shutting down the service manager. Took 0.885795116425 seconds
2018-05-21 15:25:34,841 - toil.serviceManager - INFO - ... finished shutting down the␣
↪service manager. Took 0.885795116425 seconds
jlaptop17.local 2018-05-21 15:25:34,842 MainThread INFO toil.statsAndLogging: Waiting␣
↪for stats and logging collator thread to finish ...
2018-05-21 15:25:34,842 - toil.statsAndLogging - INFO - Waiting for stats and logging␣
↪collator thread to finish ...
jlaptop17.local 2018-05-21 15:25:34,854 MainThread INFO toil.statsAndLogging: ...␣
↪finished collating stats and logs. Took 0.0120511054993 seconds
2018-05-21 15:25:34,854 - toil.statsAndLogging - INFO - ... finished collating stats and␣
↪logs. Took 0.0120511054993 seconds
jlaptop17.local 2018-05-21 15:25:34,855 MainThread INFO toil.leader: Finished toil run␣
↪successfully
2018-05-21 15:25:34,855 - toil.leader - INFO - Finished toil run successfully
{
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "nameext": "",
        "nameroot": "out",
        "http://commonwl.org/cwltool#generation": 0,
        "location": "file:///Users/john/tool_init_exercise/out",
        "class": "File",
        "size": 150
    }
jlaptop17.local 2018-05-21 15:25:34,866 MainThread INFO toil.common: Successfully␣
↪deleted the job store: <toil.jobStores.fileJobStore.FileJobStore object at 0x1057205d0>
}2018-05-21 15:25:34,866 - toil.common - INFO - Successfully deleted the job store:
↪<toil.jobStores.fileJobStore.FileJobStore object at 0x1057205d0>
```

At this point we have a fairly a functional CWL tool with test and usage documentation. This was a pretty simple example - usually you will need to put more work into the tool to get to this point - `tool_init` is really just designed to get you started.

Now lets lint and test the tool we have developed. The Planemo's `lint` (or just `l`) command will review tool for validity, obvious mistakes, and Planemo "best practices".

```
$ planemo l seqtk_seq.cwl
Linting tool /Users/john/workspace/planemo/docs/writing/seqtk_seq.cwl
Applying linter general... CHECK
.. CHECK: Tool defines a version [0.0.1].
.. CHECK: Tool defines a name [Convert to FASTA (seqtk)].
.. CHECK: Tool defines an id [seqtk_seq_v3].
.. CHECK: Tool specifies profile version [16.04].
Applying linter cwl_validation... CHECK
.. INFO: CWL appears to be valid.
Applying linter docker_image... CHECK
.. INFO: Tool will run in Docker image [quay.io/biocontainers/seqtk:1.2--1].
```

(continues on next page)

(continued from previous page)

```
Applying linter new_draft... CHECK
.. INFO: Modern CWL version [v1.0]
```

In addition to the actual tool and job files, `--test_case` caused a test file to be generated using the example command and provided test data. The file contents are as follows:

```
- doc: test generated from example command
  job: seqtk_seq_job.yml
  outputs:
    output1:
      path: test-data/2.fasta
```

Unlike the job file, this file is a Planemo-specific artifact. This file may contain 1 or more tests - each test is an element of the top-level list. `tool_init` will use the example command to build just one test.

Each test consists of a few parts:

- `doc` - this attribute provides a short description for the test.
- `job` - this can be the path to a CWL job description or a job description embedded right in the test (`tool_init` builds the latter).
- `outputs` - this section describes the expected output for a test. Each output ID of the tool or workflow under test can appear as a key. The example above just describes expected specific output file contents exactly but many more expectations can be described.

For more information on the test file format check out the Test Format docs.

The tests described in this file can be run using the `planemo test` command on the original file.

```
$ planemo test --no-container seqtk_seq.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20180508202931
cwltool INFO: Resolved '/Users/john/tool_init_exercise/seqtk_seq.cwl' to 'file:///Users/
↪john/tool_init_exercise/seqtk_seq.cwl'
cwltool INFO: [job seqtk_seq.cwl] /private/tmp/docker_tmpvLE9SS$ seqtk \
    seq \
    -A \
    /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpGM22d_/stg0c0cad75-7ca0-
↪4f3a-9d77-63e9c49f5353/2.fastq > /private/tmp/docker_tmpvLE9SS/out
cwltool INFO: [job seqtk_seq.cwl] completed success
cwltool INFO: Final process status is success
All 1 test(s) executed passed.
seqtk_seq_0: passed
```

This is a bit a different than running the job. For one thing, we don't need to specify an input job - instead Planemo will automatically find the test file and run all the jobs described inside that file. Additionally, Planemo will check the outputs to ensure the match the test expectations.

In addition to the in console display of test results as red (failing) or green (passing), Planemo also creates an HTML report for the test results by default. Many more test report options are available such `--test_output_xunit` which is useful in certain continuous integration environments. See `planemo test --help` for more options, as well as the `test_reports` command.

The above test example used cwltool to run our test and disabled containerization. By dropping the `--no-container` argument we can run the tool in a Docker container. By passing an engine argument as `--engine toil` we can run our test in Toil, an alternative CWL implementation.

```
$ planemo test seqtk_seq.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20180508202931
cwltool INFO: Resolved '/Users/john/tool_init_exercise/seqtk_seq.cwl' to 'file:///Users/
↪john/tool_init_exercise/seqtk_seq.cwl'
cwltool INFO: [job seqtk_seq.cwl] /private/tmp/docker_tmpUeIpXJ$ docker \
    run \
    -i \
    --volume=/private/tmp/docker_tmpUeIpXJ:/private/var/spool/cwl:rw \
    --volume=/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpteo_2Z:/tmp:rw \
    --volume=/Users/john/tool_init_exercise/test-data/2.fastq:/private/var/lib/cwl/
↪stg939ee60b-a194-4177-8410-c40a1acb38ea/2.fastq:ro \
    --workdir=/private/var/spool/cwl \
    --read-only=true \
    --log-driver=none \
    --user=502:20 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/private/var/spool/cwl \
    quay.io/biocontainers/seqtk:1.2--1 \
    seqtk \
    seq \
    -A \
    /private/var/lib/cwl/stg939ee60b-a194-4177-8410-c40a1acb38ea/2.fastq > /private/tmp/
↪docker_tmpUeIpXJ/out
cwltool INFO: [job seqtk_seq.cwl] completed success
cwltool INFO: Final process status is success
All 1 test(s) executed passed.
seqtk_seq_0: passed
$ planemo test --no-container --engine toil seqtk_seq.cwl
Enable beta testing mode for testing.
All 1 test(s) executed passed.
seqtk_seq_0: passed
```

For more information on the Common Workflow Language check out the Draft 3 User Guide and Specification.

6.2 Building Common Workflow Language Tools (Using the Planemo Appliance)

This tutorial is a gentle introduction to writing Common Workflow Language tools using the Planemo virtual appliance (available for Docker and Vagrant). Check out ` these instructions <https://planemo.readthedocs.org/en/latest/appliance.html>`__ for obtaining the virtual appliance if you have not done so already.

6.2.1 The Basics

This guide is going to demonstrate building up tools for commands from Heng Li's Seqtk package - a package for processing sequence data in FASTA and FASTQ files.

To get started let's install Seqtk. Here we are going to use conda to install Seqtk - but however you obtain it should be fine.

```
$ conda install --force --yes -c conda-forge -c bioconda seqtk=1.2
    ... seqtk installation ...
$ seqtk seq
        Usage:   seqtk seq [options] <in.fq>|<in.fa>
        Options: -q INT    mask bases with quality lower than INT [0]
                 -X INT    mask bases with quality higher than INT [255]
                 -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
                 -l INT    number of residues per line; 0 for 2^32-1 [0]
                 -Q INT    quality shift: ASCII-INT gives base quality [33]
                 -s INT    random seed (effective with -f) [11]
                 -f FLOAT  sample FLOAT fraction of sequences [1]
                 -M FILE   mask regions in BED or name list FILE [null]
                 -L INT    drop sequences with length shorter than INT [0]
                 -c        mask complement region (effective with -M)
                 -r        reverse complement
                 -A        force FASTA output (discard quality)
                 -C        drop comments at the header lines
                 -N        drop sequences containing ambiguous bases
                 -1        output the 2n-1 reads only
                 -2        output the 2n reads only
                 -V        shift quality by '(-Q) - 33'
```

Next we will download an example FASTQ file and test out the a simple Seqtk command - seq which converts FASTQ files into FASTA.

```
$ wget https://raw.githubusercontent.com/galaxyproject/galaxy-test-data/master/2.fastq
$ seqtk seq -A 2.fastq > 2.fasta
$ cat 2.fasta
>EAS54_6_R1_2_1_413_324
CCCTTCTTGTCTTCAGCGTTTCTCC
>EAS54_6_R1_2_1_540_792
TTGGCAGGCCAAGGCCGATGGATCA
>EAS54_6_R1_2_1_443_348
GTTGCTTCTGGCGTGGGTGGGGGGG
```

Common Workflow Language tool files are just simple YAML files, so at this point one could just open a text editor and start implementing the tool. Planemo has a command `tool_init` to quickly generate a skeleton to work from, so let's start by doing that.

```
$ planemo tool_init --cwl --id 'seqtk_seq' --name 'Convert to FASTA (seqtk)'
```

The `tool_init` command can take various complex arguments - but three two most basic ones are shown above `--cwl`, `--id` and `--name`. The `--cwl` flag tells Planemo to generate a Common Workflow Language tool. `--id` is a short identifier for this tool and it should be unique across all tools. `--name` is a short, human-readable name for the the tool - it corresponds to the `label` attribute in the CWL tool document.

The above command will generate the file `seqtk_seq.cwl` - which should look like this.

```
#!/usr/bin/env cwl-runner
cwlVersion: 'v1.0'
class: CommandLineTool
id: "seqtk_seq"
label: "Convert to FASTA (seqtk)"
inputs: [] # TODO
outputs: [] # TODO
baseCommand: []
arguments: []
doc: |
   TODO: Fill in description.
```

This tool file has the common fields required for a CWL tool with TODO notes, but you will still need to open up the editor and fill out the command, describe input parameters, tool outputs, writeup usage documentation (`doc`), etc..

The `tool_init` command can do a little bit better than this as well. We can use the test command we tried above `seqtk seq -A 2.fastq > 2.fasta` as an example to generate a command block by specifing the inputs and the outputs as follows.

```
$ planemo tool_init --force \
                    --cwl \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta
```

This will generate the following CWL tool definition - which now has correct definitions for the input, output, and command specified. These represent a best guess by planemo, and in most cases will need to be tweaked manually after the tool is generated.

```
#!/usr/bin/env cwl-runner
cwlVersion: 'v1.0'
class: CommandLineTool
id: "seqtk_seq"
label: "Convert to FASTA (seqtk)"
inputs:
  input1:
    type: File
    doc: |
      TODO
    inputBinding:
      position: 1
      prefix: "-a"
outputs:
  output1:
    type: File
    outputBinding:
      glob: out
baseCommand:
  - "seqtk"
  - "seq"
arguments: []
stdout: out
```

(continues on next page)

(continued from previous page)

```
doc: |
   TODO: Fill in description.
```

As shown at the beginning of this section, the command `seqtk seq` generates a help message for the `seq` command. `tool_init` can take that help message and stick it right in the generated tool file using the `help_from_command` option.

Generally command help messages aren't exactly appropriate for tools since they mention argument names and simillar details that are abstracted away by the tool - but they can be an excellent place to start.

The following Planemo's `tool_init` call has been enhanced to use `--help_from_command`.

```
$ planemo tool_init --force \
                    --cwl \
                    --id 'seqtk_seq' \
                    --name 'Convert to FASTA (seqtk)' \
                    --example_command 'seqtk seq -A 2.fastq > 2.fasta' \
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --requirement seqtk@1.2 \
                    --container 'quay.io/biocontainers/seqtk:1.2--0' \
                    --test_case \
                    --help_from_command 'seqtk seq'
```

This command generates the following CWL YAML file.

```
#!/usr/bin/env cwl-runner
cwlVersion: 'v1.0'
class: CommandLineTool
id: "seqtk_seq"
label: "Convert to FASTA (seqtk)"
hints:
  DockerRequirement:
    dockerPull: quay.io/biocontainers/seqtk:1.2--1
  SoftwareRequirement:
    packages:
    - package: seqtk
      version:
      - "1.2"
inputs:
  input1:
    type: File
    doc: |
      TODO
    inputBinding:
      position: 1
      prefix: "-a"
outputs:
  output1:
    type: File
    outputBinding:
      glob: out
baseCommand:
```

(continues on next page)

(continued from previous page)

```
  - "seqtk"
  - "seq"
arguments: []
stdout: out
doc: |

  Usage:   seqtk seq [options] <in.fq>|<in.fa>

  Options: -q INT    mask bases with quality lower than INT [0]
           -X INT    mask bases with quality higher than INT [255]
           -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
           -l INT    number of residues per line; 0 for 2^32-1 [0]
           -Q INT    quality shift: ASCII-INT gives base quality [33]
           -s INT    random seed (effective with -f) [11]
           -f FLOAT  sample FLOAT fraction of sequences [1]
           -M FILE   mask regions in BED or name list FILE [null]
           -L INT    drop sequences with length shorter than INT [0]
           -c        mask complement region (effective with -M)
           -r        reverse complement
           -A        force FASTA output (discard quality)
           -C        drop comments at the header lines
           -N        drop sequences containing ambiguous bases
           -1        output the 2n-1 reads only
           -2        output the 2n reads only
           -V        shift quality by '(-Q) - 33'
           -U        convert all bases to uppercases
           -S        strip of white spaces in sequences
```

In addition to generating a CWL tool adding the `--test_case` flag generates from more files that are useful including `seqtk_seq_job.yml` as shown below:

```
input1:
  class: File
  path: test-data/2.fastq
```

This is a CWL job input document and should allow you to run the example command using any CWL implementation. For instance if you have cwltool (`cwltool`) or Toil (`cwltoil`) on your `PATH` the following examples should work.

```
$ cwltool seqtk_seq.cwl seqtk_seq_job.yml
/Users/john/workspace/planemo/.venv/bin/cwltool 1.0.20180508202931
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl'
[job seqtk_seq.cwl] /private/tmp/docker_tmpXgtSLt$ docker \
    run \
    -i \
    --volume=/private/tmp/docker_tmpXgtSLt:/private/var/spool/cwl:rw \
    --volume=/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpGG1thW:/tmp:rw \
    --volume=/Users/john/tool_init_exercise/test-data/2.fastq:/private/var/lib/cwl/
↪stg7db12d3a-2375-42ed-ba60-8a0ef69ffe80/2.fastq:ro \
    --workdir=/private/var/spool/cwl \
    --read-only=true \
    --log-driver=none \
    --user=502:20 \
```

(continues on next page)

(continued from previous page)

```
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/private/var/spool/cwl \
    quay.io/biocontainers/seqtk:1.2--1 \
    seqtk \
    seq \
    -A \
    /private/var/lib/cwl/stg7db12d3a-2375-42ed-ba60-8a0ef69ffe80/2.fastq > /private/tmp/
↪docker_tmpXgtSLt/out
[job seqtk_seq.cwl] completed success
{
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "location": "file:///Users/john/tool_init_exercise/out",
        "path": "/Users/john/tool_init_exercise/out",
        "class": "File",
        "size": 150
    }
}
```

```
$ cwltoil seqtk_seq.cwl seqtk_seq_job.yml
jlaptop17.local 2018-05-21 15:25:30,630 MainThread INFO toil.lib.bioio: Root logger is␣
↪at level 'INFO', 'toil' logger at level 'INFO'.
jlaptop17.local 2018-05-21 15:25:30,648 MainThread INFO toil.jobStores.abstractJobStore:␣
↪The workflow ID is: '55a08d91-1852-4069-97a9-741abd2ea04e'
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl'
jlaptop17.local 2018-05-21 15:25:30,650 MainThread INFO cwltool: Resolved 'seqtk_seq.cwl
↪' to 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl'
jlaptop17.local 2018-05-21 15:25:31,793 MainThread INFO toil.common: Using the single␣
↪machine batch system
jlaptop17.local 2018-05-21 15:25:31,793 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxCores to CPU count of system (8).
jlaptop17.local 2018-05-21 15:25:31,793 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxMemory to physically available memory (17179869184).
jlaptop17.local 2018-05-21 15:25:31,800 MainThread INFO toil.common: Created the␣
↪workflow directory at /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/toil-55a08d91-
↪1852-4069-97a9-741abd2ea04e-132281828025877
jlaptop17.local 2018-05-21 15:25:31,800 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxDisk to physically available disk (206962089984).
jlaptop17.local 2018-05-21 15:25:31,808 MainThread INFO toil.common: User script␣
↪ModuleDescriptor(dirPath='/Users/john/workspace/planemo/.venv/lib/python2.7/site-
↪packages', name='toil.cwl.cwltoil', fromVirtualEnv=True) belongs to Toil. No need to␣
↪auto-deploy it.
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: No user script to␣
↪auto-deploy.
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: Written the␣
↪environment for the jobs to the environment file
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: Caching all jobs in␣
↪job store
jlaptop17.local 2018-05-21 15:25:31,809 MainThread INFO toil.common: 0 jobs downloaded.
jlaptop17.local 2018-05-21 15:25:31,825 MainThread INFO toil: Running Toil version 3.15.
```

(continues on next page)

(continued from previous page)

```
↪0-0e3a87e738f5e0e7cff64bfdad337d592bd92704.
jlaptop17.local 2018-05-21 15:25:31,825 MainThread INFO toil.realtimeLogger: Real-time␣
↪logging disabled
jlaptop17.local 2018-05-21 15:25:31,832 MainThread INFO toil.toilState: (Re)building␣
↪internal scheduler state
2018-05-21 15:25:31,832 - toil.toilState - INFO - (Re)building internal scheduler state
jlaptop17.local 2018-05-21 15:25:31,832 MainThread INFO toil.leader: Found 1 jobs to␣
↪start and 0 jobs with successors to run
2018-05-21 15:25:31,832 - toil.leader - INFO - Found 1 jobs to start and 0 jobs with␣
↪successors to run
jlaptop17.local 2018-05-21 15:25:31,832 MainThread INFO toil.leader: Checked batch␣
↪system has no running jobs and no updated jobs
2018-05-21 15:25:31,832 - toil.leader - INFO - Checked batch system has no running jobs␣
↪and no updated jobs
jlaptop17.local 2018-05-21 15:25:31,833 MainThread INFO toil.leader: Starting the main␣
↪loop
2018-05-21 15:25:31,833 - toil.leader - INFO - Starting the main loop
jlaptop17.local 2018-05-21 15:25:31,834 MainThread INFO toil.leader: Issued job 'file:///
↪Users/john/tool_init_exercise/seqtk_seq.cwl' seqtk seq u/c/jobzYKQ3V with job batch␣
↪system ID: 0 and cores: 1, disk: 3.0 G, and memory: 2.0 G
2018-05-21 15:25:31,834 - toil.leader - INFO - Issued job 'file:///Users/john/tool_init_
↪exercise/seqtk_seq.cwl' seqtk seq u/c/jobzYKQ3V with job batch system ID: 0 and cores:␣
↪1, disk: 3.0 G, and memory: 2.0 G
jlaptop17.local 2018-05-21 15:25:33,953 MainThread INFO toil.leader: Job ended␣
↪successfully: 'file:///Users/john/tool_init_exercise/seqtk_seq.cwl' seqtk seq u/c/
↪jobzYKQ3V
2018-05-21 15:25:33,953 - toil.leader - INFO - Job ended successfully: 'file:///Users/
↪john/tool_init_exercise/seqtk_seq.cwl' seqtk seq u/c/jobzYKQ3V
jlaptop17.local 2018-05-21 15:25:33,955 MainThread INFO toil.leader: Finished the main␣
↪loop: no jobs left to run
2018-05-21 15:25:33,955 - toil.leader - INFO - Finished the main loop: no jobs left to␣
↪run
jlaptop17.local 2018-05-21 15:25:33,955 MainThread INFO toil.serviceManager: Waiting for␣
↪service manager thread to finish ...
2018-05-21 15:25:33,955 - toil.serviceManager - INFO - Waiting for service manager␣
↪thread to finish ...
jlaptop17.local 2018-05-21 15:25:34,841 MainThread INFO toil.serviceManager: ...␣
↪finished shutting down the service manager. Took 0.885795116425 seconds
2018-05-21 15:25:34,841 - toil.serviceManager - INFO - ... finished shutting down the␣
↪service manager. Took 0.885795116425 seconds
jlaptop17.local 2018-05-21 15:25:34,842 MainThread INFO toil.statsAndLogging: Waiting␣
↪for stats and logging collator thread to finish ...
2018-05-21 15:25:34,842 - toil.statsAndLogging - INFO - Waiting for stats and logging␣
↪collator thread to finish ...
jlaptop17.local 2018-05-21 15:25:34,854 MainThread INFO toil.statsAndLogging: ...␣
↪finished collating stats and logs. Took 0.0120511054993 seconds
2018-05-21 15:25:34,854 - toil.statsAndLogging - INFO - ... finished collating stats and␣
↪logs. Took 0.0120511054993 seconds
jlaptop17.local 2018-05-21 15:25:34,855 MainThread INFO toil.leader: Finished toil run␣
↪successfully
2018-05-21 15:25:34,855 - toil.leader - INFO - Finished toil run successfully
{
```

(continues on next page)

(continued from previous page)

```
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "nameext": "",
        "nameroot": "out",
        "http://commonwl.org/cwltool#generation": 0,
        "location": "file:///Users/john/tool_init_exercise/out",
        "class": "File",
        "size": 150
    }
jlaptop17.local 2018-05-21 15:25:34,866 MainThread INFO toil.common: Successfully␣
↪deleted the job store: <toil.jobStores.fileJobStore.FileJobStore object at 0x1057205d0>
}2018-05-21 15:25:34,866 - toil.common - INFO - Successfully deleted the job store:
↪<toil.jobStores.fileJobStore.FileJobStore object at 0x1057205d0>
```

At this point we have a fairly a functional CWL tool with test and usage documentation. This was a pretty simple example - usually you will need to put more work into the tool to get to this point - `tool_init` is really just designed to get you started.

Now lets lint and test the tool we have developed. The Planemo's `lint` (or just `l`) command will review tool for validity, obvious mistakes, and Planemo "best practices".

```
$ planemo l seqtk_seq.cwl
Linting tool /Users/john/workspace/planemo/docs/writing/seqtk_seq.cwl
Applying linter general... CHECK
.. CHECK: Tool defines a version [0.0.1].
.. CHECK: Tool defines a name [Convert to FASTA (seqtk)].
.. CHECK: Tool defines an id [seqtk_seq_v3].
.. CHECK: Tool specifies profile version [16.04].
Applying linter cwl_validation... CHECK
.. INFO: CWL appears to be valid.
Applying linter docker_image... CHECK
.. INFO: Tool will run in Docker image [quay.io/biocontainers/seqtk:1.2--1].
Applying linter new_draft... CHECK
.. INFO: Modern CWL version [v1.0]
```

In addition to the actual tool and job files, `--test_case` caused a test file to be generated using the example command and provided test data. The file contents are as follows:

```
- doc: test generated from example command
  job: seqtk_seq_job.yml
  outputs:
    output1:
      path: test-data/2.fasta
```

Unlike the job file, this file is a Planemo-specific artifact. This file may contain 1 or more tests - each test is an element of the top-level list. `tool_init` will use the example command to build just one test.

Each test consists of a few parts:

- `doc` - this attribute provides a short description for the test.
- `job` - this can be the path to a CWL job description or a job description embedded right in the test (`tool_init` builds the latter).

- `outputs` - this section describes the expected output for a test. Each output ID of the tool or workflow under test can appear as a key. The example above just describes expected specific output file contents exactly but many more expectations can be described.

For more information on the test file format check out the Test Format docs.

The tests described in this file can be run using the `planemo test` command on the original file.

```
$ planemo test --no-container seqtk_seq.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20180508202931
cwltool INFO: Resolved '/Users/john/tool_init_exercise/seqtk_seq.cwl' to 'file:///Users/
↪john/tool_init_exercise/seqtk_seq.cwl'
cwltool INFO: [job seqtk_seq.cwl] /private/tmp/docker_tmpvLE9SS$ seqtk \
    seq \
    -A \
    /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpGM22d_/stg0c0cad75-7ca0-
↪4f3a-9d77-63e9c49f5353/2.fastq > /private/tmp/docker_tmpvLE9SS/out
cwltool INFO: [job seqtk_seq.cwl] completed success
cwltool INFO: Final process status is success
All 1 test(s) executed passed.
seqtk_seq_0: passed
```

This is a bit a different than running the job. For one thing, we don't need to specify an input job - instead Planemo will automatically find the test file and run all the jobs described inside that file. Additionally, Planemo will check the outputs to ensure the match the test expectations.

In addition to the in console display of test results as red (failing) or green (passing), Planemo also creates an HTML report for the test results by default. Many more test report options are available such `--test_output_xunit` which is useful in certain continuous integration environments. See `planemo test --help` for more options, as well as the `test_reports` command.

The above test example used cwltool to run our test and disabled containerization. By dropping the `--no-container` argument we can run the tool in a Docker container. By passing an engine argument as `--engine toil` we can run our test in Toil, an alternative CWL implementation.

```
$ planemo test seqtk_seq.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20180508202931
cwltool INFO: Resolved '/Users/john/tool_init_exercise/seqtk_seq.cwl' to 'file:///Users/
↪john/tool_init_exercise/seqtk_seq.cwl'
cwltool INFO: [job seqtk_seq.cwl] /private/tmp/docker_tmpUeIpXJ$ docker \
    run \
    -i \
    --volume=/private/tmp/docker_tmpUeIpXJ:/private/var/spool/cwl:rw \
    --volume=/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpteo_2Z:/tmp:rw \
    --volume=/Users/john/tool_init_exercise/test-data/2.fastq:/private/var/lib/cwl/
↪stg939ee60b-a194-4177-8410-c40a1acb38ea/2.fastq:ro \
    --workdir=/private/var/spool/cwl \
    --read-only=true \
    --log-driver=none \
    --user=502:20 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/private/var/spool/cwl \
    quay.io/biocontainers/seqtk:1.2--1 \
```

(continues on next page)

(continued from previous page)

```
    seqtk \
    seq \
    -A \
    /private/var/lib/cwl/stg939ee60b-a194-4177-8410-c40a1acb38ea/2.fastq > /private/tmp/
↪docker_tmpUeIpXJ/out
cwltool INFO: [job seqtk_seq.cwl] completed success
cwltool INFO: Final process status is success
All 1 test(s) executed passed.
seqtk_seq_0: passed
$ planemo test --no-container --engine toil seqtk_seq.cwl
Enable beta testing mode for testing.
All 1 test(s) executed passed.
seqtk_seq_0: passed
```

For more information on the Common Workflow Language check out the Draft 3 User Guide and Specification.

Additional tutorials include

6.3 Advanced Tool Development Topics

This tutorial covers some more advanced tool development topics. It assumes some basic knowledge about developing CWL tools and that you have an environment with Planemo available - check out the CWL User Guide CWL and the Planemo CWL intro tutorial if you have never developed a CWL tool.

6.3.1 Dependencies and Conda

Specifying and Using Software Requirements

Note: Planemo requires a Conda installation to target with its various Conda related commands. A properly configured Conda installation can be initialized with the `conda_init` command. This should only need to be executed once per development machine.

```
$ planemo conda_init
galaxy.tools.deps.conda_util INFO: Installing conda, this may take several minutes.
wget -q --recursive -O /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/conda_
↪installLW5zn1.sh https://repo.continuum.io/miniconda/Miniconda3-4.3.31-MacOSX-x86_64.sh
bash /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/conda_installLW5zn1.sh -b -p /
↪Users/john/miniconda3
PREFIX=/Users/john/miniconda3
installing: python-3.6.3-h47c878a_7 ...
Python 3.6.3 :: Anaconda, Inc.
installing: ca-certificates-2017.08.26-ha1e5d58_0 ...
installing: conda-env-2.6.0-h36134e3_0 ...
installing: libcxxabi-4.0.1-hebd6815_0 ...
installing: tk-8.6.7-h35a86e2_3 ...
installing: xz-5.2.3-h0278029_2 ...
installing: yaml-0.1.7-hc338f04_2 ...
installing: zlib-1.2.11-hf3cbc9b_2 ...
installing: libcxx-4.0.1-h579ed51_0 ...
```

(continues on next page)

(continued from previous page)

```
installing: openssl-1.0.2n-hdbc3d79_0 ...
installing: libffi-3.2.1-h475c297_4 ...
installing: ncurses-6.0-hd04f020_2 ...
installing: libedit-3.1-hb4e282d_0 ...
installing: readline-7.0-hc1231fa_4 ...
installing: sqlite-3.20.1-h7e4c145_2 ...
installing: asn1crypto-0.23.0-py36h782d450_0 ...
installing: certifi-2017.11.5-py36ha569be9_0 ...
installing: chardet-3.0.4-py36h96c241c_1 ...
installing: idna-2.6-py36h8628d0a_1 ...
installing: pycosat-0.6.3-py36hee92d8f_0 ...
installing: pycparser-2.18-py36h724b2fc_1 ...
installing: pysocks-1.6.7-py36hfa33cec_1 ...
installing: python.app-2-py36h54569d5_7 ...
installing: ruamel_yaml-0.11.14-py36h9d7ade0_2 ...
installing: six-1.11.0-py36h0e22d5e_1 ...
installing: cffi-1.11.2-py36hd3e6348_0 ...
installing: setuptools-36.5.0-py36h2134326_0 ...
installing: cryptography-2.1.4-py36h842514c_0 ...
installing: wheel-0.30.0-py36h5eb2c71_1 ...
installing: pip-9.0.1-py36h1555ced_4 ...
installing: pyopenssl-17.5.0-py36h51e4350_0 ...
installing: urllib3-1.22-py36h68b9469_0 ...
installing: requests-2.18.4-py36h4516966_1 ...
installing: conda-4.3.31-py36_0 ...
installation finished.
/Users/john/miniconda3/bin/conda install -y --override-channels --channel iuc --channel␣
↪conda-forge --channel bioconda --channel defaults conda=4.3.33 conda-build=2.1.18
Fetching package metadata ...................
Solving package specifications: .

Package plan for installation in environment /Users/john/miniconda3:

The following NEW packages will be INSTALLED:

    beautifulsoup4: 4.6.0-py36_0  conda-forge
    conda-build:    2.1.18-py36_0 conda-forge
    conda-verify:   2.0.0-py36_0  conda-forge
    filelock:       3.0.4-py36_0  conda-forge
    jinja2:         2.10-py36_0   conda-forge
    markupsafe:     1.0-py36_0    conda-forge
    pkginfo:        1.4.2-py36_0  conda-forge
    pycrypto:       2.6.1-py36_1  conda-forge
    pyyaml:         3.12-py36_1   conda-forge

The following packages will be UPDATED:

    conda:          4.3.31-py36_0             --> 4.3.33-py36_0 conda-forge

beautifulsoup4 100% |###################################################################
↪#| Time: 0:00:00 782.08 kB/s
filelock-3.0.4 100% |###################################################################
```

(continues on next page)

(continued from previous page)

```
↪#| Time: 0:00:00   7.95 MB/s
markupsafe-1.0 100% |#################################################################
↪#| Time: 0:00:00   5.82 MB/s
pkginfo-1.4.2- 100% |#################################################################
↪#| Time: 0:00:00   1.18 MB/s
pycrypto-2.6.1 100% |#################################################################
↪#| Time: 0:00:00   1.69 MB/s
pyyaml-3.12-py 100% |#################################################################
↪#| Time: 0:00:00   3.31 MB/s
conda-verify-2 100% |#################################################################
↪#| Time: 0:00:00   6.91 MB/s
jinja2-2.10-py 100% |#################################################################
↪#| Time: 0:00:00   2.81 MB/s
conda-4.3.33-p 100% |#################################################################
↪#| Time: 0:00:00 621.27 kB/s
conda-build-2. 100% |#################################################################
↪#| Time: 0:00:00   2.16 MB/s
Conda installation succeeded - Conda is available at '/Users/john/miniconda3/bin/conda'
```

Note: Why not just use containers?

Containers are great, use containers (be it Docker, Singularity, etc.) whenever possible to increase reproducibility and portability of your tools and workflow. Building ad hoc containers to support CWL tools (e.g. custom `Dockerfile` definitions) has serious limitations, in the next tutorial on containers we will argue that using Biocontainers built or discovered from your tool's Software Requirements is a superior approach.

Besides leading to better containers, there are other reasons to describe Software Requirements also - it will allow your tool to be used in environments without container runtimes available and provides valuable and actionable metadata about the computation described by the tool.

Read more about this whole dependency stack in our preprint Practical computational reproducibility in the life sciences

The Common Workflow Language specification loosely describes Software Requirements - a way to map CWL hints to packages, environment modules, or any other mechanism to describe dependencies for running a tool outside of a container. The large and active Galaxy tool development community has built an open source library and set of best practices for describing dependencies for Galaxy that should work just as well for CWL. The library has been integrated with cwltool and Toil to enable CWL tool authors and users to leverage the power and flexibility of the Galaxy dependency management and best practices.

While Software Requirements can be configured to resolve dependencies various ways, Planemo is configured with opinionated defaults geared at making building CWL tools that target Conda as easy as possible and build tools with requirements compatible with cwltool and Toil when running outside containers.

During the tool development introductory tutorial, we called `planemo tool_init` with the argument `--requirement seqtk@1.2` and the resulting tool contained such a `SoftwareRequirement` in the form the following the YAML fragment:

```
SoftwareRequirement:
  packages:
  - package: seqtk
    version:
    - "1.2"
```

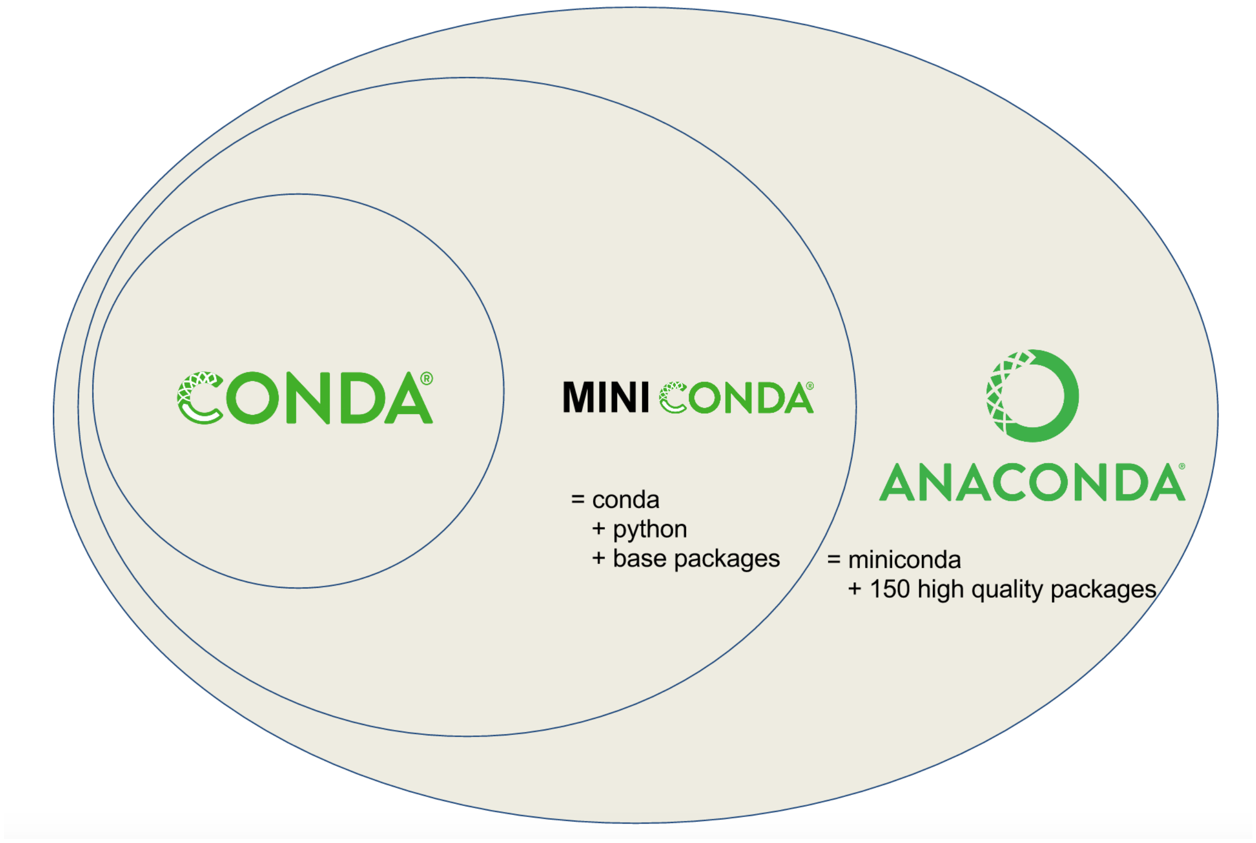
Planemo (and cwltool and Toil) can interpret these `SoftwareRequirement` annotations in various ways including as Conda packages. When interpreting these as Conda packages these runtimes can setup isolated, reproducible Conda environments for tool execution with the correct packages installed (e.g. `seqtk` in the above example).

Note: *Why Conda?*

Many different package managers could potentially be targeted here, but we focus on Conda for a few key reasons.

- No compilation at install time - binaries with their dependencies and libraries
- Support for all operating systems
- Easy to manage multiple versions of the same recipe
- HPC-ready: no root privileges needed
- Easy-to-write YAML recipes
- Viberant communities

Note: **Conda Terminology**



Conda *recipes* build *packages* that are published to *channels*.

Planemo is setup to target a few channels by default, these include `iuc`, `bioconda`, `conda_forge`, `defaults` - the whole dependency management scheme outlined here works a lot better if packages can be found in one of these "best practice" channels.

We can check if the requirements on a tool are available in best practice Conda channels using an extended form of the `planemo lint` command (`planemo lint` was introduced in the introductory tutorial). Passing `--conda_requirements` flag will ensure all listed requirements are found.

```
$ planemo lint --conda_requirements seqtk_seq.cwl
Linting tool /Users/john/workspace/planemo/docs/writing/seqtk_seq.cwl
  ...
Applying linter requirements_in_conda... CHECK
.. INFO: Requirement [seqtk@1.2] matches target in best practice Conda channel [https://
↪conda.anaconda.org/bioconda/osx-64].
```

Note: You can download a more complete version of the CWL seqtk seq from the Planemo tutorial using the command:

```
$ planemo project_init --template=seqtk_complete_cwl seqtk_example
$ cd seqtk_example
```

We can verify these tool requirements install with the `conda_install` command. With its default parameters `conda_install` processes tools and creates isolated environments for their declared Software Requirements (mirroring what can be done in production with cwltool and Toil).

```
$ planemo conda_install seqtk_seq.cwl
Install conda target CondaTarget[seqtk,version=1.2]
/home/john/miniconda3/bin/conda create -y --name __seqtk@1.2 seqtk=1.2
Fetching package metadata ...............
Solving package specifications: .........

Package plan for installation in environment /home/john/miniconda2/envs/__seqtk@1.2:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    seqtk-1.2                  |                0          29 KB  bioconda

The following NEW packages will be INSTALLED:

    seqtk: 1.2-0   bioconda
    zlib:  1.2.8-3

Fetching packages ...
seqtk-1.2-0.ta 100% |###############################################################|␣
↪Time: 0:00:00 444.71 kB/s
Extracting packages ...
[      COMPLETE      ]|###################################################################
↪##############| 100%
Linking packages ...
[      COMPLETE      ]|###################################################################
↪##############| 100%
#
# To activate this environment, use:
# > source activate __seqtk@1.2
#
```

(continues on next page)

(continued from previous page)

```
# To deactivate this environment, use:
# > source deactivate __seqtk@1.2
#
$ which seqtk
seqtk not found
$
```

The above install worked properly, but `seqtk` is not on your `PATH` because this merely created an environment within the Conda directory for the seqtk installation. Planemo will configure cwltool during testing to reuse this environment. If you wish to interactively explore the resulting enviornment to explore the installed tool or produce test data the output of the `conda_env` command can be sourced.

```
$ . <(planemo conda_env seqtk_seq.cwl)
Deactivate environment with conda_env_deactivate
(seqtk_seq) $ which seqtk
/home/planemo/miniconda3/envs/
↪jobdepsiJClEUfecc6d406196737781ff4456ec60975c137e04884e4f4b05dc68192f7cec4656/bin/seqtk
(seqtk_seq) $ seqtk seq

Usage:   seqtk seq [options] <in.fq>|<in.fa>

Options: -q INT    mask bases with quality lower than INT [0]
         -X INT    mask bases with quality higher than INT [255]
         -n CHAR   masked bases converted to CHAR; 0 for lowercase [0]
         -l INT    number of residues per line; 0 for 2^32-1 [0]
         -Q INT    quality shift: ASCII-INT gives base quality [33]
         -s INT    random seed (effective with -f) [11]
         -f FLOAT  sample FLOAT fraction of sequences [1]
         -M FILE   mask regions in BED or name list FILE [null]
         -L INT    drop sequences with length shorter than INT [0]
         -c        mask complement region (effective with -M)
         -r        reverse complement
         -A        force FASTA output (discard quality)
         -C        drop comments at the header lines
         -N        drop sequences containing ambiguous bases
         -1        output the 2n-1 reads only
         -2        output the 2n reads only
         -V        shift quality by '(-Q) - 33'
         -U        convert all bases to uppercases
         -S        strip of white spaces in sequences
(seqtk_seq) $ conda_env_deactivate
$
```

As shown above the `conda_env_deactivate` will be created in this environment and can be used to restore your initial shell configuration.

Here is a portion of the output from the testing command `planemo test seqtk_seq.cwl` demonstrating using this tool.

```
$ planemo test --no-container seqtk_seq.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20170828135420
cwltool INFO: Resolved '/Users/john/workspace/planemo/project_templates/seqtk_complete_
```

(continues on next page)

(continued from previous page)

```
↪cwl/seqtk_seq.cwl' to 'file:///Users/john/workspace/planemo/project_templates/seqtk_
↪complete_cwl/seqtk_seq.cwl'
cwltool INFO: [job seqtk_seq.cwl] /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/
↪T/tmpaDQ1nK$ seqtk \
    seq \
    -a \
    /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpJtPKCr/stg24cf7e67-5ca6-
↪44a4-a46b-26cbe104e1d4/2.fastq > /private/var/folders/78/
↪zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpaDQ1nK/out
cwltool INFO: [job seqtk_seq.cwl] completed success
cwltool INFO: Final process status is success
galaxy.tools.parser.factory INFO: Loading CWL tool - this is experimental - tool likely␣
↪will not function in future at least in same way.
All 1 test(s) executed passed.
seqtk_seq_0: passed
```

Since seqtk isn't on the path and we did not use a container, we can see the SoftwareRequirement resolution was successful and it found the environment we previously installed with conda_install.

This can be used outside of Planemo testing as well, the following invocation shows running a job with cwltool using an environment like the one created above:

```
$ cwltool --no-container --beta-conda-dependencies seqtk_seq.cwl seqtk_seq_job.yml
/Users/john/workspace/planemo/.venv/bin/cwltool 1.0.20180508202931
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/workspace/planemo/project_templates/
↪seqtk_complete_cwl/seqtk_seq.cwl'
No handlers could be found for logger "rdflib.term"
[job seqtk_seq.cwl] /private/tmp/docker_tmpDQYeqC$ seqtk \
    seq \
    -a \
    /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpQwBqPo/stg8cf2282a-d807-
↪4f90-b94d-feeda004cacd/2.fastq > /private/tmp/docker_tmpDQYeqC/out
PREFIX=/Users/john/workspace/planemo/project_templates/seqtk_complete_cwl/cwltool_deps/_
↪conda
installing: python-3.6.3-h47c878a_7 ...
Python 3.6.3 :: Anaconda, Inc.
installing: ca-certificates-2017.08.26-ha1e5d58_0 ...
installing: conda-env-2.6.0-h36134e3_0 ...
installing: libcxxabi-4.0.1-hebd6815_0 ...
installing: tk-8.6.7-h35a86e2_3 ...
installing: xz-5.2.3-h0278029_2 ...
installing: yaml-0.1.7-hc338f04_2 ...
installing: zlib-1.2.11-hf3cbc9b_2 ...
installing: libcxx-4.0.1-h579ed51_0 ...
installing: openssl-1.0.2n-hdbc3d79_0 ...
installing: libffi-3.2.1-h475c297_4 ...
installing: ncurses-6.0-hd04f020_2 ...
installing: libedit-3.1-hb4e282d_0 ...
installing: readline-7.0-hc1231fa_4 ...
installing: sqlite-3.20.1-h7e4c145_2 ...
installing: asn1crypto-0.23.0-py36h782d450_0 ...
installing: certifi-2017.11.5-py36ha569be9_0 ...
installing: chardet-3.0.4-py36h96c241c_1 ...
```

(continues on next page)

(continued from previous page)

```
installing: idna-2.6-py36h8628d0a_1 ...
installing: pycosat-0.6.3-py36hee92d8f_0 ...
installing: pycparser-2.18-py36h724b2fc_1 ...
installing: pysocks-1.6.7-py36hfa33cec_1 ...
installing: python.app-2-py36h54569d5_7 ...
installing: ruamel_yaml-0.11.14-py36h9d7ade0_2 ...
installing: six-1.11.0-py36h0e22d5e_1 ...
installing: cffi-1.11.2-py36hd3e6348_0 ...
installing: setuptools-36.5.0-py36h2134326_0 ...
installing: cryptography-2.1.4-py36h842514c_0 ...
installing: wheel-0.30.0-py36h5eb2c71_1 ...
installing: pip-9.0.1-py36h1555ced_4 ...
installing: pyopenssl-17.5.0-py36h51e4350_0 ...
installing: urllib3-1.22-py36h68b9469_0 ...
installing: requests-2.18.4-py36h4516966_1 ...
installing: conda-4.3.31-py36_0 ...
installation finished.
Fetching package metadata .................
Solving package specifications: .

Package plan for installation in environment /Users/john/workspace/planemo/project_
↪templates/seqtk_complete_cwl/cwltool_deps/_conda:

The following packages will be UPDATED:

    conda: 4.3.31-py36_0 --> 4.3.33-py36_0 conda-forge

conda-4.3.33-p 100% |################################################################|␣
↪Time: 0:00:00   1.13 MB/s


Package plan for installation in environment /Users/john/workspace/planemo/project_
↪templates/seqtk_complete_cwl/cwltool_deps/_conda/envs/__seqtk@1.2:

The following NEW packages will be INSTALLED:

    seqtk: 1.2-1    bioconda
    zlib:  1.2.11-0 conda-forge


[job seqtk_seq.cwl] completed success
{
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "location": "file:///Users/john/workspace/planemo/project_templates/seqtk_
↪complete_cwl/out",
        "path": "/Users/john/workspace/planemo/project_templates/seqtk_complete_cwl/out",
        "class": "File",
        "size": 150
    }
}
```

(continues on next page)

(continued from previous page)

```
Final process status is success
```

This demonstrates that cwltool will install the packages needed on the first run, if we rerun cwltool it will reuse that previous environment.

```
$ cwltool --no-container --beta-conda-dependencies seqtk_seq.cwl seqtk_seq_job.yml
/Users/john/workspace/planemo/.venv/bin/cwltool 1.0.20180508202931
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/workspace/planemo/project_templates/
↪seqtk_complete_cwl/seqtk_seq.cwl'
No handlers could be found for logger "rdflib.term"
[job seqtk_seq.cwl] /private/tmp/docker_tmp4vvE_i$ seqtk \
    seq \
    -a \
    /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpcvQ3Ph/stg2ef3a21c-9fb0-
↪4099-88c2-36e24719901d/2.fastq > /private/tmp/docker_tmp4vvE_i/out
[job seqtk_seq.cwl] completed success
{
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "location": "file:///Users/john/workspace/planemo/project_templates/seqtk_
↪complete_cwl/out",
        "path": "/Users/john/workspace/planemo/project_templates/seqtk_complete_cwl/out",
        "class": "File",
        "size": 150
    }
}
Final process status is success
```

And the same thing is possible with Toil.

```
$ cwltoil --no-container --beta-conda-dependencies seqtk_seq.cwl seqtk_seq_job.yml
jlaptop17.local 2018-05-23 15:27:25,754 MainThread INFO toil.lib.bioio: Root logger is␣
↪at level 'INFO', 'toil' logger at level 'INFO'.
jlaptop17.local 2018-05-23 15:27:25,785 MainThread INFO toil.jobStores.abstractJobStore:␣
↪The workflow ID is: '92328fb2-33b7-44cd-879f-41d8cbf94555'
Resolved 'seqtk_seq.cwl' to 'file:///Users/john/workspace/planemo/project_templates/
↪seqtk_complete_cwl/seqtk_seq.cwl'
jlaptop17.local 2018-05-23 15:27:25,787 MainThread INFO cwltool: Resolved 'seqtk_seq.cwl
↪' to 'file:///Users/john/workspace/planemo/project_templates/seqtk_complete_cwl/seqtk_
↪seq.cwl'
jlaptop17.local 2018-05-23 15:27:27,002 MainThread WARNING rdflib.term: http://schema.
↪org/docs/!DOCTYPE html does not look like a valid URI, trying to serialize this will␣
↪break.
jlaptop17.local 2018-05-23 15:27:27,396 MainThread INFO rdflib.plugins.parsers.pyRdfa:␣
↪Current options:
        preserve space                          : True
        output processor graph                  : True
        output default graph                    : True
        host language                           : RDFa Core
        accept embedded RDF                     : False
        check rdfa lite                         : False
```

(continues on next page)

(continued from previous page)

```
        cache vocabulary graphs                : False

jlaptop17.local 2018-05-23 15:27:29,797 MainThread INFO toil.common: Using the single␣
↪machine batch system
jlaptop17.local 2018-05-23 15:27:29,798 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxCores to CPU count of system (8).
jlaptop17.local 2018-05-23 15:27:29,798 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxMemory to physically available memory (17179869184).
jlaptop17.local 2018-05-23 15:27:29,808 MainThread INFO toil.common: Created the␣
↪workflow directory at /var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/toil-92328fb2-
↪33b7-44cd-879f-41d8cbf94555-132281828025877
jlaptop17.local 2018-05-23 15:27:29,808 MainThread WARNING toil.batchSystems.
↪singleMachine: Limiting maxDisk to physically available disk (202669449216).
jlaptop17.local 2018-05-23 15:27:29,815 MainThread INFO toil.common: User script␣
↪ModuleDescriptor(dirPath='/Users/john/workspace/planemo/.venv/lib/python2.7/site-
↪packages', name='toil.cwl.cwltoil', fromVirtualEnv=True) belongs to Toil. No need to␣
↪auto-deploy it.
jlaptop17.local 2018-05-23 15:27:29,816 MainThread INFO toil.common: No user script to␣
↪auto-deploy.
jlaptop17.local 2018-05-23 15:27:29,816 MainThread INFO toil.common: Written the␣
↪environment for the jobs to the environment file
jlaptop17.local 2018-05-23 15:27:29,816 MainThread INFO toil.common: Caching all jobs in␣
↪job store
jlaptop17.local 2018-05-23 15:27:29,816 MainThread INFO toil.common: 0 jobs downloaded.
jlaptop17.local 2018-05-23 15:27:29,911 MainThread INFO toil: Running Toil version 3.15.
↪0-0e3a87e738f5e0e7cff64bfdad337d592bd92704.
jlaptop17.local 2018-05-23 15:27:29,911 MainThread INFO toil.realtimeLogger: Real-time␣
↪logging disabled
jlaptop17.local 2018-05-23 15:27:29,937 MainThread INFO toil.toilState: (Re)building␣
↪internal scheduler state
2018-05-23 15:27:29,937 - toil.toilState - INFO - (Re)building internal scheduler state
jlaptop17.local 2018-05-23 15:27:29,938 MainThread INFO toil.leader: Found 1 jobs to␣
↪start and 0 jobs with successors to run
2018-05-23 15:27:29,938 - toil.leader - INFO - Found 1 jobs to start and 0 jobs with␣
↪successors to run
jlaptop17.local 2018-05-23 15:27:29,938 MainThread INFO toil.leader: Checked batch␣
↪system has no running jobs and no updated jobs
2018-05-23 15:27:29,938 - toil.leader - INFO - Checked batch system has no running jobs␣
↪and no updated jobs
jlaptop17.local 2018-05-23 15:27:29,938 MainThread INFO toil.leader: Starting the main␣
↪loop
2018-05-23 15:27:29,938 - toil.leader - INFO - Starting the main loop
jlaptop17.local 2018-05-23 15:27:29,939 MainThread INFO toil.leader: Issued job 'file:///
↪Users/john/workspace/planemo/project_templates/seqtk_complete_cwl/seqtk_seq.cwl' seqtk␣
↪seq e/V/jobsxUpTU with job batch system ID: 0 and cores: 1, disk: 3.0 G, and memory: 2.
↪0 G
2018-05-23 15:27:29,939 - toil.leader - INFO - Issued job 'file:///Users/john/workspace/
↪planemo/project_templates/seqtk_complete_cwl/seqtk_seq.cwl' seqtk seq e/V/jobsxUpTU␣
↪with job batch system ID: 0 and cores: 1, disk: 3.0 G, and memory: 2.0 G
jlaptop17.local 2018-05-23 15:27:31,409 MainThread INFO toil.leader: Job ended␣
↪successfully: 'file:///Users/john/workspace/planemo/project_templates/seqtk_complete_
↪cwl/seqtk_seq.cwl' seqtk seq e/V/jobsxUpTU
```

(continues on next page)

(continued from previous page)

```
2018-05-23 15:27:31,409 - toil.leader - INFO - Job ended successfully: 'file:///Users/
↪john/workspace/planemo/project_templates/seqtk_complete_cwl/seqtk_seq.cwl' seqtk seq e/
↪V/jobsxUpTU
jlaptop17.local 2018-05-23 15:27:31,411 MainThread INFO toil.leader: Finished the main␣
↪loop: no jobs left to run
2018-05-23 15:27:31,411 - toil.leader - INFO - Finished the main loop: no jobs left to␣
↪run
jlaptop17.local 2018-05-23 15:27:31,411 MainThread INFO toil.serviceManager: Waiting for␣
↪service manager thread to finish ...
2018-05-23 15:27:31,411 - toil.serviceManager - INFO - Waiting for service manager␣
↪thread to finish ...
jlaptop17.local 2018-05-23 15:27:31,946 MainThread INFO toil.serviceManager: ...␣
↪finished shutting down the service manager. Took 0.535056114197 seconds
2018-05-23 15:27:31,946 - toil.serviceManager - INFO - ... finished shutting down the␣
↪service manager. Took 0.535056114197 seconds
jlaptop17.local 2018-05-23 15:27:31,947 MainThread INFO toil.statsAndLogging: Waiting␣
↪for stats and logging collator thread to finish ...
2018-05-23 15:27:31,947 - toil.statsAndLogging - INFO - Waiting for stats and logging␣
↪collator thread to finish ...
jlaptop17.local 2018-05-23 15:27:31,960 MainThread INFO toil.statsAndLogging: ...␣
↪finished collating stats and logs. Took 0.0131621360779 seconds
2018-05-23 15:27:31,960 - toil.statsAndLogging - INFO - ... finished collating stats and␣
↪logs. Took 0.0131621360779 seconds
jlaptop17.local 2018-05-23 15:27:31,961 MainThread INFO toil.leader: Finished toil run␣
↪successfully
2018-05-23 15:27:31,961 - toil.leader - INFO - Finished toil run successfully
{
    "output1": {
        "checksum": "sha1$322e001e5a99f19abdce9f02ad0f02a17b5066c2",
        "basename": "out",
        "nameext": "",
        "nameroot": "out",
        "http://commonwl.org/cwltool#generation": 0,
        "location": "file:///Users/john/workspace/planemo/project_templates/seqtk_
↪complete_cwl/out",
        "class": "File",
        "size": 150
    }
jlaptop17.local 2018-05-23 15:27:31,972 MainThread INFO toil.common: Successfully␣
↪deleted the job store: <toil.jobStores.fileJobStore.FileJobStore object at 0x10554d490>
}2018-05-23 15:27:31,972 - toil.common - INFO - Successfully deleted the job store:
↪<toil.jobStores.fileJobStore.FileJobStore object at 0x10554d490>
```

Finding Existing Conda Packages

How did we know what software name and software version to use? We found the existing packages available for Conda and referenced them. To do this yourself, you can simply use the planemo command `conda_search`. If we do a search for `seqt` it will show all the software and all the versions available matching that search term - including `seqtk`.

```
$ planemo conda_search seqt
/Users/john/miniconda3/bin/conda search --override-channels --channel iuc --channel␣
↪conda-forge --channel bioconda --channel defaults '*seqt*'
Loading channels: done
# Name                   Version           Build  Channel
bioconductor-htseqtools         1.26.0        r3.4.1_0  bioconda
bioconductor-seqtools         1.10.0        r3.3.2_0  bioconda
bioconductor-seqtools         1.10.0        r3.4.1_0  bioconda
bioconductor-seqtools         1.12.0        r3.4.1_0  bioconda
seqtk                       r75                0  bioconda
seqtk                       r82                0  bioconda
seqtk                       r82                1  bioconda
seqtk                       r93                0  bioconda
seqtk                       1.2                0  bioconda
seqtk                       1.2                1  bioconda
```

Note: The Planemo command `conda_search` is a light wrapper around the underlying `conda search` command but configured to use the same channels and other options as Planemo and Galaxy. The following Conda command would also work to search:

```
$ $HOME/miniconda3/bin/conda -c iuc -c conda-forge -c bioconda '*seqt*'
```

For Conda versions 4.3.X or less, the search invocation would be something a bit different:

```
$ $HOME/miniconda3/bin/conda -c iuc -c conda-forge -c bioconda seqt
```

Alternatively the Anaconda website can be used to search for packages. Typing `seqtk` into the search form on that page and clicking the top result will bring on to this page with information about the Bioconda package.

When using the website to search though, you need to aware of what channel you are using. By default, Planemo and Galaxy will search a few different Conda channels. While it is possible to configure a local Planemo or Galaxy to target different channels - the current best practice is to add tools to the existing channels.

The existing channels include:

- Bioconda (github | conda) - best practice channel for various bioinformatics packages.
- Conda-Forge (github | conda) - best practice channel for general purpose and widely useful computing packages and libraries.
- iuc (github | conda) - best practice channel for other more Galaxy specific packages.

Exercise - Leveraging Bioconda

Use the `project_init` command to download this exercise.

```
$ planemo project_init --template conda_exercises_cwl conda_exercises
$ cd conda_exercises/exercise_1
$ ls
pear.cwl              test-data
```

This project template contains a few exercises. The first uses a CWL tool for PEAR - Paired-End reAd mergeR. This tool however has no `SoftwareRequirement` or container annotations and so will not work properly without modification.

1. Run `planemo test pear.cwl` to verify the tool does not function without dependencies defined.
2. Use `--conda_requirements` flag with `planemo lint` to verify it does indeed lack requirements.
3. Use `planemo conda_search` or the Anaconda website to search for the correct package and version in a best practice channel.
4. Update `pear.cwl` with the correct `SoftwareRequirement` hints.
5. Re-run the `lint` command from above to verify the tool now has the correct dependency definition.
6. Re-run the `test` command from above to verify the tool test now works properly.

Building New Conda Packages

Frequently packages your tool will require are not found in Bioconda or conda-forge yet. In these cases, it is likely best to contribute your package to one of these projects. Unless the tool is exceedingly general Bioconda is usually the correct starting point.

Note: Many things that are not strictly or even remotely "bio" have been accepted into Bioconda - including tools for image analysis, natural language processing, and cheminformatics.

To get quickly learn to write Conda recipes for typical Galaxy tools, please read the following pieces of external documentation.

- Contributing to Bioconda in particular focusing on
  - One time setup
  - Contributing a recipe (through "Write a Recipe")
- Building conda packages in particular
  - Building conda packages with conda skeleton (the best approach for common scripting languages such as R and Python)
  - Building conda packages from scratch
  - Building conda packages for general code projects
  - Using conda build
- Then return to the Bioconda documentation and read
  - The rest of "Contributing a recipe" continuing from Testing locally
  - And finally Guidelines for bioconda recipes

These guidelines in particular can be skimmed depending on your recipe type, for instance that document provides specific advice for:

- Python
- R (CRAN)
- R (Bioconductor)
- Perl
- C/C++

To go a little deeper, you may want to read:

- Specification for meta.yaml
- Environment variables
- Custom channels

And finally to debug problems the Bioconda troubleshooting documentation may prove useful.

Exercise - Build a Recipe

If you have just completed the exercise above - this exercise can be found in parent folder. Get there with `cd ../exercise_2`. If not, the exercise can be downloaded with

```
$ planemo project_init --template conda_exercises_cwl conda_exercises
$ cd conda_exercises/exercise_2
$ ls
fleeqtk_seq.cwl      fleeqtk_seq_tests.yml         test-data
```

This is the skeleton of a tool wrapping the parody bioinformatics software package fleeqtk. fleeqtk is a fork of the project seqtk that many Planemo tutorials are built around and the example tool should hopefully be fairly familiar. fleeqtk version 1.3 can be downloaded from here and built using `make`. The result of `make` includes a single executable `fleeqtk`.

1. Clone and branch Bioconda.
2. Build a recipe for fleeqtk version 1.3. You may wish to start from scratch (`conda skeleton` is not available for C programs like fleeqtk), or copy the recipe of seqtk and modify it for fleeqtk.
3. Use `conda build` or Bioconda tooling to build the recipe.
4. Run `planemo test --conda_use_local fleeqtk_seq.cwl` to verify the resulting package works as expected.

Congratulations on writing a Conda recipe and building a package! Upon succesfully building and testing such a Bioconda package, you would normally push your branch to Github and open a pull request. This step is skipped here as to not pollute Bioconda with unneeded software packages.

6.3.2 Dependencies and Containers

Note: This section is a continuation of *Dependencies and Conda*, please review that section for background information on resolving Software Requirements with Conda.

Common Workflow Language tools can be annotated with arbitrary Docker requirements, see the CWL User Guide for a discussion about how to do this in general.

This document will discuss some techniques to find containers automatically from the `SoftwareRequirement` annotations when using Planemo, cwltool, or Toil. You will ultimately want to explicitly annotate your tools with the containers we describe here so that other CWL implementations will be able to find containers for your tool, but there are real advantages to using these containers instead of ad-hoc things you may build with a `Dockerfile`.

- They provide superior reproducibility because the same binary Conda packages will automatically be used for both bare metal dependencies and inside containers.
- They are constructed automatically from existing Conda packages so you as a tool developer won't need to write `Dockerfile` s or register projects on Docker Hub.
- They are produced using mulled which produce very small containers that make deployment easier regardless of the CWL implementation you are using.
- Annotating Software Requirements reduces the opaqueness of the Docker process. With this method it is entirely traceable how the container was constructed from what sources were fetched, which exact build of every dependency was used, to how packages in the container were built. Beyond that metadata about the packages can be fetched from Bioconda (e.g. this).

Read more about this reproducibility stack in our preprint Practical computational reproducibility in the life sciences.

BioContainers

Note: This section is a continuation of *Dependencies and Conda*, please review that section for background information on resolving Software Requirements with Conda.

Finding BioContainers

If a tool contains Software Requirements in best practice Conda channels, a BioContainers-style container can be found or built for it.

As reminder, `planemo lint --conda_requirements <tool.cwl>` can be used to check if a tool contains only best-practice `requirement` tags. The `lint` command can also be fed the `--biocontainers` flag to check if a BioContainers container has been registered that is compatible with that tool.

This last linter indicates that indeed a container has been registered that is compatible with this tool – `quay.io/biocontainers/seqtk:1.2--1`. We didn't do any extra work to build this container for this tool, all Bioconda recipes are packaged into containers and registered on quay.io as part of the BioContainers project.

This tool can be tested using `planemo test` in its BioContainer Docker container using the flag `--biocontainers` as shown below.

The Conda exercises project template has an example tool (`exercise3`) that we can use to demonstrate `--biocontainers`. If you are continuing from the Conda tutorial, simply move to `../exercise3` otherwise using `planemo project_init` to grab the exercise as show below.

```
$ planemo project_init --template conda_exercises_cwl conda_exercises
$ cd conda_exercises/exercise3
$ planemo lint --biocontainers seqtk_seq.cwl
Linting tool /home/planemo/conda_exercises_cwl/exercise_3/seqtk_seq.cwl
Applying linter general... CHECK
.. CHECK: Tool defines a version [0.0.1].
.. CHECK: Tool defines a name [Convert to FASTA (seqtk)].
.. CHECK: Tool defines an id [seqtk_seq].
.. CHECK: Tool specifies profile version [16.04].
Applying linter cwl_validation... CHECK
.. INFO: CWL appears to be valid.
Applying linter docker_image... WARNING
.. WARNING: Tool does not specify a DockerPull source.
Applying linter new_draft... CHECK
.. INFO: Modern CWL version [v1.0]
Applying linter biocontainer_registered... CHECK
.. INFO: BioContainer best-practice container found [quay.io/biocontainers/seqtk:1.2--1].
Failed linting
```

```
$ planemo test --biocontainers seqtk_seq.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20180508202931
cwltool INFO: Resolved '/Users/john/workspace/planemo/project_templates/conda_exercises_
↪cwl/exercise_3/seqtk_seq.cwl' to 'file:///Users/john/workspace/planemo/project_
↪templates/conda_exercises_cwl/exercise_3/seqtk_seq.cwl'
galaxy.tools.deps.containers INFO: Checking with container resolver␣
↪[ExplicitContainerResolver[]] found description [None]
galaxy.tools.deps.containers INFO: Checking with container resolver␣
↪[CachedMulledDockerContainerResolver[namespace=biocontainers]] found description [None]
galaxy.tools.deps.containers INFO: Checking with container resolver␣
↪[MulledDockerContainerResolver[namespace=biocontainers]] found description␣
↪[ContainerDescription[identifier=quay.io/biocontainers/seqtk:1.2--1,type=docker]]
cwltool INFO: [job seqtk_seq.cwl] /private/tmp/docker_tmpMEipaU$ docker \
    run \
    -i \
    --volume=/private/tmp/docker_tmpMEipaU:/private/tmp/docker_tmpMEipaU:rw \
    --volume=/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpxkm9dp:/tmp:rw \
    --volume=/Users/john/workspace/planemo/project_templates/conda_exercises_cwl/
↪exercise_3/test-data/2.fastq:/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/
↪tmpjAVM_1/stgddf6fc2a-dd13-4322-9b88-68571a1697dd/2.fastq:ro \
    --workdir=/private/tmp/docker_tmpMEipaU \
    --read-only=true \
    --log-driver=none \
    --user=502:20 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/private/tmp/docker_tmpMEipaU \
    quay.io/biocontainers/seqtk:1.2--1 \
    seqtk \
    seq \
    -a \
    /private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpjAVM_1/stgddf6fc2a-dd13-
```

(continues on next page)

(continued from previous page)

```
↪4322-9b88-68571a1697dd/2.fastq > /private/tmp/docker_tmpMEipaU/out
cwltool INFO: [job seqtk_seq.cwl] completed success
cwltool INFO: Final process status is success
All 1 test(s) executed passed.
seqtk_seq_0: passed
```

Exercise - Leveraging Bioconda

1. Try the above command without the `--biocontainers` argument. Verify the tool does not run in a container by default.
2. Add a DockerRequirement based on the the lint output above to annotate this tool with a Biocontainers Docker container and rerun test to verify the tool works now.

Building BioContainers

In this seqtk example above the relevant BioContainer already existed on quay.io, this won't always be the case. For tools that contain multiple Software Requirements tags an existing container likely won't exist. The mulled toolkit (distributed with planemo or available standalone) can be used to build containers for such tools. For such tools, if cwltool or Toil is configured to use BioContainers it will attempt to build these containers on the fly by default (though this behavior can be disabled).

You can try it directly using the `mull` command in Planemo. The `conda_testing` Planemo project template has a toy example tool with two requirements for demonstrating this - bwa_and_samtools.cwl.

```
$ planemo project_init --template=conda_testing_cwl conda_testing
$ cd conda_testing/
$ planemo mull bwa_and_samtools.cwl
/Users/john/.planemo/involucro -v=3 -f /Users/john/workspace/planemo/.venv/lib/python2.7/
↪site-packages/galaxy_lib-17.9.0-py2.7.egg/galaxy/tools/deps/mulled/invfile.lua -set␣
↪CHANNELS='iuc,bioconda,r,defaults,conda-forge' -set TEST='true' -set TARGETS=
↪'samtools=1.3.1,bwa=0.7.15' -set REPO='quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820' -
↪set BINDS='build/dist:/usr/local/' -set PREINSTALL='conda install --quiet --yes␣
↪conda=4.3' build
/Users/john/.planemo/involucro -v=3 -f /Users/john/workspace/planemo/.venv/lib/python2.7/
↪site-packages/galaxy_lib-17.9.0-py2.7.egg/galaxy/tools/deps/mulled/invfile.lua -set␣
↪CHANNELS='iuc,bioconda,r,defaults,conda-forge' -set TEST='true' -set TARGETS=
↪'samtools=1.3.1,bwa=0.7.15' -set REPO='quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820' -
↪set BINDS='build/dist:/usr/local/' -set PREINSTALL='conda install --quiet --yes␣
↪conda=4.3' build
[Jun 19 11:28:35] DEBU Run file [/Users/john/workspace/planemo/.venv/lib/python2.7/site-
↪packages/galaxy_lib-17.9.0-py2.7.egg/galaxy/tools/deps/mulled/invfile.lua]
[Jun 19 11:28:35] STEP Run image [continuumio/miniconda:latest] with command [[rm -rf /
↪data/dist]]
[Jun 19 11:28:35] DEBU Creating container [step-730a02d79e]
[Jun 19 11:28:35] DEBU Created container [5e4b5f83c455 step-730a02d79e], starting it
[Jun 19 11:28:35] DEBU Container [5e4b5f83c455 step-730a02d79e] started, waiting for␣
↪completion
[Jun 19 11:28:36] DEBU Container [5e4b5f83c455 step-730a02d79e] completed with exit code␣
```

(continues on next page)

(continued from previous page)

```
↪[0] as expected
[Jun 19 11:28:36] DEBU Container [5e4b5f83c455 step-730a02d79e] removed
[Jun 19 11:28:36] STEP Run image [continuumio/miniconda:latest] with command [[/bin/sh -
↪c conda install --quiet --yes conda=4.3 && conda install  -c iuc -c bioconda -c r -c␣
↪defaults -c conda-forge  samtools=1.3.1 bwa=0.7.15 -p /usr/local --copy --yes --quiet]]
[Jun 19 11:28:36] DEBU Creating container [step-e95bf001c8]
[Jun 19 11:28:36] DEBU Created container [72b9ca0e56f8 step-e95bf001c8], starting it
[Jun 19 11:28:37] DEBU Container [72b9ca0e56f8 step-e95bf001c8] started, waiting for␣
↪completion
[Jun 19 11:28:46] SOUT Fetching package metadata .........
[Jun 19 11:28:47] SOUT Solving package specifications: .
[Jun 19 11:28:50] SOUT
[Jun 19 11:28:50] SOUT Package plan for installation in environment /opt/conda:
[Jun 19 11:28:50] SOUT
[Jun 19 11:28:50] SOUT The following packages will be UPDATED:
[Jun 19 11:28:50] SOUT
[Jun 19 11:28:50] SOUT conda: 4.3.11-py27_0 --> 4.3.22-py27_0
[Jun 19 11:28:50] SOUT
[Jun 19 11:29:04] SOUT Fetching package metadata ................
[Jun 19 11:29:06] SOUT Solving package specifications: .
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:56] SOUT Package plan for installation in environment /usr/local:
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:56] SOUT The following NEW packages will be INSTALLED:
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:56] SOUT bwa:        0.7.15-1      bioconda
[Jun 19 11:29:56] SOUT curl:       7.52.1-0
[Jun 19 11:29:56] SOUT libgcc:     5.2.0-0
[Jun 19 11:29:56] SOUT openssl:    1.0.2l-0
[Jun 19 11:29:56] SOUT pip:        9.0.1-py27_1
[Jun 19 11:29:56] SOUT python:     2.7.13-0
[Jun 19 11:29:56] SOUT readline:   6.2-2
[Jun 19 11:29:56] SOUT samtools:   1.3.1-5       bioconda
[Jun 19 11:29:56] SOUT setuptools: 27.2.0-py27_0
[Jun 19 11:29:56] SOUT sqlite:     3.13.0-0
[Jun 19 11:29:56] SOUT tk:         8.5.18-0
[Jun 19 11:29:56] SOUT wheel:      0.29.0-py27_0
[Jun 19 11:29:56] SOUT zlib:       1.2.8-3
[Jun 19 11:29:56] SOUT
[Jun 19 11:29:57] DEBU Container [72b9ca0e56f8 step-e95bf001c8] completed with exit code␣
↪[0] as expected
[Jun 19 11:29:57] DEBU Container [72b9ca0e56f8 step-e95bf001c8] removed
[Jun 19 11:29:57] STEP Wrap [build/dist] as [quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0]
[Jun 19 11:29:57] DEBU Creating container [step-6f1c176372]
[Jun 19 11:29:58] DEBU Packing succeeded
```

As the output indicates, this command built the container named `quay.io/biocontainers/mulled-v2-fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0`. This is the same namespace / URL that would be used if or when published by the BioContainers project.

Note: The first part of this `mulled-v2` hash is a hash of the package names that went into it, the second the packages used and build number. Check out the Multi-package Containers web application to explore best practice channels and build such hashes.

We can see this new container when running the Docker command `images` and explore the new container interactively with `docker run`.

```
$ docker images
REPOSITORY                                                                    TAG          ␣
↪                                IMAGE ID            CREATED             SIZE
quay.io/biocontainers/mulled-v2-fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40  ␣
↪03dc1d2818d9de56938078b8b78b82d967c1f820-0   a740fe1e6a9e        16 hours ago         ␣
↪104 MB
quay.io/biocontainers/seqtk                                                   1.2--0       ␣
↪                                10bc359ebd30        2 days ago          7.34 MB
continuumio/miniconda                                                         latest       ␣
↪                                6965a4889098        3 weeks ago         437 MB
bgruening/busybox-bash                                                        0.1          ␣
↪                                3d974f51245c        9 months ago        6.73 MB
$ docker run -i -t quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0 /
↪bin/bash
bash-4.2# which samtools
/usr/local/bin/samtools
bash-4.2# which bwa
/usr/local/bin/bwa
```

As before, we can test running the tool inside its container in cwltool using the `--biocontainers` flag.

```
$ planemo test --biocontainers bwa_and_samtools.cwl
Enable beta testing mode for testing.
cwltool INFO: /Users/john/workspace/planemo/.venv/bin/planemo 1.0.20180508202931
cwltool INFO: Resolved '/Users/john/workspace/planemo/project_templates/conda_testing_
↪cwl/bwa_and_samtools.cwl' to 'file:///Users/john/workspace/planemo/project_templates/
↪conda_testing_cwl/bwa_and_samtools.cwl'
galaxy.tools.deps.containers INFO: Checking with container resolver␣
↪[ExplicitContainerResolver[]] found description [None]
galaxy.tools.deps.containers INFO: Checking with container resolver␣
↪[CachedMulledDockerContainerResolver[namespace=biocontainers]] found description␣
↪[ContainerDescription[identifier=quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0,
↪type=docker]]
cwltool INFO: [job bwa_and_samtools.cwl] /private/tmp/docker_tmpYJnm04$ docker \
    run \
    -i \
    --volume=/private/tmp/docker_tmpYJnm04:/private/tmp/docker_tmpYJnm04:rw \
    --volume=/private/var/folders/78/zxz5mz4d0jn53xf0l06j7ppc0000gp/T/tmpVI06me:/tmp:rw \
    --workdir=/private/tmp/docker_tmpYJnm04 \
    --read-only=true \
    --user=502:20 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/private/tmp/docker_tmpYJnm04 \
    quay.io/biocontainers/mulled-v2-
```

(continues on next page)

(continued from previous page)

```
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0 \
    sh \
    -c \
    'bwa > bwa_help.txt 2>&1; samtools > samtools_help.txt 2>&1'
cwltool INFO: [job bwa_and_samtools.cwl] completed success
cwltool INFO: Final process status is success
All 1 test(s) executed passed.
bwa_and_samtools_0: passed
```

In particular take note of the line:

```
2017-03-01 10:20:59,142 INFO  [galaxy.tools.deps.containers] Checking with container␣
↪resolver [CachedMulledDockerContainerResolver[namespace=biocontainers]] found␣
↪description [ContainerDescription[identifier=quay.io/biocontainers/mulled-v2-
↪fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1d2818d9de56938078b8b78b82d967c1f820-0,
↪type=docker]]
```

Here we can see the container ID (`quay.io/biocontainers/mulled-v2-fe8faa35dbf6dc65a0f7f5d4ea12e31a79f73e40:03dc1` from earlier has been cached on our Docker host is picked up by cwltool. This is used to run the simple tool tests and indeed they pass.

In our initial seqtk example, the container resolver that matched was of type `MulledDockerContainerResolver` indicating that the Docker image would be downloaded from the BioContainers repository and this time the resolve that matched was of type `CachedMulledDockerContainerResolver` meaning that cwltool would just use the locally cached version from the Docker host (i.e. the one we built with `planemo mull` above).

Note: Planemo doesn't yet expose options that make it possible to build mulled containers for local packages that have yet to be published to anaconda.org but the mulled toolkit allows this. See mulled documentation for more information. However, once a container for a local package is built with `mulled-build-tool` the `--biocontainers` command should work to test it.

Publishing BioContainers

Building unpublished BioContainers on the fly is great for testing but for production use and to increase reproducibility such containers should ideally be published as well.

BioContainers maintains a registry of package combinations to be published using these long mulled hashes. This registry is represented as a Github repository named multi-package-containers. The Planemo command `container_register` will inspect a tool and open a Github pull request to add the tool's combination of packages to the registry. Once merged, this pull request will result in the corresponding BioContainers image to be published (with the correct mulled has as its name) - these can be subsequently be picked up by Galaxy.

Various Github related settings need to be configured in order for Planemo to be able to open pull requests on your behalf as part of the `container_register` command. To simplify all of this - the Planemo community maintains a list of Github repositories containing Galaxy and/or CWL tools that are scanned daily by Travis. For each such repository, the Travis job will run `container_register` across the repository on all tools resulting in new registry pull requests for all new combinations of tools. This list is maintained in a script named `monitor.sh` in the planemo-monitor repository. The easiest way to ensure new containers are built for your tools is simply to open open a pull request to add your tool repositories to this list.

CHAPTER

SEVEN

PUBLISHING TO THE TOOL SHED

The Galaxy Tool Shed (referred to colloquially in Planemo as the “shed”) can store Galaxy tools, dependency definitions, and workflows among other Galaxy artifacts. This guide will assume some basic familiarity with the shed - please review the Tool Shed Wiki for an introduction.

7.1 Configuring a Shed Account

Before getting started, it is a good idea to have accounts on the Galaxy test and (optionally) main Tool Sheds. Also, if you haven’t initialized a global Planemo configuration file (`~/.planemo.yml`) this can be done with.

```
planemo config_init
```

This will populate a template `~/.planemo.yml` file and provide locations to fill in shed credentials for the test and main Tool Sheds. For each shed, fill in either an API `key` or an `email` and `password`. Also specify the `shed_username` created when registering shed accounts. All these options can be specified and/or overridden on each planemo command invocation - but that becomes tedious quickly.

7.2 Creating a Repository

Planemo can be used to used to publish “repositories” to the Tool Shed. A single GitHub repository or locally managed directory of tools may correspond to any number of Tool Shed repositories. Planemo maps files to Tool Shed repositories using a special file called `.shed.yml`.

From a directory containing tools, a package definition. etc… the `shed_init` command can be used to bootstrap a new `.shed.yml` file.

```
planemo shed_init --name=<name>
                  --owner=<shed_username>
                  --description=<short description>
                  [--remote_repository_url=<URL to .shed.yml on github>]
                  [--homepage_url=<Homepage for tool.>]
                  [--long_description=<long description>]
                  [--category=<category name>]*
```

There is not a lot of magic happening here, this file could easily be created directly with a text editor - but the command has a `--help` to assist you and does some very basic validation.

Note: Periods and hyphens are disallowed in repository names, it is recommended replacing periods in the version number with underscores.

The following naming conventions are recommended and in some cases Planemo will determine the repository type based on adherence to these conventions (for packages and suites specifically).

Repository Type	Recommended Name	Examples
Data Managers	`data_manager_$name`	`data_manager_bowtie2`
Packages	`package_$name_$version`	`package_aragorn_1_2_36`
Tool Suites	`suite_$name`	`suite_samtools`
Tools	`$name`	`stringtie`, `bedtools`

More information on `.shed.yml` can be found as part of the IUC best practice documentation.

After reviewing `.shed.yml`, this configuration file and relevant shed artifacts can be quickly linted using the following command.

```
planemo shed_lint --tools
```

Once the details the `.shed.yml` are set and it is time to create the remote repository and upload artifacts to it - the following two commands can be used - the first only needs to be run once and creates the repository based the metadata in `.shed.yml` and the second uploads your actual artifacts to it.

```
planemo shed_create --shed_target testtoolshed
```

7.3 Updating a Repository

Ensure the Galaxy Test Tool Shed is enabled in Galaxy's `config/tool_sheds_conf.xml` file and install and test the new repository.

If modifications are required these can be reviewed using the `shed_diff` command.

```
planemo shed_diff --shed_target testtoolshed
```

Note: If you look at tools-iuc you will see it is common practice to leave details such as shed target and changeset_revision from `tool_dependencies.xml` and `repository_dependencies.xml` files. These are required by the Tool Shed but it will populate them on upload and leaving them blank allows uploading the same artifacts to the Test and Main sheds. The upshot of this is however is that `shed_diff` will always print diffs on these artifacts.

Modified artifacts can be uploaded using the following command.

```
planemo shed_update --check_diff --shed_target testtoolshed
```

The `--check_diff` option here will ensure there are significant differnces before uploading new contents to the tool shed.

Once tools and required dependency files have been published to the tool shed, the actual shed dependencies can be automatically and installed and tool tests ran using the command:

```
planemo shed_test --shed_target testtoolshed
```

Once your artifacts are ready for publication to the main Tool Shed, the following commands to create a repository there and populate it with your repository contents.

```
planemo shed_create
```

7.4 Advanced Usage

The above usage is relatively straight forward - it will map the current directory to a single repository in the Tool Shed.

See Pull Request 143 and linked examples for details on more advanced options such as mapping each tool to its own repository automatically (a best practice) or building enitrely custom repository definitions manually.

CHAPTER

EIGHT

TEST FORMAT

Planemo has traditionally been used to test Galaxy tools.

```
$ planemo test galaxy_tool.xml
```

This starts a Galaxy instance, runs the tests described in the XML file, prints a nice summary of the test results (pass or fail for each test) in the console and creates an HTML report in the current directory. Additional bells and whistles include the ability to generate XUnit reports, publish test results and get embedded Markdown to link to them for PRs, and test remote artifacts in Git repositories.

Much of this same functionality is now also available for Galaxy Workflows as well as Common Workflow Language (CWL) tools and workflows. The rest of this page describes this testing format and testing options for these artifacts - for information about testing Galaxy tools specifically using the embedded tool XML tests see Test-Driven Development of Galaxy tools tutorial.

Unlike the traditional Galaxy tool approach, these newer types of artifacts should define tests in files located next artifact. For instance, if `planemo test` is called on a Galaxy workflow called `ref-rnaseq.ga` tests should be defined in `ref-rnaseq-tests.yml` or `ref-rnaseq-tests.yaml`. If instead it is called on a CWL tool called `seqtk_seq.cwl`, tests can be defined in `seqtk_seq_tests.yml` for instance.

Below are two examples of such YAML files - the first for a CWL tool and the second for Galaxy workflow. Note the same testing file format is used for both kinds of artifacts.

```
- doc: simple usage test
  job: pear_job.yml
  outputs:
    assembled_pairs:
      path: test-data/pear_assembled_results1.fastq
    unassembled_forward_reads:
      path: test-data/pear_unassembled_forward_results1.fastq
```

```
- doc: Test sample data for Microbial variant calling workflow
  job:
    mutant_R1:
      class: File
      path: mutant_R1.fastq
    mutant_R2:
      class: File
      path: mutant_R2.fastq
    wildtype.fna:
      class: File
      location: https://zenodo.org/record/582600/files/wildtype.fna
    wildtype.gbk:
```

(continues on next page)

(continued from previous page)

```
      class: File
      location: https://zenodo.org/record/582600/files/wildtype.gbk
    wildtype.gff:
      class: File
      location: https://zenodo.org/record/582600/files/wildtype.gff
  outputs:
    jbrowse_html:
      asserts:
        has_text:
          text: "JBrowseDefaultMainPage"
    snippy_fasta:
      asserts:
        has_line:
          line: '>Wildtype Staphylococcus aureus strain WT.'
    snippy_tabular:
      asserts:
        has_n_columns:
          n: 2
```

The above examples illustrate that each test file is broken into a list of test cases. Each test case should have a `doc` describing the test, a `job` description the describes the inputs for an execution of the target artifact, and an `outputs` mapping that describes assertions about outputs to test.

8.1 job

The `job` object can be a mapping embedded right in the test file or a reference to a an external "job" input file. The job input file is a proper CWL job document - which is fairly straight forward as demonstrated in the above examples. Planemo adapts the CWL job document to Galaxy workflows and tools - using input names for Galaxy tools and input node labels for workflows.

Input files can be specified using either `path` attributes (which should generally be file paths relative to the artifact and test directory) or `location` (which should be a URI). The examples above demonstrate using both paths relative to the tool file and test data published to Zenodo.

Embedded job objects result in cleaner test suites that are simpler to read. One advantage of instead using external job input files is that the job object can be reused to invoke the runnable artifact outside the context of testing with `planemo run`.

Note: These job objects can be run directly with `planemo run`.

```
$ planemo run --engine=<engine_type> [ENGINE_OPTIONS] [ARTIFACT_PATH] [JOB_PATH]
```

This should be familar to CWL developers - and indeed if `--engine=cwltool` this works as a formal CWL runner. Planemo provides a uniform interface to Galaxy for Galaxy workflows and tools though using the same CLI invocation if `--engine=galaxy` (for a Planemo managed Galaxy instance), `--engine=docker_galaxy` (for a Docker instance of Galaxy launched by Planemo), or `--engine=external_galaxy` (for a running remote Galaxy instance).

Certain Galaxy objects don't map cleanly to CWL job objects so Planemo attempts to extend the format with new constructs for running and testing Galaxy objects - such as describing collections and composite inputs.

8.1.1 Galaxy Collection Inputs

The following example demonstrates two ways to create input lists for Galaxy tests.

```yaml
- doc: Test Explicit Collection Creation.
  job:
    input1:
      class: Collection
      collection_type: list
      elements:
        - identifier: el1
          class: File
          path: hello.txt
  outputs:
    wf_output_1:
      checksum: "sha1$a0b65939670bc2c010f4d5d6a0b3e4e4590fb92b"
- doc: Test CWL-style list inputs.
  job:
    input1:
      - class: File
        path: hello.txt
  outputs:
    wf_output_1:
      checksum: "sha1$a0b65939670bc2c010f4d5d6a0b3e4e4590fb92b"
```

Simply specifying files in YAML lists in the input job (like vanilla CWL job descriptions) will result in a simple Galaxy list. This is simple but the downside is you have no control of the list identifiers - which are often important in Galaxy workflows. When more control is desired, you may describe an explicit Galaxy collection with an input object of `class: Collection`. This variant (also shown in the above example) allows creating collections of type other than `list` and allows specifying element identifiers with the `identifier` declaration under the list of collection `elements`.

The explicit Galaxy collection creation syntax also makes describing nested collections such as lists of pairs very natural. The following example is used in Planemo's test suite to illustrate this:

```yaml
- doc: Test Explicit Collection Creation.
  job:
    input1:
      class: Collection
      collection_type: 'list:paired'
      elements:
        - class: Collection
          type: paired
          identifier: el1
          elements:
          - identifier: forward
            class: File
            path: hello.txt
          - identifier: reverse
            class: File
            path: hello.txt
  outputs:
    wf_output_1:
      checksum: "sha1$7bd92c6cd84285e4fc7215d506bbabfe328acb8f"
```

8.1.2 Galaxy Composite Inputs

The syntax for specifying composite inputs is a little more basic still and simply must be specified as a list of local files (mirroring Galaxy Tool XML test syntax). While `class` is assumed to be `File` and URIs aren't yet tested.

```
- doc: Test Composite Inputs
  job:
    input1:
      class: File
      filetype: imzml
      composite_data:
        - path: Example_Continuous.imzML
        - path: Example_Continuous.ibd
  outputs:
    wf_output_1:
      checksum: "sha1$0d2ad51f69d7b5df0f4d2b2a47b17478f2fca509"
```

8.1.3 Galaxy Tags

Requires Galaxy 20.09 or newer.

Tags and group tags play important roles in many Galaxy workflows. These can be tested by simply add a list of `tags:` to the YAML corresponding to the dataset in the collection. The following example demonstrates this:

```
- doc: Test using tags.
  job:
    input_c:
      class: Collection
      collection_type: list
      elements:
        - identifier: el1
          class: File
          path: hello.txt
          tags: ['group:which:moo']
        - identifier: el2
          class: File
          path: not_hello.txt
          tags: ['group:which:cow']
  outputs:
    wf_output_1:
      checksum: "sha1$a0b65939670bc2c010f4d5d6a0b3e4e4590fb92b"
```

8.2 outputs

Galaxy tools and CWL artifacts have obvious output names that much match the mapping in this block on test file. Galaxy workflows require explicit output labels to be used with tests, but the important outputs in your workflows should be labeled anyway to work with Galaxy subworkflows and more cleanly with API calls.

If an output is known, fixed, and small it makes a lot of sense to just include a copy of the output next to your test and set `file: relative/path/to/output` in your output definition block as show in the first example above. For completely reproducible processes this is a great guarentee that results are fixed over time, across CWL engines and engine versions. If the results are fixed but large - it may make sense to just describe the outputs by a SHA1 checksum.

```yaml
- doc: Simple concat workflow test
  job: wf1.gxwf-job.yml
  outputs:
    wf_output_1:
      checksum: "sha1$a0b65939670bc2c010f4d5d6a0b3e4e4590fb92b"
```

One advantage of included an exact file instead of a checksum is that Planemo can produce very nice line by line diffs for incorrect test results by comparing an expected output to an actual output.

There are reasons one may not be able to write such exact test assertions about outputs however, perhaps date or time information is incorporated into the result, unseeded random numbers are used, small numeric differences occur across runtimes of interest, etc.. For these cases, a variety of other assertions can be executed against the execution results to verify outputs. The types and implementation of these test assertions match those available to Galaxy tool outputs in XML but have equivalent YAML formulations that should be used in test descriptions.

Even if one can write exact tests, a really useful technique is to write sanity checks on outputs as one builds up workflows that may be changing rapidly and developing complex tools or worklflows via a Test-Driven Development cycle using Planemo. *Tests shouldn't just be an extra step you have to do after development is done, they should guide development as well.*

The workflow example all the way above demonstrates some assertions one can make about the contents of files. The full list of assertions available is only documented for the Galaxy XML format but it is straightforward to adapt to the YAML format above - check out the Galaxy XSD for more information.

Some examples of inexact file comparisons derived from an artificial test case in the Planemo test suite is shown below, these are more options available for checking outputs that may change in small ways over time.

```yaml
- doc: test_sha1_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      checksum: sha1$2ef7bde608ce5404e97d5f042f95f89f1c232871
- doc: test_sha1_fail
  job: cat_tool_job.json
  outputs:
    output_file:
      checksum: sha1$2ef7bde608ce5404e97d5f042f95f89f1c232872
- doc: test_compare_direct_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      file: hello.txt
- doc: test_compare_sim_size_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello.txt
      compare: sim_size
      delta: 5
- doc: test_compare_sim_size_fail
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello.txt
      compare: sim_size
```

(continues on next page)

(continued from previous page)

```
      delta: 3
- doc: test_compare_re_match_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      file: hello_regex.txt
      compare: re_match
- doc: test_compare_re_match_fail
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello_regex.txt
      compare: re_match
- doc: test_compare_re_match_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      file: hello_regex.txt
      compare: re_match_multiline
- doc: test_compare_re_match_fail
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello_regex.txt
      compare: re_match_multiline
- doc: test_contains_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      file: hello_truncated.txt
      compare: contains
- doc: test_contains_fail
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello.txt
      compare: contains
- doc: test_diff_pass
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello.txt
      compare: diff
      lines_diff: 2
- doc: test_diff_fail
  job: cat_tool_job.json
  outputs:
    output_file:
      file: not_hello.txt
      compare: diff
      lines_diff: 1
```

8.3 Engines for Testing

Below are descriptions of various testing engines that can be used with Planemo (both with the test command and the run command) as well as some command-line options of particular interest for testing. The first two types `cwltool` and `toil` can be used to test CWL artifacts (tools and workflows). The remaining engine types are variations on engines that target Galaxy and are useful for testing workflows (and tools with newer style tests or job documents).

8.3.1 cwltool

```
$ planemo test --engine cwltool [--no-container] [--biocontainers]
```

This is the most straight forward engine, it can be used to test CWL tools and workflows using the CWL reference implementation cwltool (bundled as a dependency of Planemo). Use the `--no-container` option to disable Docker and use Conda resolution of `SoftwareRequirement``s or applications on the ``PATH`. Use the `--biocontainers` flag to use BioContainers for tools without explicit `DockerRequirement` hints.

8.3.2 toil

```
$ planemo test --engine toil [--no-container] [--biocontainers]
```

This engine largely mirrors the `cwltool` engine but runs CWL artifacts using Toil. Toil is an optional dependency of Planemo so you will likely have to install it in Planemo's environment using `pip install toil`.

8.3.3 galaxy

```
$ planemo test [--docker] [--biocontainers] [--profile <profile>] [--galaxy_root <path>]␣
↪[--extra_tools <path>]
```

This is the default engine type, but can be made explicit `--engine galaxy`. With this engine Planemo will start a Galaxy instance and test against it.

Planemo will automatically detect and load "stock" Galaxy tools used by workflows and install any Tool Shed tools contained in the workflow, if other non-Tool Shed tools are required for a workflow they can be loaded using `--extra_tools`.

Set `--galaxy_root` to target an externally cloned Galaxy directory or use `--galaxy_branch` to target a particular branch of Galaxy other than the latest stable.

Use the `--biocontainers` flag to enable Docker and use BioContainers for tools or use `--docker` to use Docker but limited to tools configured with `container` tags.

By default Galaxy when configured by Planemo will attempt to run with an sqlite database. This configuration is quite buggy and should not be used to test workflows. The `--profile` option can be used to target a pre-configured Postgres database created with `planemo profile_create` and use it for testing. In addition to making Galaxy more robust this should speed up testing after the initial setup of the database.

```
planemo profile_create --database_type [postgres|docker_postgres] my_cool_name
planemo test --profile my_cool_name
```

If `--database_type` is specified as `docker_postgres`, Planemo will attempt to startup a postgres server in a Docker container automatically for testing. If instead `postgres` is specified Planemo will attempt to interact with Postgres using `psql` (assumed to be on the `PATH`). For a description on more Postgres connection options check out the documentation for the `database_create` command that has similar options.

Profiles may also really help testing local setups by saving previously installed shed repository installations and Conda environments.

8.3.4 `docker_galaxy`

```
$ planemo test --engine docker_galaxy [--extra_tools <path>] [--docker_extra_volume
↪<path>] [--docker_galaxy_image <image>]
```

With this engine Planemo will start a Docker container to run tests against it. See the docker-galaxy-stable project spearheaded by Björn Grüning for more information on Docker-ized Galaxy execution. The exact container image to use can be controlled using the `--docker_galaxy_image` option.

Planemo will automatically detect and load "stock" Galaxy tools used by workflows and install any Tool Shed tools contained in the workflow, if other non-Tool Shed tools are required - they can be loaded using `--extra_tools`.

At the time of this writing, there is a bug in Planemo that requires using the `--docker_extra_volume` option to mount test data into the testing container.

8.3.5 `external_galaxy`

> $ planemo test –engine external_galaxy –galaxy_admin_key <admin_key> –galaxy_user_key <user_key> [–no_shed_install] [–polling_backoff <integer>] –galaxy_url <url>

This is primarily useful for testing workflows against already running Galaxy instances. An admin or bootstrap API key should be supplied to install missing tool repositories for the workflow and a user API key should be supplied to run the workflow using. If you wish to skip tool shed repository installation (this requires all the tools be present already), use the `--no_shed_install` option. If you want to reduce the load on the target Galaxy while checking for the status changes use the `--polling_backoff <integer>` option where integer is the incremental increase in seconds for every request.

To run tool tests against a running Galaxy, `galaxy-tool-test` is a script that gets installed with galaxy-tool-util and so may very well already be on your `PATH`. Check out the options available with that using `galaxy-tool-test --help`. If you're interested in running all the tool tests corresponding to a workflow on a running server, check out the galaxy-workflow-tool-tests project that is a wrapper around galaxy-tool-test that has all the same options but that filters to the tool tests to just run those from a specific workflow.

This engine can also be used to test workflows already available in the running Galaxy instance. While you don't need to download and synchronize the target workflow on your local filesystem, you do need to provide a path to find the test definition and test data paths.

An example of doing this is included in Planemo's test data. The workflow test definition `wf11-remote.gxwf-test.yml` exists but no corresponding workflow file `wf11-remote.gxwf.yml` exists. The workflow is assumed to already exist in some Galaxy server. For instance, it might exist somewhere with id `99113b2b119318e1`. Then `planemo test` could be run with `gxid://workflows/99113b2b119318e1?runnable_path=/path/to/wf11-remote.gxwf.yml` as the last argument to test this workflow with that test data. Note this path `/path/to/wf11-remote.gxwf.yml` doesn't need to exist, but it is used to find `wf11-remote.gxwf-test.yml`.

8.4 Galaxy Testing Template

The following a script that can be used with continuous integration (CI) services such Travis to test Galaxy workflows in a Github repository. This shell script can be configured via various environment variables and shows off some of the modalities Planemo `test` should work in (there may be bugs but we are trying to stablize this functionality).

```
#!/bin/bash

# Usage: http://planemo.readthedocs.io/en/latest/test_format.html#galaxy-testing-template

: ${PLANEMO_TARGET:="planemo==0.52.0"}
: ${PLANEMO_OPTIONS:=""}  # e.g. PLANEMO_OPTIONS="--verbose"
: ${PLANEMO_PROFILE_NAME:="wxflowtest"}
: ${PLANEMO_SERVE_PORT:="9019"}
: ${PLANEMO_GALAXY_BRANCH:="master"}
: ${PLANEMO_TEST_STYLE:="serve_and_test"}  # profile_serve_and_test, serve_and_test,␣
↪docker_serve_and_test, test, docker_test, docker_test_path_paste
: ${PLANEMO_SERVE_DATABASE_TYPE:="postgres"}  # used if not using Docker with PLANEMO_
↪TEST_STYLE
: ${PLANEMO_DOCKER_GALAXY_IMAGE:="quay.io/bgruening/galaxy:20.05"}  # used if used␣
↪Docker with PLANEMO_TEST_STYLE
: ${PLANEMO_VIRTUAL_ENV:=".venv"}
: ${GALAXY_URL:="http://localhost:$PLANEMO_SERVE_PORT"}

# Ensure Planemo is installed.
if [ ! -d "${PLANEMO_VIRTUAL_ENV}" ]; then
    virtualenv "${PLANEMO_VIRTUAL_ENV}"
    . "${PLANEMO_VIRTUAL_ENV}"/bin/activate
    pip install -U pip>7
    # Intentionally expand wildcards in PLANEMO_TARGET.
    shopt -s extglob
    pip install ${PLANEMO_TARGET}
fi
. "${PLANEMO_VIRTUAL_ENV}"/bin/activate

# Run test.
# This example shows off a bunch of different ways one could test with Planemo,
# but for actual workflow testing projects - probably best just to take one of the last
# two very easy invocations to simplify things.
if [ "$PLANEMO_TEST_STYLE" = "profile_serve_and_test" ]; then
    planemo $PLANEMO_OPTIONS profile_create \
        --database_type "$PLANEMO_SERVE_DATABASE_TYPE" \
        "$PLANEMO_PROFILE_NAME"
    planemo $PLANEMO_OPTIONS serve \
        --daemon \
        --galaxy_branch "$PLANEMO_GALAXY_BRANCH" \
        --profile "$PLANEMO_PROFILE_NAME" \
        --port "$PLANEMO_SERVE_PORT" \
        "$1"
    planemo $PLANEMO_OPTIONS test \
        --galaxy_url "$GALAXY_URL" \
        --engine external_galaxy \
        "$1"
```

(continues on next page)

(continued from previous page)

```
elif [ "$PLANEMO_TEST_STYLE" = "serve_and_test" ]; then
    planemo $PLANEMO_OPTIONS serve \
        --daemon \
        --galaxy_branch "$PLANEMO_GALAXY_BRANCH" \
        --database_type "$PLANEMO_SERVE_DATABASE_TYPE" \
        --port "$PLANEMO_SERVE_PORT" \
        "$1"
    planemo $PLANEMO_OPTIONS test \
        --galaxy_url "$GALAXY_URL" \
        --engine external_galaxy \
        "$1"
elif [ "$PLANEMO_TEST_STYLE" = "docker_serve_and_test" ]; then
    docker pull "${PLANEMO_DOCKER_GALAXY_IMAGE}"
    planemo $PLANEMO_OPTIONS serve \
        --daemon \
        --engine docker_galaxy \
        --docker_galaxy_image "${PLANEMO_DOCKER_GALAXY_IMAGE}" \
        --port "$PLANEMO_SERVE_PORT" \
        "$1"
    planemo $PLANEMO_OPTIONS test \
        --galaxy_url "$GALAXY_URL" \
        --engine external_galaxy \
        "$1"
elif [ "$PLANEMO_TEST_STYLE" = "test" ]; then
    # TODO: this conda_init shouldn't be needed, but this mode is broken without it.
    planemo conda_init || true

    planemo $PLANEMO_OPTIONS test \
        --database_type "$PLANEMO_SERVE_DATABASE_TYPE" \
        --galaxy_branch "$PLANEMO_GALAXY_BRANCH" \
        "$1"
elif [ "$PLANEMO_TEST_STYLE" = "docker_test" ]; then
    # TODO: This variant isn't super usable yet because there is too much logging, hence␣
↪the dev null
    # redirect.
    docker pull "${PLANEMO_DOCKER_GALAXY_IMAGE}"
    planemo $PLANEMO_OPTIONS test \
        --engine docker_galaxy \
        --docker_galaxy_image "${PLANEMO_DOCKER_GALAXY_IMAGE}" \
        "$1" > /dev/null
elif [ "$PLANEMO_TEST_STYLE" = "docker_test_path_paste" ]; then
    # Same as above but mount the test data and use file:// path pastes when uploading
    # files (more robust and quick if working with really large files).
    docker pull "${PLANEMO_DOCKER_GALAXY_IMAGE}"
    planemo $PLANEMO_OPTIONS test \
        --engine docker_galaxy \
        --docker_extra_volume . \
        --paste_test_data_paths \
        --docker_galaxy_image "${PLANEMO_DOCKER_GALAXY_IMAGE}" \
        "$1" > /dev/null
elif [ "$PLANEMO_TEST_STYLE" = "manual_docker_run_and_test" ]; then
    docker pull "${PLANEMO_DOCKER_GALAXY_IMAGE}"
```

(continues on next page)

(continued from previous page)

```
    docker run -d -e "NONUSE=nodejs,proftp,reports" -p "${PLANEMO_SERVE_PORT}:80" "$
↪{PLANEMO_DOCKER_GALAXY_IMAGE}"
    galaxy-wait -g "http://localhost:${PLANEMO_SERVE_PORT}"
    planemo $PLANEMO_OPTIONS test \
        --engine external_galaxy \
        --galaxy_url "$GALAXY_URL" \
        --galaxy_admin_key admin \
        --galaxy_user_key admin \
        "$1"
elif [ "$PLANEMO_TEST_STYLE" = "external_galaxy" ]; then
    if [[ -n $PLANEMO_INSTALL_TOOLS ]]; then
        INSTALL_TOOLS="";
    else
        INSTALL_TOOLS="--no_shed_install";
    fi
    planemo $PLANEMO_OPTIONS test \
        --engine external_galaxy \
        --galaxy_url "$GALAXY_URL" \
        --galaxy_admin_key "$PLANEMO_ADMIN_KEY" \
        --galaxy_user_key "$PLANEMO_USER_KEY" \
        $INSTALL_TOOLS \
        "$1"
else
    echo "Unknown test style ${PLANEMO_TEST_STYLE}"
    exit 1
fi
```

A Travis configuration file (.travis.yml) that would test workflows using a Docker Galaxy image might look like:

```
language: python
sudo: true
python: 2.7
env:
  global:
    - PLANEMO_TEST_SCRIPT=https://raw.githubusercontent.com/galaxyproject/planemo/master/
↪scripts/run_galaxy_workflow_tests.sh
    - PLANEMO_TEST_STYLE=docker_serve_and_test
    - PLANEMO_TARGET="planemo==0.52.0"
    - PLANEMO_DOCKER_GALAXY_IMAGE="quay.io/bgruening/galaxy:18.01"
  matrix:
    - WORKFLOW_TEST=example1/ref-rnaseq.ga
    - WORKFLOW_TEST=example2/chipseq.ga

script: bash <(curl -s "$PLANEMO_TEST_SCRIPT") "$WORKFLOW_TEST"

services:
  - docker
```

To skip Docker and instead test with a native Galaxy instance and postgres database one might use the configuration:

```
language: python
python: 2.7
```

(continues on next page)

(continued from previous page)

```
env:
  global:
    - PLANEMO_TEST_SCRIPT=https://raw.githubusercontent.com/galaxyproject/planemo/master/
↪scripts/run_galaxy_workflow_tests.sh
    - PLANEMO_TEST_STYLE=serve_and_test
    - PLANEMO_TARGET="planemo==0.52.0"
    - PLANEMO_GALAXY_BRANCH="release_18.05"
   matrix:
    - WORKFLOW_TEST=example1/ref-rnaseq.ga
    - WORKFLOW_TEST=example2/chipseq.ga

script: bash <(curl -s "$PLANEMO_TEST_SCRIPT") "$WORKFLOW_TEST"

services:
  - postgres
```

CHAPTER

NINE

BEST PRACTICES FOR MAINTAINING GALAXY WORKFLOWS

There are a number of things the user interface of Galaxy will allow that are not considered best practices because they make the workflow harder to test, use within subworkflows and invocation reports, and consume via the API. It is easier to use workflows in all of these contexts if they stick to the best practices discussed in this document.

Many of these best practices can be checked with Planemo using the follow command:

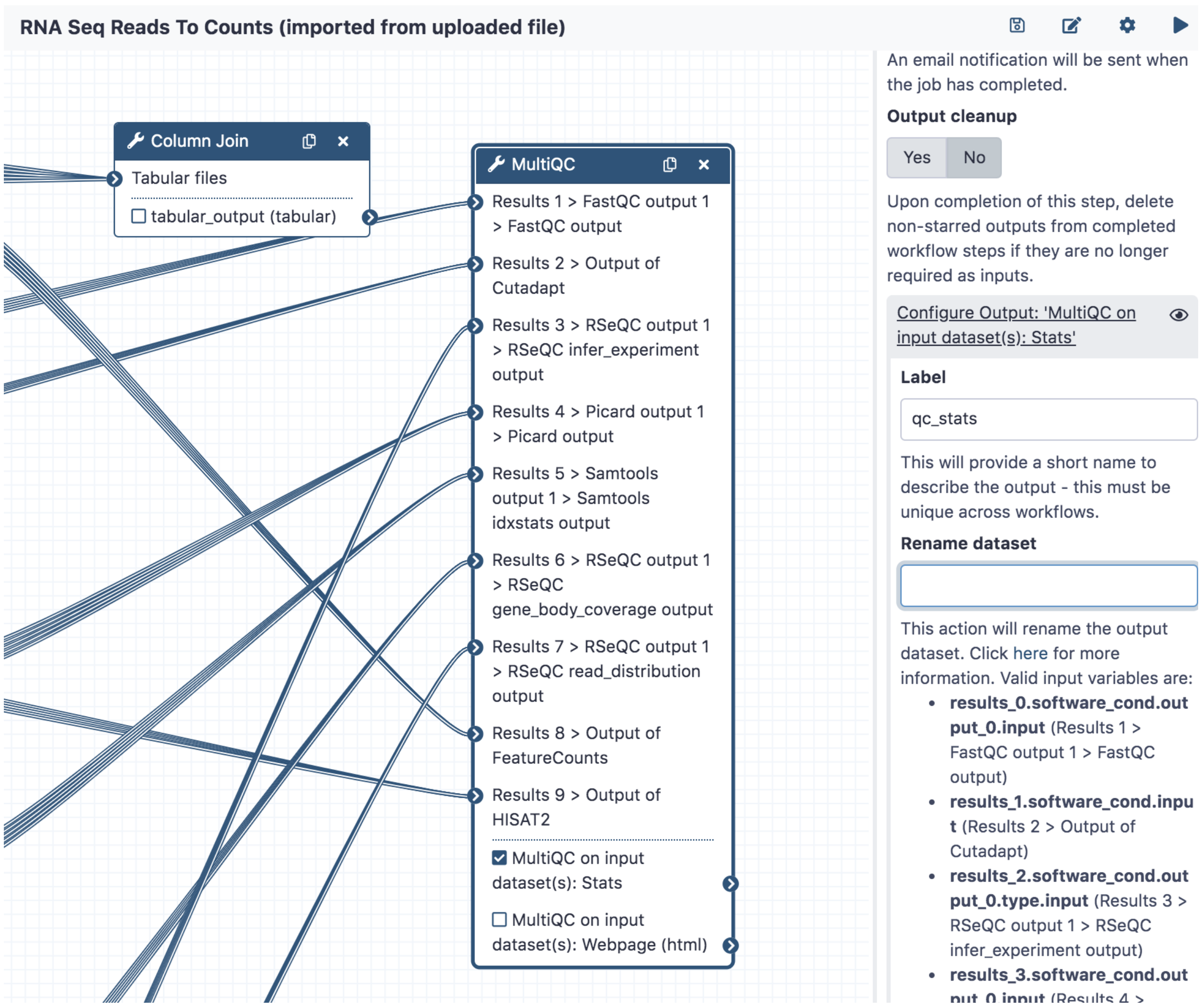
```
$ planemo workflow_lint path/to/workflow.ga
```

If `workflow_lint` is sent a directory, it will scan the directory for workflow artifacts - including Galaxy workflows and Dockstore `.dockstore.yml` files that register Galaxy workflows.

9.1 Workflow Structure

9.1.1 Outputs

Workflows should define explicit, labelled outputs. Galaxy doesn't require you to declare outputs explicitly or label them - but doing so provides a lot of advantages. A workflow with declared, labelled outputs specifies an explicit interface that is much easier to consume when building a report for the workflow, testing the workflow, using the workflow via the API, and using the workflow as a subworkflow in Galaxy.


RNA Seq Reads To Counts (imported from uploaded file)

Column Join
Tabular files
☐ tabular_output (tabular)

MultiQC
Results 1 > FastQC output 1 > FastQC output
Results 2 > Output of Cutadapt
Results 3 > RSeQC output 1 > RSeQC infer_experiment output
Results 4 > Picard output 1 > Picard output
Results 5 > Samtools output 1 > Samtools idxstats output
Results 6 > RSeQC output 1 > RSeQC gene_body_coverage output
Results 7 > RSeQC output 1 > RSeQC read_distribution output
Results 8 > Output of FeatureCounts
Results 9 > Output of HISAT2
☑ MultiQC on input dataset(s): Stats
☐ MultiQC on input dataset(s): Webpage (html)

An email notification will be sent when the job has completed.

Output cleanup

Yes No

Upon completion of this step, delete non-starred outputs from completed workflow steps if they are no longer required as inputs.

Configure Output: 'MultiQC on input dataset(s): Stats'

Label

qc_stats

This will provide a short name to describe the output - this must be unique across workflows.

Rename dataset

This action will rename the output dataset. Click here for more information. Valid input variables are:

- **results_0.software_cond.output_0.input** (Results 1 > FastQC output 1 > FastQC output)
- **results_1.software_cond.input** (Results 2 > Output of Cutadapt)
- **results_2.software_cond.output_0.type.input** (Results 3 > RSeQC output 1 > RSeQC infer_experiment output)
- **results_3.software_cond.output_0.input** (Results 4 >

The above screenshot demonstrates a MultiQC node with its stats output marked as an output and labelled as `qc_stats`.

Note: The Galaxy workflow editor will help you ensure that workflow output labels are unique across a workflow. Also be sure to add labels to your outputs before using a workflow as a subworkflow in another workflow so less stable and less contextualized step indicies and tool output names don't need to be used.

9.1.2 Inputs

Similarly to outputs and for similar reasons, all inputs should be explicit (with labelled input nodes) and tool steps should not have disconnected data inputs (even though the GUI can handle this) or consume workflow parameters. Older style "runtime parameters" should only be used for post job actions and newer type workflow parameter inputs should be used to manipulate tool logic.

A full tutorial on building Galaxy workflows with newer explicit workflow parameters can be found as part of the Galaxy Training Network.

In addition to making the interface easier to use in the context of subworkflows, the API, testing, etc.. future enhancements to Galaxy will allow a much simpler UI for workflows that only use explicit input parameters in this fashion.

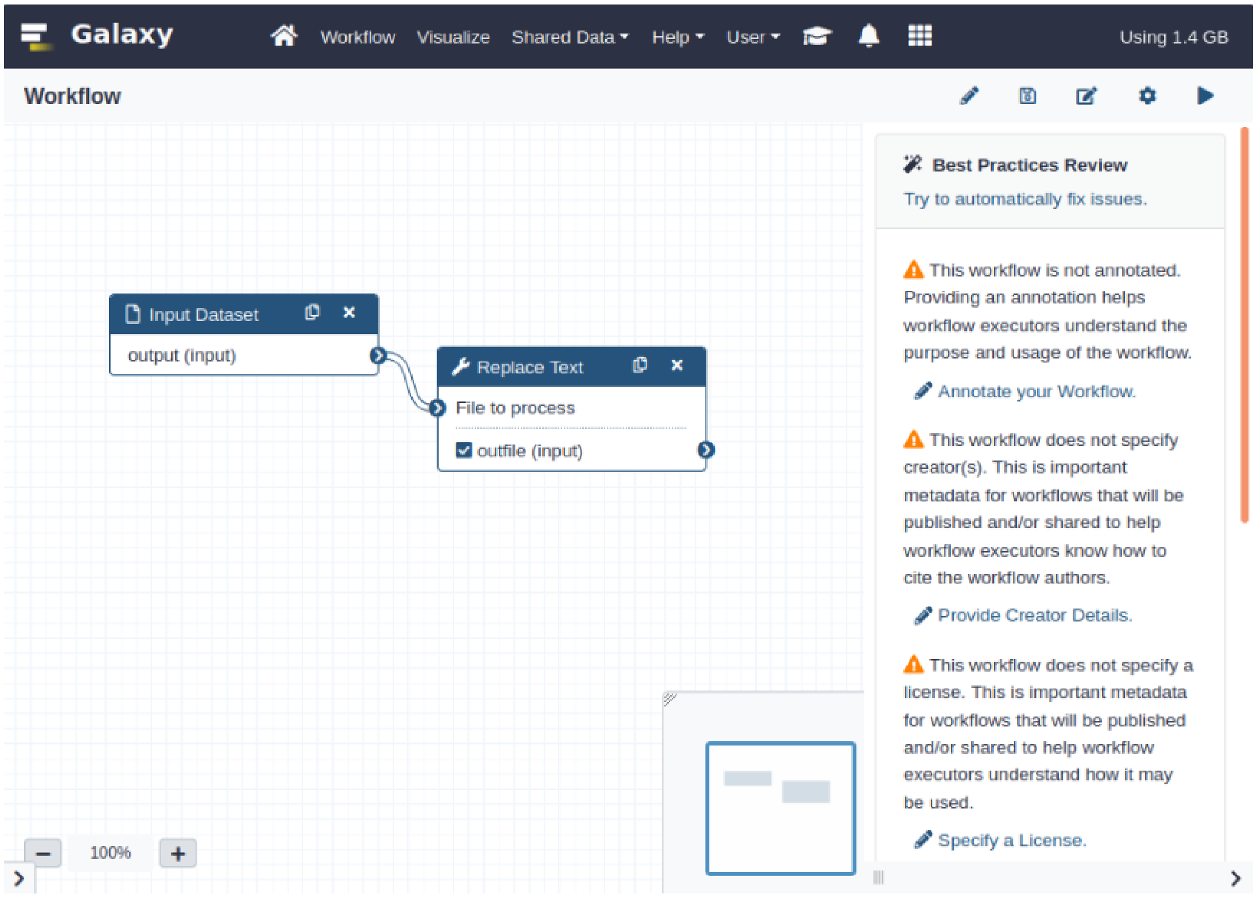
https://github.com/galaxyproject/galaxy/pull/9151

9.1.3 Tools

The tools used within a workflow should be packaged with Galaxy by default or published to the main Galaxy ToolShed. Using private tool sheds or the test tool shed limits the ability of other Galaxy's to use the workflow. In addition, `workflow_lint` will check that the tools starting with `toolshed.g2.bx.psu.edu` are available in the toolshed.

9.1.4 Syntax

Planemo's `workflow_lint` also checks if workflows have the correct JSON or YAML syntax, and ensures workflows follow certain 'best practices'. Best practices can also be checked in the 'Workflow Best Practices' panel of Galaxy's workflow editor and to some extent automatically fixed.



Best Practices Review

Try to automatically fix issues.

This workflow is not annotated. Providing an annotation helps workflow executors understand the purpose and usage of the workflow.

Annotate your Workflow.

This workflow does not specify creator(s). This is important metadata for workflows that will be published and/or shared to help workflow executors know how to cite the workflow authors.

Provide Creator Details.

This workflow does not specify a license. This is important metadata for workflows that will be published and/or shared to help workflow executors understand how it may be used.

Specify a License.

The `workflow_lint` subcommand allows the same checks to be made via the command line; this may be less of a problem for workflows exported from a Galaxy instance but can assist with workflows hand-edited or implemented using the newer YAML gxformat2 syntax.

```
$ planemo workflow_lint path/to/workflow.ga
```

Running this command makes the following checks:

- The workflow is annotated
- The workflow specifies a creator

- The workflow specifies a license
- All workflow steps are connected to formal input parameters
- All workflow steps are annotated and labelled
- No legacy 'untyped' parameters are used, e.g. a variable such as *${report_name}* in an input field
- All outputs are labelled.

In addition to checking the structure of the JSON or YAML file and workflow best practices, `workflow_lint` also checks the workflow test file is formatted correctly and at least one valid test is specified.

9.2 Tests

Writing workflow tests allows consumers of your workflow to know it works in their Galaxy environment and can allow for richer continuous integration (CI). Check out the Planemo Test Format documentation for more information on the format and how to test workflows with Planemo.

Planemo can help you stud out tests for a workflow developed within the UI quickly with the `workflow_test_init` command.

```
$ planemo workflow_test_init path/to/workflow.ga
```

This command creates a template test file, with inputs, parameters and expected outputs left blank for you to fill in. If you've already run the workflow on an external Galaxy server, you can generate a more complete test file directly from the invocation ID using the `--from_invocation` option.

```
$ planemo workflow_test_init --from_invocation <INVOCATION_ID> --galaxy_url <GALAXY␣
↪SERVER URL> --galaxy_user_key" <GALAXY API KEY>
```

You also need to specify the server URL and your API key, as Galaxy invocation IDs are only unique to a particular server. You can obtain the invocation ID from `<GALAXY SERVER URL>/workflows/invocations`.

9.3 Publishing

Unlike with Galaxy tools - the Galaxy team doesn't endorse a specific registry for Galaxy workflows. But also unlike Galaxy tools, any user can just paste a URL for a workflow right into the user interface so sharing a workflow can be as easy as passing around a GitHub link.

9.3.1 Github

Even if you're publishing your workflows to other registries or website, we always recommend publishing workflows to Github (or a publicly available Gitlab server).

9.3.2 Dockstore

A repository containing Galaxy workflows and published to GitHub can be registered with Dockstore. This allows others to search for the workflow and access it using standard GA4GH APIs. In the future, deep bi-directional integration between Galaxy and Dockstore will be available that will make these workflows even more useful.

A `.dockstore.yml` file should be placed in the root of your workflow repository before registering the repository with Dockstore. This will allow Dockstore to find your workflows and their tests automatically.

Planemo can create this file for you by executing the `dockstore_init` command from the root of your workflow repository

```
$ planemo dockstore_init
```

Planemo's `workflow_lint` will check the contents of your `.dockstore.yml` file during execution if this file is present.

9.3.3 Workflow Hub

Information on uploading workflows to workflowhub.eu can be found here.

CHAPTER

TEN

RUNNING GALAXY WORKFLOWS

Planemo offers a number of convenient commands for working with Galaxy workflows. Workflows are made up of a number of individual tools, which are executed in sequence, automatically. They allow Galaxy users to perform complex analyses made up of multiple simple steps.

Workflows can be easily created, edited and run using the Galaxy user interface (i.e. in the web-browser), as is described in the workflow tutorial provided by the Galaxy Training Network. However, in some circumstances, executing workflows may be awkward via the graphical interface. For example, you might want to run workflows a very large number of times, or you might want to automatically trigger workflow execution as a particular time as new data becomes available. For these applications, being able to execute workflows via the command line is very useful. This tutorial provides an introduction to the `planemo run` command, which allows Galaxy tools and workflows to be executed simply via the command line.

10.1 The Basics

This tutorial will demonstrate workflow execution with a very simple test workflow from the workflow-testing repository. This repository contains a number of workflows which are tested regularly against the European Galaxy server.

```
$ git clone https://github.com/usegalaxy-eu/workflow-testing.git
$ cd workflow-testing/example3
$ ls
data  tutorial.ga  tutorial-job.yml  tutorial-tests.yml
```

The `example3` directory contains three files. Firstly, `tutorial.ga` contains a complete definition of the workflow in JSON format, which can be easily exported from any Galaxy server after creating a new workflow. Secondly, `tutorial-job.yml` contains a list of the files and parameters (in YAML format) which should be used for each of the workflow inputs upon execution. Thirdly, `tutorial-tests.yml` contains a list of tests (similar to Galaxy tool tests) which can be used to validate the outputs of the workflow.

The `tutorial.ga` workflow takes two input datasets and one input parameter, and consists of two steps; firstly, `Dataset 1` and `Dataset 2` are concatenated together, and secondly, a certain number of lines (specified by the `Number of lines` parameter) are randomly selected. If you want to view it in the Galaxy interface, you can do so with the command `planemo workflow_edit tutorial.ga`.

The simplest way to run a workflow with planemo is on a locally hosted Galaxy instance, just like executing a tool test with `planemo test`. This can be achieved with the command

```
$ planemo run tutorial.ga tutorial-job.yml --download_outputs --output_directory . --
↪output_json output.json
```

You can optionally (and probably should) add the `--galaxy_root` flag with the location of a local copy of the Galaxy source code, which will allow the instance to be spun up considerably faster.

Note that `--download_outputs --output_directory . --output_json output.json` is optional, but allow saving the output to a local file. The contents should be something like:

```
$ cat tutorial_output.txt
is
hello
world
$ cat output.json
{"output": {"class": "File", "path": "/home/user/workflow-testing/example3/tutorial_
→output.txt", "checksum": "sha1$4d7ab2b2bb0102ee5ec472a5971ca86081ff700c", "size": 15,
→"basename": "tutorial_output.txt", "nameroot": "tutorial_output", "nameext": ".txt"}}
```

You can also run the workflow on a local Dockerized Galaxy. For this, exactly the same command can be used, with `--engine docker_galaxy --ignore_dependency_problems` appended. Please note that you need to have Docker installed and that it may take significantly longer to complete than the previous command.

```
$ planemo run tutorial.ga tutorial-job.yml --download_outputs --output_directory . --
→output_json output.json --engine docker_galaxy --ignore_dependency_problems
```

This introduces the concept of an engine, which Planemo provides to allow workflows to be flexibly executed using the setup and workflow execution system of the user's choice. The full list of engines provided by Galaxy is: `galaxy` (the default, used in the first example above), `docker_galaxy`, `cwltool`, `toil` and `external_galaxy`.

As a final example to demonstrate workflow testing, try:

```
$ planemo test tutorial.ga
```

In this case, Planemo automatically detects that it should test the workflow with the `tutorial-tests.yml`, so this file should be present and named correctly. If you inspect its contents:

```
$ cat tutorial-tests.yml
- doc: Test outline for tutorial.ga
  job:
    Dataset 1:
      class: File
      path: "data/dataset1.txt"
    Dataset 2:
      class: File
      path: "data/dataset2.txt"
    Number of lines: 3
  outputs:
    output:
      class: File
      path: "data/output.txt"
```

you see that the job parameters are defined identically to the `tutorial-job.yml` file, with the addition of an output. For the test to pass, the output file produced by the workflow must be identical to that stored in `data/output.txt`.

The three commands above demonstrate the basics of workflow execution with Planemo. For large scale workflow execution, however, it's likely that you would prefer to use the more extensive resources provided by a public Galaxy server, rather than running on a local instance. The tutorial therefore now turns to the use of the `galaxy_external` engine, which as the name suggests, runs workflows on a Galaxy external to Planemo.

10.2 Workflow execution against an external Galaxy

The first requirement for executing workflows on an external Galaxy server is a user account for that server. If you don't already have one, https://usegalaxy.org, https://usegalaxy.eu and https://usegalaxy.org.au all provide free accounts which can be used for this tutorial.

Assuming you have selected a server for this tutorial and have an account, you need to retrieve the API key associated with that account. This can be found at `{server_url}/user/api_key`, or by going to the 'User' dropdown menu, selecting 'Preferences' and then clicking on 'Manage API key'.

Now you can run the workflow with:

```
$ planemo run tutorial.ga tutorial-job.yml --engine external_galaxy --galaxy_url SERVER_
↪URL --galaxy_user_key YOUR_API_KEY
```

If you want to set the name of the history in which the workflow executes, add `--history_name NAME` to the command. You should be able to see the workflow executing in the web browser, if you navigate to the 'List all histories' view. If you prefer to download data without interacting with the web interface at all, you can add `--download_outputs --output_directory . --output_json output.json` to the command as before.

Typing `--engine external_galaxy --galaxy_url SERVER_URL --galaxy_user_key YOUR_API_KEY` each time you want to execute a workflow is a bit annoying. Fortunately, Planemo provides the option to create 'profiles' which save this information for you. To create a new profile called `tutorial_profile`, you can run a command like the following:

```
$ planemo profile_create tutorial_profile --galaxy_url SERVER_URL --galaxy_user_key YOUR_
↪API_KEY --engine external_galaxy
Profile [tutorial_profile] created.
```

This allows creation of multiple profiles (e.g. for different Galaxy servers). A list of all created profiles is provided by the `profile_list` subcommand:

```
$ planemo profile_list
Looking for profiles...
tutorial_profile
usegalaxy-eu
usegalaxy-org
3 configured profiles are available.
```

Once the new `tutorial_profile` is created, a workflow can be executed with:

```
$ planemo run tutorial.ga tutorial-job.yml --profile tutorial_profile
```

10.3 Generating the job file

The example workflow used so far provides not only the workflow, but also the job file which specifies the inputs to be used. If you have created and downloaded your own workflow, you need to create this job file yourself. As a first step, ensure that your workflow is linted correctly:

```
$ planemo workflow_lint tutorial.ga
Applying linter tests... CHECK
.. CHECK: Tests appear structurally correct
```

In this case, linting completes successfully, but you might see a message such as `WARNING: Workflow contained output without a label` or `WARNING: Test referenced File path not found`.

To generate the job file, you can now run:

```
$ planemo workflow_job_init tutorial.ga -o tutorial-init-job.yml
```

This generates a template for the job file which you can modify yourself. Opening `tutorial-init-job.yml` should show the following:

```
$ cat tutorial-init-job.yml
Dataset 1:
  class: File
  path: todo_test_data_path.ext
Dataset 2:
  class: File
  path: todo_test_data_path.ext
Number of lines: todo_param_value
```

For each of the specified inputs in the workflow, an entry is created in the output YAML file. The two dataset inputs are classified as `class:  File`, with a placeholder path included, which you should change to the paths of your chosen input files. You can also specify the URL of a file available online, by replacing the `path` attribute with `location` (e.g. `location:  https://website.org/file.txt`). The placeholder value for the `Number of lines` parameter should also be replaced, ensuring it is of the correct type, i.e. in this case an integer.

Another more complex example, also including a collection as input, might look like the following:

```
input_dataset:
  class: File
  path: todo_test_data_path.ext
input_collection:
  class: Collection
  collection_type: list
  elements:
  - class: File
    identifier: todo_element_name
    path: todo_test_data_path.ext
  - class: File
    identifier: todo_element_name
    path: todo_test_data_path.ext
input_parameter: todo_param_value
```

For the collection, each dataset is listed, with a path and identifier specified.

If you are creating a workflow for the first time, you should include tests to ensure it works in the way intended. These tests can be run using the `planemo test`, command, just like Galaxy tool testing (for more information, see here). These tests require a test file, similar to the job file used so far, which also specifies expected outputs which can be used to validate the workflow. An equivalent planemo command for creating a template for these test files is also available:

```
$ planemo workflow_test_init tutorial.ga -o tutorial-init-test.yml
$ cat tutorial-init-test.yml
- doc: Test outline for tutorial.ga
  job:
    Dataset 1:
      class: File
      path: todo_test_data_path.ext
```

(continues on next page)

(continued from previous page)

```
    Dataset 2:
      class: File
      path: todo_test_data_path.ext
    Number of lines: todo_param_value
  outputs:
    output:
      class: ''
```

10.4 Using workflow and dataset IDs

If you ran all the commands above then you probably noticed that both the workflow and the input datasets get newly uploaded at each execution. If you want to run the same workflow multiple times, you may prefer to avoid this. In the examples given so far, all workflows and datasets are specified by means of a local path, but Planemo also allows you to use the IDs created by Galaxy as well. These IDs are unique to each Galaxy server, so this approach isn't transferrable if you want to run your workflows on multiple servers.

The first step is to ensure all the datasets which are required for the workflow are already uploaded. You can either do this by running the workflow once in the normal way, as described above, or just manually uploading through the web interface.

To get dataset IDs, you can click on the dataset's 'View details' button (a small letter 'i' in a circle). This provides various information about the dataset and the job which created it. Under the 'Job information' section, there is a row named 'History Content API ID'. For each input dataset, copy this string (it will probably look something like `457d46215431cc37baf96108ad87f351`) and paste it into the workflow job file so it looks something like the following:

```
Dataset 1:
  class: File
  galaxy_id: "457d46215431cc37baf96108ad87f351"
Dataset 2:
  class: File
  galaxy_id: "55f30adf41ae36455431abeaa185ed89"
Number of lines: 3
```

i.e. just replace the `path` line with `galaxy_id`.

You can do exactly the same with a collection; either of the following will work:

```
input_collection1:
  class: Collection
  galaxy_id: "9d362c51f575db89"
input_collection2:
  class: Collection
  collection_type: list
  elements:
  - class: File
    identifier: element 1
    galaxy_id: "457d46215431cc37baf96108ad87f351"
```

For `input_collection1`, an existing collection will be used (by specifying its collection ID), whereas for `input_collection2`, a new collection will be created from a list of existing datasets.

Once the job file has been modified, run `planemo run` as before. The result should be the same, though it should be a bit faster, since the upload step was skipped. Instead, the selected datasets get copied to a new history, which unlike a new upload, doesn't result in any additional storage being used.

To run the workflow using a workflow ID, replace the workflow file path with the workflow ID from the Galaxy server:

```
$ planemo run 501da2f0ba775fd0 tutorial-job.yml --profile tutorial_profile
```

10.5 Using aliases

Once you are dealing with a large number of workflows and datasets, you may find that it becomes difficult to keep track of the file paths or IDs which you are using for execution, particularly if you are executing workflows based on their ID. Planemo offers the option to create aliases, or easily memorable mnemonics, for Galaxy workflows, with the following command:

```
$ planemo create_alias 501da2f0ba775fd0 --alias my_favorite_workflow --profile tutorial_
↪profile
```

You can then execute the workflow with:

```
$ planemo run my_favorite_workflow tutorial-job.yml --profile tutorial_profile
```

Note that aliases are associated with a particular profile, so if you want to execute the same workflow with multiple profiles, you should recreate the alias for each one. Aliases can be created either for workflow IDs (as above) or for workflow file paths. You can list all aliases associated with a profile with:

```
$ planemo list_alias --profile tutorial_profile
```

10.6 Checking invocations

Assuming you know the workflow ID (or an alias for it), you can get a list of all created invocations with:

```
$ planemo list_invocations my_favorite_workflow --profile tutorial_profile
```

This indicates the number of datasets created, as well as the state they are in (running, errored, paused, etc.)

10.7 Profile configuration files

Information about each of the files is located in a configuration file, located at `~/.planemo/profiles/{profile_name}/planemo_profile_options.json`.

If you ran all the commands in this tutorial, the contents should be similar to the following:

```
$ cat ~/.planemo/profiles/tutorial_profile/planemo_profile_options.json
{
  "galaxy_url": "SERVER_URL",
  "galaxy_user_key": "YOUR_API_KEY",
  "galaxy_admin_key": null,
  "engine": "external_galaxy",
```

(continues on next page)

(continued from previous page)

```
  "aliases": {
    "my_favorite_workflow": "501da2f0ba775fd0"
  }
}
```

You can also delete unwanted profiles or aliases with these commands:

```
$ planemo delete_alias --alias my_favorite_workflow --profile tutorial_profile
$ planemo profile_delete tutorial_profile
```

10.8 Rerunning failed jobs

A frequent issue that arises when running a complex workflow is that component jobs can fail, resulting in failure of the entire workflow. These jobs can be rerun in the graphical interface, selecting the `Resume dependencies from this job ?` option, which restarts the paused workflow (so-called 'remapping' of the failed job over the previously created output dataset(s)). However, if there are a large number of failures, and you believe the errors are transitory, e.g. due to some temporary server issue, you can rerun several failed jobs simultaneously using the `planemo rerun` command:

```
$ planemo rerun --history 68008488b4fb94de
$ planemo rerun --invocation 27267240b7d1f22a a9b086729787c907c
$ planemo rerun --job a2b39deaa34509bb 3318707f2f0ff1fd
```

In the first two cases, all failed, remappable jobs which are associated with the specified history(s) or invocation(s) will be rerun. In the third case, the specified jobs will simply be rerun.

CHAPTER

ELEVEN

AUTOMATING GALAXY WORKFLOWS

The BioBlend library can be used to invoke and monitor Galaxy workflows. Planemo provides a higher-level interfaces to working with workflows - both as a Python library and via the command line. Planemo can take care of orchestrating details such as launching and configuring a Galaxy instance, installing required tools for the workflow, monitoring the workflow invocation, and downloading the results back to a local directory.

```
planemo run workflow.ga job.yml
```

The format of the job file should be YAML or JSON and matches the job definition used by the workflow test format.

A job template for a particular workflow can be created with the `workflow_job_init` command.

```
planemo workflow_job_init my-workflow.ga
```

This will create a `my-workflow-job.yml` file in the current directory.

Planemo run and the underlying Python code when used as a library support a wide range of arguments for how to run a Galaxy workflow.

CHAPTER

TWELVE

AUTOUPDATING TOOLS

Galaxy tools use underlying command-line software which are specified using `<requirement>` tags in the XML wrapper. If developers continue to release new versions of the software, the creator/maintainer needs to ensure that the tool requirements are bumped, otherwise the tool will become updated.

Planemo provides a `autoupdate` subcommand which can be used to perform this task automatically. Basic usage is as follows:

```
planemo autoupdate tool_wrapper.xml
```

There are various flags which can be applied; these are some of the most important:

- `--recursive`, which performs the autoupdate recursively on subdirectories
- `--dry-run`, which checks if tool requirements are up-to-date without making the necessary change automatically
- `--test`, which runs tests on all autoupdated tools. If this flag is used, all options available for `planemo test` are also available.
- `--update_test_data` (if `--test` is also selected) which will update test data for failed tests
- `--skiplist`, pointing to a list of tool wrappers for which updates should be skipped
- `--skip_requirements` with a comma-separated list of packages not to update. `python`, `perl`, `r-base` are skipped by default.

One of the most efficient ways to use it is by implementing a CI cron job which runs the command on an entire GitHub repository of tool wrappers.

12.1 Formatting tools

`autoupdate` assumes that XML tools are formatted in a certain way - in accordance with the IUC best practices. In particular:

- the tool `version` attribute must be set to `@TOOL_VERSION@+galaxy0` or `@TOOL_VERSION@+galaxy@VERSION_SUFFIX@`
- A token `@TOOL_VERSION@` should be created which corresponds to the version number of the main requirement.
- Optionally, a token `@VERSION_SUFFIX@` should be created, which should be an integer representing the number of times the XML wrapper has been updated since `@TOOL_VERSION@` was updated.

12.2 Updating workflows

The `autoupdate` subcommand can also be used to automatically update workflows so that they are using the most recent Galaxy tools available.

```
planemo autoupdate workflow.ga
```

In the basic usage with the above command, a local Galaxy instance will be spun up and the workflow uploaded, refactored to include the most recent tool versions, and re-downloaded.

Workflows can also be updated against an external Galaxy server; see the example below. (Please note the server must be running Galaxy version 21.05 or more recent.)

```
planemo autoupdate workflow.ga --profile usegalaxy-eu
```

In this case, the workflow returned will contain the most recent tool and subworkflow versions available on that Galaxy server.

An equivalent alternative, if you don't want to use Planemo profiles, is the following:

```
planemo autoupdate workflow.ga --galaxy_url SERVER_URL --galaxy_user_key API_KEY
```

12.3 Implementing an autoupdate CI job

A useful way to use the autoupdate command is to implement it as a CI job, so that tools in a repo can be updated on a regular basis (e.g. weekly). An example implementation is the planemo-autoupdate GitHub bot.

CHAPTER

THIRTEEN

COMMANDS

Planemo is a set of utilities for developing Galaxy tools. Each utility is implemented as a subcommand of the `planemo` executable. This section of the documentation describes these commands.

13.1 `autoupdate` command

This section is auto-generated from the help text for the planemo command `autoupdate`. This help message can be generated with `planemo autoupdate --help`.

Usage:

```
planemo autoupdate [OPTIONS] TOOL_PATH
```

Help

Auto-update tool requirements by checking against Conda and updating if newer versions are available. **Options**:

```
--dry-run                       Perform a dry run autoupdate without modifying
                                the XML files.
-r, --recursive                 Recursively perform command for
                                subdirectories.
--test                          Test updated XML files.
--skiplist TEXT                 Skiplist file, containing a list of tools or
                                workflows for which autoupdate should be
                                skipped.
--skip_requirements TEXT        Comma-separated list of requirements which
                                should be not be updated. Default is
                                python,r-base,perl.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--paste_test_data_paths / --no_paste_test_data_paths
                                By default Planemo will use or not use
                                Galaxy's path paste option to load test data
                                into a history based on the engine type it is
                                targeting. This can override the logic to
                                explicitly enable or disable path pasting.
--update_test_data              Update test-data directory with job outputs
                                (normally written to directory
```

(continues on next page)

(continued from previous page)

```
                                 --job_output_files if specified.)
--test_output PATH               Output test report (HTML - for humans)
                                 defaults to tool_test_output.html.
--test_output_text PATH          Output test report (Basic text - for display
                                 in CI)
--test_output_markdown PATH      Output test report (Markdown style - for
                                 humans & computers)
--test_output_xunit PATH         Output test report (xunit style - for CI
                                 systems
--test_output_junit PATH         Output test report (jUnit style - for CI
                                 systems
--test_output_allure DIRECTORY   Output test allure2 framework resutls
--test_output_json PATH          Output test report (planemo json) defaults to
                                 tool_test_output.json.
--job_output_files DIRECTORY     Write job outputs to specified directory.
--summary [none|minimal|compact]
                                 Summary style printed to planemo's standard
                                 output (see output reports for more complete
                                 summary). Set to 'none' to disable completely.
--test_timeout INTEGER           Maximum runtime of a single test in seconds.
--galaxy_root DIRECTORY          Root of development galaxy directory to
                                 execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                 Python version to start Galaxy under
--extra_tools PATH               Extra tool sources to include in Galaxy's tool
                                 panel (file or directory). These will not be
                                 linted/tested/etc... but they will be
                                 available to workflows and for interactive
                                 use.
--install_galaxy                 Download and configure a disposable copy of
                                 Galaxy from github.
--galaxy_branch TEXT             Branch of Galaxy to target (defaults to
                                 master) if a Galaxy root isn't specified.
--galaxy_source TEXT             Git source of Galaxy to target (defaults to
                                 the official galaxyproject github source if a
                                 Galaxy root isn't specified.
--skip_venv                      Do not create or source a virtualenv
                                 environment for Galaxy, this should be used to
                                 preserve an externally configured virtual
                                 environment or conda environment.
--no_cache_galaxy                Skip caching of Galaxy source and dependencies
                                 obtained with --install_galaxy. Not caching
                                 this results in faster downloads (no git) - so
                                 is better on throw away instances such with
                                 TravisCI.
--no_cleanup                     Do not cleanup temp files created for and by
                                 Galaxy.
--galaxy_email TEXT              E-mail address to use when launching single-
                                 user Galaxy server.
--docker / --no_docker           Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT                Command used to launch docker (defaults to
                                 docker).
```

(continues on next page)

(continued from previous page)

```
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
```

(continues on next page)

(continued from previous page)

```
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--report_level [all|warn|error]
--report_xunit PATH             Output an XUnit report, useful for CI testing
--fail_level [warn|error]
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
--galaxy_admin_key TEXT         Admin key to use with external Galaxy engine.
--help                          Show this message and exit.
```

13.2 ci_find_repos command

This section is auto-generated from the help text for the planemo command `ci_find_repos`. This help message can be generated with `planemo ci_find_repos --help`.

Usage:

```
planemo ci_find_repos [OPTIONS] PROJECT
```

Help

Find all shed repositories in one or more directories.

Currently, a repository is considered any directory with a .shed.yml or .dockstore.yml file.

Options:

```
--exclude PATH                  Paths to exclude.
--exclude_from FILE             File of paths to exclude.
--changed_in_commit_range TEXT  Exclude paths unchanged in git commit range.
--chunk_count INTEGER           Split output into chunks of this many item and
                                print --chunk such group.
--chunk INTEGER                 When output is split into --chunk_count
                                groups, output the group 0-indexedby this
                                option.
--output TEXT                   File to output to, or - for standard output.
--help                          Show this message and exit.
```

13.3 ci_find_tools command

This section is auto-generated from the help text for the planemo command `ci_find_tools`. This help message can be generated with `planemo ci_find_tools --help`.

Usage:

```
planemo ci_find_tools [OPTIONS] PROJECT
```

Help

Find all tools in one or more directories.

Tools can be chunked up, filtered, etc... to build lists of tools to perform operations over for continuous integration operations.

Options:

```
--exclude PATH                  Paths to exclude.
--exclude_from FILE             File of paths to exclude.
--changed_in_commit_range TEXT  Exclude paths unchanged in git commit range.
--chunk_count INTEGER           Split output into chunks of this many item and
                                print --chunk such group.
--chunk INTEGER                 When output is split into --chunk_count
                                groups, output the group 0-indexedby this
                                option.
--output TEXT                   File to output to, or - for standard output.
```

(continues on next page)

(continued from previous page)

```
--group_tools                   Group tools of the same repository on a single
                                line.
--help                          Show this message and exit.
```

13.4 ci_setup command

This section is auto-generated from the help text for the planemo command `ci_setup`. This help message can be generated with `planemo ci_setup --help`.

Usage:

```
planemo ci_setup [OPTIONS]
```

Help

Launch Galaxy instance, then terminate instance.

Useful for populating a CI cache.

Options:

```
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
```

(continues on next page)

(continued from previous page)

```
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--help                          Show this message and exit.
```

13.5 clone command

This section is auto-generated from the help text for the planemo command `clone`. This help message can be generated with `planemo clone --help`.

Usage:

```
planemo clone [OPTIONS] TARGET PROJECT
```

Help

Short-cut to quickly clone, fork, and branch a relevant Github repo.

For instance, the following will clone, fork, and branch the tools-iuc repository to allow a subsequent pull request to fix a problem with bwa.

```
$ planemo clone --branch bwa-fix tools-iuc
$ cd tools-iuc
$ # Make changes.
$ git add -p # Add desired changes.
$ git commit -m "Fix bwa problem."
$ planemo pull_request -m "Fix bwa problem."
```

These changes do require that a github access token is specified in ~/.planemo.yml. An access token can be generated by going to https://github.com/settings/tokens.

Options:

```
--fork / --skip_fork
--branch TEXT         Create a named branch on result.
--help                Show this message and exit.
```

13.6 conda_build command

This section is auto-generated from the help text for the planemo command `conda_build`. This help message can be generated with `planemo conda_build --help`.

Usage:

```
planemo conda_build [OPTIONS] RECIPE_DIR
```

Help

Perform conda build with Planemo's conda. **Options**:

```
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--help                          Show this message and exit.
```

13.7 conda_env command

This section is auto-generated from the help text for the planemo command `conda_env`. This help message can be generated with `planemo conda_env --help`.

Usage:

```
planemo conda_env [OPTIONS] TOOL_PATH
```

Help

Activate a conda environment for tool.

Source the output of this command to activate a conda environment for this tool.

```
$ . <(planemo conda_env seqtk_seq.xml)
Deactivate environment with conda_env_deactivate
(seqtk_seq_v6) $ which seqtk
/home/planemo/miniconda2/envs/
↪jobdepsDkzcjjfecc6d406196737781ff4456ec60975c137e04884e4f4b05dc68192f7cec4656/bin/seqtk
(seqtk_seq_v6) $ conda_env_deactivate
$
```

Options:

```
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--help                          Show this message and exit.
```

13.8 conda_init command

This section is auto-generated from the help text for the planemo command `conda_init`. This help message can be generated with `planemo conda_init --help`.

Usage:

```
planemo conda_init [OPTIONS]
```

Help

Download and install conda.

This will download conda for managing dependencies for your platform using the appropriate Miniconda installer.

By running this command, you are agreeing to the terms of the conda license a 3-clause BSD 3 license. Please review full license at http://docs.continuum.io/anaconda/eula.

Planemo will print a warning and terminate with an exit code of 7 if Conda is already installed.

Options:

```
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--help                          Show this message and exit.
```

13.9 conda_install command

This section is auto-generated from the help text for the planemo command `conda_install`. This help message can be generated with `planemo conda_install --help`.

Usage:

```
planemo conda_install [OPTIONS] TARGET
```

Help

Install conda packages for tool requirements. **Options**:

```
-r, --recursive                 Recursively perform command for
                                subdirectories.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--global                        Install Conda dependencies globally instead of
                                in requirement specific environments packaged
                                for tools. If the Conda bin directory is on
                                your PATH, tools may still use binaries but
                                this is more designed for interactive testing
                                and debugging.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--help                          Show this message and exit.
```

13.10 conda_search command

This section is auto-generated from the help text for the planemo command conda_search. This help message can be generated with planemo conda_search --help.

Usage:

```
planemo conda_search [OPTIONS] TERM
```

Help

Perform conda search with Planemo's conda.

Implicitly adds channels Planemo is configured with.

Options:

```
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--help                          Show this message and exit.
```

13.11 config_init command

This section is auto-generated from the help text for the planemo command `config_init`. This help message can be generated with `planemo config_init --help`.

Usage:

```
planemo config_init [OPTIONS] PROJECT
```

Help

Initialise global configuration for Planemo.

Helps initialize global configuration (in home directory) for Planemo.

Options:

```
--template TEXT
--help           Show this message and exit.
```

13.12 container_register command

This section is auto-generated from the help text for the planemo command `container_register`. This help message can be generated with `planemo container_register --help`.

Usage:

```
planemo container_register [OPTIONS] TOOL_PATH
```

Help

Register multi-requirement containers as needed.

BioContainers publishes all Bioconda packages automatically as individual container images. These however are not enough for tools with multiple best-practice requirements. Such requirements should be recorded and published so that a container can be created and registered for these tools.

Options:

```
-r, --recursive                 Recursively perform command for
                                subdirectories.
--mulled_namespace TEXT         Build a mulled image with the specified
                                namespace - defaults to biocontainers. Galaxy
                                currently only recognizes images with the
                                namespace biocontainers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--output_directory DIRECTORY    Container registration directory (defaults to
                                ~/.planemo/multi-package-containers.
```

(continues on next page)

(continued from previous page)

```
-m, --message TEXT              Commit and pull request message template for
                                registration interactions.
--pull_request / --no_pull_request
                                Fork and create a pull request against
                                BioContainers/multi-package-containers for
                                these changes.
--force_push / --no_force_push  Force push branch for pull request in case it
                                already exists.
--help                          Show this message and exit.
```

13.13 `create_alias` command

This section is auto-generated from the help text for the planemo command `create_alias`. This help message can be generated with `planemo create_alias --help`.

Usage:

```
planemo create_alias [OPTIONS] OBJ
```

Help

Add an alias for a path or a workflow or dataset ID. Aliases are associated with a particular planemo profile.

Options:

```
--alias TEXT    Name of an alias.
--profile TEXT  Name of profile (created with the profile_create command) to
                use with this command.  [required]
--help          Show this message and exit.
```

13.14 `database_create` command

This section is auto-generated from the help text for the planemo command `database_create`. This help message can be generated with `planemo database_create --help`.

Usage:

```
planemo database_create [OPTIONS] IDENTIFIER
```

Help

Create a *development* database.

Currently the only implementation is postgres which will be managed with `psql`.

Planemo `database_` commands make it very easy to create and destroy databases, therefore it should not be used for production data - and it should not even be connnected to a production database server. Planemo is intended for development purposes only.

Planemo will assume that it can manage and access postgres databases without specifying a password. This can be accomplished by configuring postgres to not required a password for the planemo user or by specifying a password in a `.pgpass` file.

Planemo can be configured to not require a password for the planemo user in the postgres configuration file `pg_hba.conf` (on Ubuntu/Debian linux distros this file is in /etc/postgresql/<postgres_version>/main/ directory). Adding the following lines to that file will allow planemo and Galaxy to access the databases without a password.

```
# "local" is for Unix domain socket connections only
local   all   all                     trust
# IPv4 local connections:
host    all   all    127.0.0.1/32     trust
# IPv6 local connections:
host    all   all    ::1/128          trust
```

More information on the `pg_hda.conf` configuration file can be found at http://www.postgresql.org/docs/9.3/static/auth-pg-hba-conf.html.

Information on `.pgpass` files can be found at at the following location: http://www.postgresql.org/docs/9.4/static/libpq-pgpass.html. In Ubuntu and Debian distros - a postgres user likely already exists and its password can be set by setting up a file `~/.pgpass` file with the following contents.

```
*:*:*:postgres:<postgres_password>
```

Options:

```
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--help                          Show this message and exit.
```

13.15 database_delete command

This section is auto-generated from the help text for the planemo command `database_delete`. This help message can be generated with `planemo database_delete --help`.

Usage:

```
planemo database_delete [OPTIONS] IDENTIFIER
```

Help

Delete a *development* database.

Currently the only implementation is postgres which will be managed with `psql`.

Planemo `database_` commands make it very easy to create and destroy databases, therefore it should not be used for production data - and it should not even be connnected to a production database server. Planemo is intended for development purposes only.

Planemo will assume that it can manage and access postgres databases without specifying a password. This can be accomplished by configuring postgres to not required a password for the planemo user or by specifying a password in a `.pgpass` file.

Planemo can be configured to not require a password for the planemo user in the postgres configuration file `pg_hba.conf` (on Ubuntu/Debian linux distros this file is in /etc/postgresql/<postgres_version>/main/ directory). Adding the following lines to that file will allow planemo and Galaxy to access the databases without a password.

```
# "local" is for Unix domain socket connections only
local    all    all                     trust
# IPv4 local connections:
host     all    all    127.0.0.1/32     trust
# IPv6 local connections:
host     all    all    ::1/128          trust
```

More information on the `pg_hda.conf` configuration file can be found at http://www.postgresql.org/docs/9.3/static/auth-pg-hba-conf.html.

Information on `.pgpass` files can be found at at the following location: http://www.postgresql.org/docs/9.4/static/libpq-pgpass.html. In Ubuntu and Debian distros - a postgres user likely already exists and its password can be set by setting up a file `~/.pgpass` file with the following contents.

```
*:*:*:postgres:<postgres_password>
```

Options:

```
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
```

(continues on next page)

(continued from previous page)

```
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--help                          Show this message and exit.
```

13.16 database_list command

This section is auto-generated from the help text for the planemo command `database_list`. This help message can be generated with `planemo database_list --help`.

Usage:

```
planemo database_list [OPTIONS]
```

Help

List databases in configured database source.

Currently the only implementation is postgres which will be managed with `psql`.

Planemo `database_` commands make it very easy to create and destroy databases, therefore it should not be used for production data - and it should not even be connnected to a production database server. Planemo is intended for development purposes only.

Planemo will assume that it can manage and access postgres databases without specifying a password. This can be accomplished by configuring postgres to not required a password for the planemo user or by specifying a password in a `.pgpass` file.

Planemo can be configured to not require a password for the planemo user in the postgres configuration file `pg_hba.conf` (on Ubuntu/Debian linux distros this file is in /etc/postgresql/<postgres_version>/main/ directory). Adding the following lines to that file will allow planemo and Galaxy to access the databases without a password.

```
# "local" is for Unix domain socket connections only
local   all   all                    trust
# IPv4 local connections:
host    all   all    127.0.0.1/32    trust
# IPv6 local connections:
host    all   all    ::1/128         trust
```

More information on the `pg_hda.conf` configuration file can be found at http://www.postgresql.org/docs/9.3/static/auth-pg-hba-conf.html.

Information on `.pgpass` files can be found at at the following location: http://www.postgresql.org/docs/9.4/static/libpq-pgpass.html. In Ubuntu and Debian distros - a postgres user likely already exists and its password can be set by setting up a file `~/.pgpass` file with the following contents.

```
*:*:*:postgres:<postgres_password>
```

Options:

```
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--help                          Show this message and exit.
```

13.17 `delete_alias` command

This section is auto-generated from the help text for the planemo command `delete_alias`. This help message can be generated with `planemo delete_alias --help`.

Usage:

```
planemo delete_alias [OPTIONS]
```

Help

List aliases for a path or a workflow or dataset ID. Aliases are associated with a particular planemo profile.

Options:

```
--alias TEXT    Name of an alias.  [required]
--profile TEXT  Name of profile (created with the profile_create command) to
                use with this command.  [required]
--help          Show this message and exit.
```

13.18 docker_build command

This section is auto-generated from the help text for the planemo command `docker_build`. This help message can be generated with `planemo docker_build --help`.

Usage:

```
planemo docker_build [OPTIONS] TOOL_PATH
```

Help

Build (and optionally cache) Docker images.

Loads the tool or tools referenced by `TOOL_PATH` (by default all tools in current directory), and ensures they all reference the same Docker image and then attempts to find a Dockerfile for these tools (can be explicitly specified with `--dockerfile` but by default it will check the tool's directory and the current directory as well).

This command will then build and tag the image so it is ready to be tested and published. The docker_shell command be used to test out the built image.

```
% planemo docker_build bowtie2.xml # asssumes Dockerfile in same dir
% planemo docker_shell --from_tag bowtie2.xml
```

This can optionally also cache the images.

Options:

```
--dockerfile TEXT
--docker_image_cache TEXT
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--help                          Show this message and exit.
```

13.19 docker_shell command

This section is auto-generated from the help text for the planemo command `docker_shell`. This help message can be generated with `planemo docker_shell --help`.

Usage:

```
planemo docker_shell [OPTIONS] TOOL_PATH
```

Help

Launch shell in Docker container for a tool.

Will launch a shell in the Docker container referenced by the specified tool. Prints a command to do this the way Galaxy would in job files it generates - so be sure to wrap this in $(. . .) to launch the subshell.

```
$ $(planemo docker_shell bowtie2.xml)
...
root@b8754062f875:/#
```

Options:

```
--from_tag                      Treat the tool's Docker container identifier
                                as a locally cached tag.
--shell TEXT                    Shell to launch in container (defaults to
                                /bin/bash).
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--help                          Show this message and exit.
```

13.20 dockstore_init command

This section is auto-generated from the help text for the planemo command `dockstore_init`. This help message can be generated with `planemo dockstore_init --help`.

Usage:

```
planemo dockstore_init [OPTIONS] PROJECT
```

Help

Initialize a .dockstore.yml configuration file for workflows in directory.

Walk supplied directory and find all Galaxy workflows and test configurations and create a `.dockstore.yml` with references to these files. Be sure to push this file to Github before registering your workflow repository with Dockstore.

Visit Dockstore at https://dockstore.org/. Find more about registering workflows with Dockstore at https://docs.dockstore.org/en/develop/getting-started/dockstore-workflows.html.

Options:

```
--publish / --no_publish  Set publish attribute to true in .dockstore.yml file
--help                    Show this message and exit.
```

13.21 docs command

This section is auto-generated from the help text for the planemo command `docs`. This help message can be generated with `planemo docs --help`.

Usage:

```
planemo docs [OPTIONS]
```

Help

Open Planemo documentation in web browser. **Options**:

```
--help  Show this message and exit.
```

13.22 lint command

This section is auto-generated from the help text for the planemo command `lint`. This help message can be generated with `planemo lint --help`.

Usage:

```
planemo lint [OPTIONS] TOOL_PATH
```

Help

Check for common errors and best practices. **Options**:

```
--report_level [all|warn|error]
--report_xunit PATH             Output an XUnit report, useful for CI testing
--fail_level [warn|error]
-s, --skip TEXT                 Comma-separated list of lint tests to skip
                                (e.g. passing --skip 'citations,xml_order'
                                would skip linting of citations and best-
                                practice XML ordering.
--xsd / --no_xsd                Include tool XSD validation in linting
                                process.
-r, --recursive                 Recursively perform command for
                                subdirectories.
--urls                          Check validity of URLs in XML files
--doi                           Check validity of DOIs in XML files
--conda_requirements            Check tool requirements for availability in
                                best practice Conda channels.
--biocontainer, --biocontainers
                                Check best practice BioContainer namespaces
                                for a container definition applicable for this
                                tool.
--help                          Show this message and exit.
```

13.23 `list_alias` command

This section is auto-generated from the help text for the planemo command `list_alias`. This help message can be generated with `planemo list_alias --help`.

Usage:

```
planemo list_alias [OPTIONS]
```

Help

List aliases for a path or a workflow or dataset ID. Aliases are associated with a particular planemo profile.

Options:

```
--profile TEXT  Name of profile (created with the profile_create command) to
                use with this command.  [required]
--help          Show this message and exit.
```

13.24 `list_invocations` command

This section is auto-generated from the help text for the planemo command `list_invocations`. This help message can be generated with `planemo list_invocations --help`.

Usage:

```
planemo list_invocations [OPTIONS] WORKFLOW_IDENTIFIER
```

Help

Get a list of invocations for a particular workflow ID or alias.

Options:

```
--profile TEXT  Name of profile (created with the profile_create command) to
                use with this command.  [required]
--help          Show this message and exit.
```

13.25 `list_repos` command

This section is auto-generated from the help text for the planemo command `list_repos`. This help message can be generated with `planemo list_repos --help`.

Usage:

```
planemo list_repos [OPTIONS] PROJECT
```

Help

Find all shed repositories in one or more directories and output as yaml.

Currently, a shed repository is considered a directory with a .shed.yml file.

Options:

```
--exclude PATH                  Paths to exclude.
--exclude_from FILE             File of paths to exclude.
--changed_in_commit_range TEXT  Exclude paths unchanged in git commit range.
--chunk_count INTEGER           Split output into chunks of this many item and
                                print --chunk such group.
--chunk INTEGER                 When output is split into --chunk_count
                                groups, output the group 0-indexedby this
                                option.
--output TEXT                   File to output to, or - for standard output.
--help                          Show this message and exit.
```

13.26 `merge_test_reports` command

This section is auto-generated from the help text for the planemo command `merge_test_reports`. This help message can be generated with `planemo merge_test_reports --help`.

Usage:

```
planemo merge_test_reports [OPTIONS] INPUT_PATHS FILE_PATH
```

Help

Merge tool_test_output.json files from multiple runs. **Options**:

```
--help  Show this message and exit.
```

13.27 `mull` command

This section is auto-generated from the help text for the planemo command `mull`. This help message can be generated with `planemo mull --help`.

Usage:

```
planemo mull [OPTIONS] TOOL_PATH
```

Help

Build containers for specified tools.

Supplied tools will be inspected for referenced requirement packages. For each combination of requirements a "mulled" container will be built. Galaxy can automatically discover this container and subsequently use it to run or test the tool.

For this to work, the tool's requirements will need to be present in a known Conda channel such as bioconda (https://github.com/bioconda/bioconda-recipes). This can be verified by running `planemo lint --conda_requirements` on the target tool(s).

Options:

```
-r, --recursive                 Recursively perform command for
                                subdirectories.
--mulled_conda_version TEXT     Install a specific version of Conda before
                                running the command, by default the version
                                that comes with the continuumio miniconda3
```

(continues on next page)

(continued from previous page)

```
                                 image will be used under Linux and under Mac
                                 OS X Conda will be upgraded to to work around
                                 a bug in 4.2.
--mulled_namespace TEXT          Build a mulled image with the specified
                                 namespace - defaults to biocontainers. Galaxy
                                 currently only recognizes images with the
                                 namespace biocontainers.
--mulled_command TEXT            Mulled action to perform for targets - this
                                 defaults to 'build-and-test'.
--conda_channels, --conda_ensure_channels TEXT
                                 Ensure conda is configured with specified
                                 comma separated list of channels.
--help                           Show this message and exit.
```

13.28 `mulled_init` command

This section is auto-generated from the help text for the planemo command `mulled_init`. This help message can be generated with `planemo mulled_init --help`.

Usage:

```
planemo mulled_init [OPTIONS]
```

Help

Download and install involucro for mull command.

This will happen automatically when using the mull command, but this can be pre-installed in an environment using this command.

Options:

```
--mulled_conda_version TEXT  Install a specific version of Conda before
                             running the command, by default the version that
                             comes with the continuumio miniconda3 image will
                             be used under Linux and under Mac OS X Conda will
                             be upgraded to to work around a bug in 4.2.
--mulled_namespace TEXT      Build a mulled image with the specified namespace
                             - defaults to biocontainers. Galaxy currently
                             only recognizes images with the namespace
                             biocontainers.
--mulled_command TEXT        Mulled action to perform for targets - this
                             defaults to 'build-and-test'.
--help                       Show this message and exit.
```

13.29 normalize command

This section is auto-generated from the help text for the planemo command `normalize`. This help message can be generated with `planemo normalize --help`.

Usage:

```
planemo normalize [OPTIONS] TOOL_PATH
```

Help

Generate normalized tool XML from input.

This will break the formatting of your tool and is currently only intended for viewing macro expansions for for use with XSD validation (see https://github.com/JeanFred/Galaxy-XSD for instance). Please do not use the output as is - it frequently makes tool less readable not more.

The top-level blocks will be reordered and whitespace fixed according to the tool development best practices outlined on the Galaxy wiki.

```
% # Print normalized version of tool.
% planemo normalize tool.xml
<tool>
...
% # Print a variant of tool with all macros expanded out, useful for
% # debugging complex macros.
% planemo normalize --expand_macros tool.xml
<tool>
...
```

Options:

```
--expand_macros  Expand macros while normalizing tool XML - useful to see how
                 macros are evaluated.
--skip_reorder   Planemo will reorder top-level tool blocks according to tool
                 development best practices as part of this command, this flag
                 will disable that behavior.
--skip_reindent  Planemo will reindent the XML according to tool development
                 best practices as part of this command, this flag will
                 disable that behavior.
--help           Show this message and exit.
```

13.30 open command

This section is auto-generated from the help text for the planemo command `open`. This help message can be generated with `planemo open --help`.

Usage:

```
planemo open [OPTIONS] PATH
```

Help

Open latest Planemo test results in a web browser. **Options**:

```
--help  Show this message and exit.
```

13.31 profile_create command

This section is auto-generated from the help text for the planemo command `profile_create`. This help message can be generated with `planemo profile_create --help`.

Usage:

```
planemo profile_create [OPTIONS] PROFILE_NAME
```

Help

Create a profile. **Options**:

```
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.
```

(continues on next page)

(continued from previous page)

```
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
--galaxy_admin_key TEXT         Admin key to use with external Galaxy engine.
--help                          Show this message and exit.
```

13.32 profile_delete command

This section is auto-generated from the help text for the planemo command `profile_delete`. This help message can be generated with `planemo profile_delete --help`.

Usage:

```
planemo profile_delete [OPTIONS] PROFILE_NAME
```

Help

Delete a profile. **Options**:

```
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--help                          Show this message and exit.
```

13.33 `profile_list` command

This section is auto-generated from the help text for the planemo command `profile_list`. This help message can be generated with `planemo profile_list --help`.

Usage:

```
planemo profile_list [OPTIONS]
```

Help

List configured profile names. **Options**:

```
--help  Show this message and exit.
```

13.34 `project_init` command

This section is auto-generated from the help text for the planemo command `project_init`. This help message can be generated with `planemo project_init --help`.

Usage:

```
planemo project_init [OPTIONS] PROJECT
```

Help

(Experimental) Initialize a new tool project.

This is only a proof-of-concept demo right now.

Options:

```
--template TEXT
--help           Show this message and exit.
```

13.35 `pull_request` command

This section is auto-generated from the help text for the planemo command `pull_request`. This help message can be generated with `planemo pull_request --help`.

Usage:

```
planemo pull_request [OPTIONS] PROJECT
```

Help

Short-cut to quickly create a pull request for a relevant Github repo.

For instance, the following will clone, fork, and branch the tools-iuc repository to allow a subsequent pull request to fix a problem with bwa.

```
$ planemo clone --branch bwa-fix tools-iuc
$ cd tools-iuc
$ # Make changes.
```

(continues on next page)

(continued from previous page)

```
$ git add -p # Add desired changes.
$ git commit -m "Fix bwa problem."
$ planemo pull_request -m "Fix bwa problem."
```

These changes do require that a github access token is specified in ~/.planemo.yml. An access token can be generated by going to https://github.com/settings/tokens.

Options:

```
-m, --message TEXT  Message describing the pull request to create.
--help              Show this message and exit.
```

13.36 rerun command

This section is auto-generated from the help text for the planemo command `rerun`. This help message can be generated with `planemo rerun --help`.

Usage:

```
planemo rerun [OPTIONS] RERUNNABLE_IDS
```

Help

Planemo command for rerunning and remapping failed jobs on an external Galaxy server. Supply a list of history, invocation or job IDs, identifying the ID type using the –invocation, –history or –job flag, and all associated failed jobs will be rerun.

Please note: attempting to rerun non-remappable jobs will result in an exit code of 1. As jobs cannot be remapped more than once, running this command two or more times with the same history or job IDs will therefore return an exit code of 1. If avoiding this is important, you should specify the invocation ID instead if possible.

```
% planemo rerun --invocation / --history / --job RERUNNABLE_IDS
```

Options:

```
--profile TEXT          Name of profile (created with the profile_create
                        command) to use with this command.
--galaxy_url TEXT       Remote Galaxy URL to use with external Galaxy engine.
--galaxy_user_key TEXT  User key to use with external Galaxy engine.
--invocation            Rerun failed jobs associated by one or more invocation
                        IDs.
--history               Rerun failed jobs associated by one or more history
                        IDs.
--job                   Rerun failed jobs specified by one or more job IDs.
--help                  Show this message and exit.
```

13.37 run command

This section is auto-generated from the help text for the planemo command `run`. This help message can be generated with `planemo run --help`.

Usage:

```
planemo run [OPTIONS] RUNNABLE_PATH_OR_ID JOB_PATH
```

Help

Planemo command for running tools and jobs.

```
% planemo run cat1-tool.cwl cat-job.json
```

Options:

```
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
```

(continues on next page)

(continued from previous page)

```
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
```

(continues on next page)

(continued from previous page)

```
                                  When uploading files to Galaxy for tool or
                                  workflow tests or runs, upload multiple files
                                  simultaneously without waiting for the
                                  previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                  When uploading files to Galaxy for tool or
                                  workflow tests or runs, check that the history
                                  is in an 'ok' state before beginning tool or
                                  workflow execution.
--profile TEXT                    Name of profile (created with the
                                  profile_create command) to use with this
                                  command.
--postgres                        Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                  Type of database to use for profile - 'auto',
                                  'sqlite', 'postgres', and 'postgres_docker'
                                  are available options. Use postgres to use an
                                  existing postgres server you user can access
                                  without a password via the psql command. Use
                                  postgres_docker to have Planemo manage a
                                  docker container running postgres. Data with
                                  postgres_docker is not yet persisted past when
                                  you restart the docker container launched by
                                  Planemo so be careful with this option.
--postgres_psql_path TEXT         Name or or path to postgres client binary
                                  (psql).
--postgres_database_user TEXT     Postgres username for managed development
                                  databases.
--postgres_database_host TEXT     Postgres host name for managed development
                                  databases.
--postgres_database_port TEXT     Postgres port for managed development
                                  databases.
--file_path DIRECTORY             Location for files created by Galaxy (e.g.
                                  database/files).
--database_connection TEXT        Database connection string to use for Galaxy.
--shed_tool_conf TEXT             Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT             Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                  By default Planemo will configure Galaxy to
                                  run in single-user mode where there is just
                                  one user and this user is automatically logged
                                  it. Use --no_galaxy_single_user to prevent
                                  Galaxy from running this way.
--cwl                             Configure Galaxy for use with CWL tool. (this
                                  option is experimental and will be replaced
                                  when and if CWL support is merged into
                                  Galaxy).
--cwl_galaxy_root DIRECTORY       Root of development galaxy directory to
                                  execute command with (must be branch of Galaxy
                                  with CWL support, this option is experimental
                                  and will be replaced with --galaxy_root when
                                  and if CWL support is merged into Galaxy.
```

(continues on next page)

(continued from previous page)

```
--tags TEXT                     Comma-separated list of tags to add to the
                                created history.
--output_directory, --outdir DIRECTORY
                                Where to store outputs of a 'run' task.
--output_json FILE              Where to store JSON dictionary describing
                                outputs of a 'run' task.
--download_outputs / --no_download_outputs
                                After tool or workflow runs are complete,
                                download the output files to the location
                                specified by --output_directory.
--engine [galaxy|docker_galaxy|cwltool|toil|external_galaxy]
                                Select an engine to run or test artifacts such
                                as tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container or the CWL reference implementation
                                'cwltool' and 'toil' be selected.
--non_strict_cwl                Disable strict validation of CWL.
--no-container, --no_container  If cwltool engine is used, disable Docker
                                container usage.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--install_tool_dependencies / --no_install_tool_dependencies
                                Turn on installation of tool dependencies
                                using classic toolshed packages.
--install_resolver_dependencies / --no_install_resolver_dependencies
                                Skip installing tool dependencies through
                                resolver (e.g. conda).
--install_repository_dependencies / --no_install_repository_dependencies
                                Skip installing the repository dependencies.
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.
--galaxy_admin_key TEXT         Admin key to use with external Galaxy engine.
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
--history_name TEXT             Name to give a Galaxy history, if one is
                                created.
--no_wait                       After invoking a job or workflow, do not wait
                                for completion.
```

(continues on next page)

(continued from previous page)

```
--paste_test_data_paths / --no_paste_test_data_paths
                                By default Planemo will use or not use
                                Galaxy's path paste option to load test data
                                into a history based on the engine type it is
                                targeting. This can override the logic to
                                explicitly enable or disable path pasting.
--update_test_data              Update test-data directory with job outputs
                                (normally written to directory
                                --job_output_files if specified.)
--test_output PATH              Output test report (HTML - for humans)
                                defaults to tool_test_output.html.
--test_output_text PATH         Output test report (Basic text - for display
                                in CI)
--test_output_markdown PATH     Output test report (Markdown style - for
                                humans & computers)
--test_output_xunit PATH        Output test report (xunit style - for CI
                                systems
--test_output_junit PATH        Output test report (jUnit style - for CI
                                systems
--test_output_allure DIRECTORY  Output test allure2 framework resutls
--test_output_json PATH         Output test report (planemo json) defaults to
                                tool_test_output.json.
--job_output_files DIRECTORY    Write job outputs to specified directory.
--summary [none|minimal|compact]
                                Summary style printed to planemo's standard
                                output (see output reports for more complete
                                summary). Set to 'none' to disable completely.
--test_timeout INTEGER          Maximum runtime of a single test in seconds.
--help                          Show this message and exit.
```

13.38 serve command

This section is auto-generated from the help text for the planemo command `serve`. This help message can be generated with `planemo serve --help`.

Usage:

```
planemo serve [OPTIONS] TOOL_PATH
```

Help

Launch Galaxy instance with specified tools.

The Galaxy tool panel will include just the referenced tool or tools (by default all the tools in the current working directory) and the upload tool.

planemo will search parent directories to see if any is a Galaxy instance - but one can pick the Galaxy instance to use with the `--galaxy_root` option or force planemo to download a disposable instance with the `--install_galaxy` flag.

`planemo` will run the Galaxy instance in an existing virtualenv if one exists in a `.venv` directory in the specified `--galaxy_root`. Otherwise, the Galaxy instance will run in a clean virtualenv created in `/tmp`.

`planemo` uses temporarily generated config files and environment variables to attempt to shield this execution of Galaxy from manually launched runs against that same Galaxy root - but this may not be bullet proof yet, so please be careful and do not try this against a production Galaxy instance.

Options:

```
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
```

(continues on next page)

(continued from previous page)

```
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--non_strict_cwl                Disable strict validation of CWL.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
```

(continues on next page)

(continued from previous page)

```
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--daemon                        Serve Galaxy process as a daemon.
--pid_file FILE                 Location of pid file is executed with
                                --daemon.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
```

(continues on next page)

(continued from previous page)

```
--skip_client_build             Do not build Galaxy client when serving
                                Galaxy.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--cwl                           Configure Galaxy for use with CWL tool. (this
                                option is experimental and will be replaced
                                when and if CWL support is merged into
                                Galaxy).
--cwl_galaxy_root DIRECTORY     Root of development galaxy directory to
                                execute command with (must be branch of Galaxy
                                with CWL support, this option is experimental
                                and will be replaced with --galaxy_root when
                                and if CWL support is merged into Galaxy.
--help                          Show this message and exit.
```

13.39 `share_test` command

This section is auto-generated from the help text for the planemo command `share_test`. This help message can be generated with `planemo share_test --help`.

Usage:

```
planemo share_test [OPTIONS] FILE_PATH
```

Help

Publish JSON test results as sharable Gist.

This will upload the JSON test results to Github as a Gist and produce sharable URL.

The sharable URL can be used to share an HTML version of the report that can be easily embedded in pull requests or commit messages.

Requires a ~/.planemo.yml with a Github access token defined in a 'github' section of that configuration file. An access token can be generated by going to https://github.com/settings/tokens.

Options:

```
--help  Show this message and exit.
```

13.40 shed_build command

This section is auto-generated from the help text for the planemo command `shed_build`. This help message can be generated with `planemo shed_build --help`.

Usage:

```
planemo shed_build [OPTIONS] TOOL_PATH
```

Help

Create a Galaxy tool tarball.

This will use the .shed.yml file to prepare a tarball (which you could upload to the Tool Shed manually).

Options:

```
--help  Show this message and exit.
```

13.41 shed_create command

This section is auto-generated from the help text for the planemo command `shed_create`. This help message can be generated with `planemo shed_create --help`.

Usage:

```
planemo shed_create [OPTIONS] PROJECT
```

Help

Create a repository in a Galaxy Tool Shed.

This will read the settings from the `.shed.yml` file.

Options:

```
-r, --recursive          Recursively perform command for nested repository
                         directories.
--fail_fast              If multiple repositories are specified and an error
                         occurs stop immediately instead of processing
                         remaining repositories.
--owner TEXT             Tool Shed repository owner (username).
--name TEXT              Tool Shed repository name (defaults to the inferred
                         tool directory name).
--shed_email TEXT        E-mail for Tool Shed auth (required unless shed_key
                         is specified).
--shed_key TEXT          API key for Tool Shed access. An API key is required
                         unless e-mail and password is specified. This key
                         can be specified with either --shed_key or
                         --shed_key_from_env.
--shed_key_from_env TEXT Environment variable to read API key for Tool Shed
                         access from.
--shed_password TEXT     Password for Tool Shed auth (required unless
                         shed_key is specified).
-t, --shed_target TEXT   Tool Shed to target (this can be 'toolshed',
```

(continues on next page)

(continued from previous page)

```
                           'testtoolshed', 'local' (alias for
                           http://localhost:9009/), an arbitrary url or
                           mappings defined ~/.planemo.yml.
-m, --message TEXT         Commit message for tool shed upload.
--skip_upload              Skip upload contents as part of operation, only
                           update metadata.
--help                     Show this message and exit.
```

13.42 shed_diff command

This section is auto-generated from the help text for the planemo command `shed_diff`. This help message can be generated with `planemo shed_diff --help`.

Usage:

```
planemo shed_diff [OPTIONS] PROJECT
```

Help

diff between local repository and Tool Shed.

By default, this will produce a diff between this repository and what would be uploaded to the Tool Shed with the *shed_upload* command - but this command can be made to compare other combinations of repositories. Here are some examples

```
$ # diff for this repository and the main Tool Shed
$ planemo shed_diff
$ # diff for this repository and the test Tool Shed
$ planemo shed_diff --shed_target testtoolshed
$ # diff for the test Tool Shed and main Tool Shed
$ planemo shed_diff --shed_target_source testtoolshed
$ # diff for two an explicitly specified repositories (ignores
$ # current project's shed YAML file.)
$ planemo shed_diff --owner peterjc --name blast_rbh
    --shed_target_source testtoolshed
```

This command will return an exit code of:

- 0 if there are no detected differences.
- 1 if there are differences.
- 2 if the target repository doesn't exist.
- >200 if there are errors attempting to perform a diff.

Warning: `shed_diff` doesn't inspect repository metadata, this difference applies only to the file contents of files that would actually be uploaded to the repository.

Options:

```
-r, --recursive              Recursively perform command for nested repository
                             directories.
--fail_fast                  If multiple repositories are specified and an error
                             occurs stop immediately instead of processing
```

(continues on next page)

(continued from previous page)

```
                            remaining repositories.
--owner TEXT                Tool Shed repository owner (username).
--name TEXT                 Tool Shed repository name (defaults to the inferred
                            tool directory name).
--shed_email TEXT           E-mail for Tool Shed auth (required unless shed_key
                            is specified).
--shed_key TEXT             API key for Tool Shed access. An API key is
                            required unless e-mail and password is specified.
                            This key can be specified with either --shed_key or
                            --shed_key_from_env.
--shed_key_from_env TEXT    Environment variable to read API key for Tool Shed
                            access from.
--shed_password TEXT        Password for Tool Shed auth (required unless
                            shed_key is specified).
-t, --shed_target TEXT      Tool Shed to target (this can be 'toolshed',
                            'testtoolshed', 'local' (alias for
                            http://localhost:9009/), an arbitrary url or
                            mappings defined ~/.planemo.yml.
-o, --output PATH           Send diff output to specified file.
--shed_target_source TEXT   Source Tool Shed to diff against (will ignore local
                            project info specified). To compare the main Tool
                            Shed against the test, set this to testtoolshed.
--raw                       Do not attempt smart diff of XML to filter out
                            attributes populated by the Tool Shed.
--report_xunit PATH         Output an XUnit report, useful for CI testing
--help                      Show this message and exit.
```

13.43 shed_init command

This section is auto-generated from the help text for the planemo command shed_init. This help message can be generated with planemo shed_init --help.

Usage:

```
planemo shed_init [OPTIONS] PROJECT
```

Help

Bootstrap new Tool Shed .shed.yml file.

This Tool Shed configuration file is used by other planemo commands such as shed_lint, shed_create, shed_upload, and shed_diff to manage repositories in a Galaxy Tool Shed.

Options:

```
--from_workflow PATH            Attempt to generate repository dependencies
                                from specified workflow.
--description TEXT              Specify repository description for .shed.yml.
--long_description TEXT         Specify repository long_description for
                                .shed.yml.
--remote_repository_url TEXT    Specify repository remote_repository_url for
                                .shed.yml.
```

(continues on next page)

(continued from previous page)

```
--homepage_url TEXT             Specify repository homepage_url for .shed.yml.
--category [Assembly|Astronomy|ChIP-seq|Climate Analysis|CLIP-seq|Combinatorial␣
↪Selections|Computational chemistry|Constructive Solid Geometry|Convert Formats|Data␣
↪Export|Data Managers|Data Source|Ecology|Entomology|Epigenetics|Fasta␣
↪Manipulation|Fastq Manipulation|Flow Cytometry Analysis|Genome annotation|Genome␣
↪editing|Genome-Wide Association Study|Genomic Interval␣
↪Operations|GIS|Graphics|Imaging|InteractiveTools|Machine Learning|Materials␣
↪science|Metabolomics|Metagenomics|Micro-array Analysis|Molecular␣
↪Dynamics|Nanopore|Next Gen Mappers|NLP|Ontology␣
↪Manipulation|Phylogenetics|Proteomics|RNA|SAM|Sequence Analysis|Statistics|Structural␣
↪Materials Analysis|Synthetic Biology|Systems Biology|Text Manipulation|Tool Dependency␣
↪Packages|Tool Generators|Transcriptomics|Variant Analysis|Visualization|Web Services]
                                Specify repository category for .shed.yml (may
                                specify multiple).
--owner TEXT                    Tool Shed repository owner (username).
--name TEXT                     Tool Shed repository name (defaults to the
                                inferred tool directory name).
-f, --force                     Overwrite existing files if present.
--help                          Show this message and exit.
```

13.44 shed_lint command

This section is auto-generated from the help text for the planemo command `shed_lint`. This help message can be generated with `planemo shed_lint --help`.

Usage:

```
planemo shed_lint [OPTIONS] PROJECT
```

Help

Check Tool Shed repository for common issues.

With the `--tools` flag, this command lints actual Galaxy tools in addition to tool shed artifacts.

With the `--urls` flag, this command searches for `<package>$URL</package>` and download actions which specify URLs. Each of those are accessed individually. By default, this tool requests the first hundred or so bytes of each listed URL and validates that a 200 OK was received. In tool XML files, the `--urls` option checks through the help text for mentioned URLs and checks those.

Options:

```
-r, --recursive                 Recursively perform command for nested
                                repository directories.
--fail_fast                     If multiple repositories are specified and an
                                error occurs stop immediately instead of
                                processing remaining repositories.
--report_level [all|warn|error]
--fail_level [warn|error]
-s, --skip TEXT                 Comma-separated list of lint tests to skip
                                (e.g. passing --skip 'citations,xml_order'
                                would skip linting of citations and best-
                                practice XML ordering.
```

(continues on next page)

(continued from previous page)

```
--tools                         Lint tools discovered in the process of
                                linting repositories.
--xsd / --no_xsd                Include tool XSD validation in linting
                                process.
--ensure_metadata               Ensure .shed.yml files contain enough metadata
                                for each repository to allow automated
                                creation and/or updates.
--urls                          Check validity of URLs in XML files
--biocontainer, --biocontainers
                                Check best practice BioContainer namespaces
                                for a container definition applicable for this
                                tool.
--help                          Show this message and exit.
```

13.45 shed_serve command

This section is auto-generated from the help text for the planemo command `shed_serve`. This help message can be generated with `planemo shed_serve --help`.

Usage:

```
planemo shed_serve [OPTIONS] PROJECT
```

Help

Launch Galaxy with Tool Shed dependencies.

This command will start a Galaxy instance configured to target the specified shed, find published artifacts (tools and dependencies) corresponding to command-line arguments and `.shed.yml` file(s), install these artifacts, and serve a Galaxy instances that can be logged into and explored interactively.

Options:

```
-r, --recursive                 Recursively perform command for nested
                                repository directories.
--fail_fast                     If multiple repositories are specified and an
                                error occurs stop immediately instead of
                                processing remaining repositories.
--owner TEXT                    Tool Shed repository owner (username).
--name TEXT                     Tool Shed repository name (defaults to the
                                inferred tool directory name).
--shed_email TEXT               E-mail for Tool Shed auth (required unless
                                shed_key is specified).
--shed_key TEXT                 API key for Tool Shed access. An API key is
                                required unless e-mail and password is
                                specified. This key can be specified with
                                either --shed_key or --shed_key_from_env.
--shed_key_from_env TEXT        Environment variable to read API key for Tool
                                Shed access from.
--shed_password TEXT            Password for Tool Shed auth (required unless
                                shed_key is specified).
-t, --shed_target TEXT          Tool Shed to target (this can be 'toolshed',
```

(continues on next page)

(continued from previous page)

```
                                'testtoolshed', 'local' (alias for
                                http://localhost:9009/), an arbitrary url or
                                mappings defined ~/.planemo.yml.
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
```

(continues on next page)

(continued from previous page)

```
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--non_strict_cwl                Disable strict validation of CWL.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
```

(continues on next page)

(continued from previous page)

```
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--daemon                        Serve Galaxy process as a daemon.
--pid_file FILE                 Location of pid file is executed with
                                --daemon.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--skip_client_build             Do not build Galaxy client when serving
```

(continues on next page)

(continued from previous page)

```
                                Galaxy.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--skip_dependencies             Do not install shed dependencies as part of
                                repository installation.
--help                          Show this message and exit.
```

13.46 shed_test command

This section is auto-generated from the help text for the planemo command `shed_test`. This help message can be generated with `planemo shed_test --help`.

Usage:

```
planemo shed_test [OPTIONS] PROJECT
```

Help

Run tests of published shed artifacts.

This command will start a Galaxy instance configured to target the specified shed, find published artifacts (tools and dependencies) corresponding to command-line arguments and `.shed.yml` file(s), install these artifacts, and run the tool tests for these commands.

Options:

```
-r, --recursive                 Recursively perform command for nested
                                repository directories.
--fail_fast                     If multiple repositories are specified and an
                                error occurs stop immediately instead of
                                processing remaining repositories.
--owner TEXT                    Tool Shed repository owner (username).
--name TEXT                     Tool Shed repository name (defaults to the
                                inferred tool directory name).
--shed_email TEXT               E-mail for Tool Shed auth (required unless
                                shed_key is specified).
--shed_key TEXT                 API key for Tool Shed access. An API key is
                                required unless e-mail and password is
                                specified. This key can be specified with
                                either --shed_key or --shed_key_from_env.
--shed_key_from_env TEXT        Environment variable to read API key for Tool
                                Shed access from.
--shed_password TEXT            Password for Tool Shed auth (required unless
                                shed_key is specified).
-t, --shed_target TEXT          Tool Shed to target (this can be 'toolshed',
                                'testtoolshed', 'local' (alias for
                                http://localhost:9009/), an arbitrary url or
                                mappings defined ~/.planemo.yml.
```

(continues on next page)

(continued from previous page)

```
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
```

(continues on next page)

(continued from previous page)

```
                                make use of external reference data.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
```

(continues on next page)

(continued from previous page)

```
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--paste_test_data_paths / --no_paste_test_data_paths
                                By default Planemo will use or not use
                                Galaxy's path paste option to load test data
                                into a history based on the engine type it is
                                targeting. This can override the logic to
                                explicitly enable or disable path pasting.
--update_test_data              Update test-data directory with job outputs
                                (normally written to directory
                                --job_output_files if specified.)
--test_output PATH              Output test report (HTML - for humans)
                                defaults to tool_test_output.html.
--test_output_text PATH         Output test report (Basic text - for display
                                in CI)
--test_output_markdown PATH     Output test report (Markdown style - for
                                humans & computers)
--test_output_xunit PATH        Output test report (xunit style - for CI
                                systems
--test_output_junit PATH        Output test report (jUnit style - for CI
                                systems
--test_output_allure DIRECTORY  Output test allure2 framework resutls
--test_output_json PATH         Output test report (planemo json) defaults to
                                tool_test_output.json.
--job_output_files DIRECTORY    Write job outputs to specified directory.
--summary [none|minimal|compact]
                                Summary style printed to planemo's standard
                                output (see output reports for more complete
                                summary). Set to 'none' to disable completely.
--test_timeout INTEGER          Maximum runtime of a single test in seconds.
--skip_dependencies             Do not install shed dependencies as part of
                                repository installation.
```

(continues on next page)

(continued from previous page)

```
--help                          Show this message and exit.
```

13.47 shed_update command

This section is auto-generated from the help text for the planemo command `shed_update`. This help message can be generated with `planemo shed_update --help`.

Usage:

```
planemo shed_update [OPTIONS] PROJECT
```

Help

Update Tool Shed repository.

By default this command will update both repository metadata from `.shed.yml` and upload new contents from the repository directory.

```
% planemo shed_update
```

This will update the main tool shed with the repository defined by a `.shed.yml` file in the current working directory. Both the location of the `.shed.yml` and the tool shed to upload to can be easily configured. For instance, the following command can be used if `.shed.yml` if contained in `path/to/repo` and the desire is to update the test tool shed.

```
% planemo shed_update --shed_target testtoolshed path/to/repo
```

Another important option is `--check_diff` - this doesn't affect the updating of shed metadata but it will check for differences before uploading new contents to the tool shed. This may important because the tool shed will automatically populate certain attributes in tool shed artifact files (such as `tool_dependencies.xml`) and this may cause unwanted installable revisions to be created when there are no important changes.

The lower-level `shed_upload` command should be used instead if the repository doesn't define complete metadata in a `.shed.yml`.

Options:

```
--report_xunit PATH          Output an XUnit report, useful for CI testing
-r, --recursive              Recursively perform command for nested repository
                             directories.
--fail_fast                  If multiple repositories are specified and an
                             error occurs stop immediately instead of
                             processing remaining repositories.
--owner TEXT                 Tool Shed repository owner (username).
--name TEXT                  Tool Shed repository name (defaults to the
                             inferred tool directory name).
--shed_email TEXT            E-mail for Tool Shed auth (required unless
                             shed_key is specified).
--shed_key TEXT              API key for Tool Shed access. An API key is
                             required unless e-mail and password is specified.
                             This key can be specified with either --shed_key
                             or --shed_key_from_env.
--shed_key_from_env TEXT     Environment variable to read API key for Tool
                             Shed access from.
```

(continues on next page)

(continued from previous page)

```
--shed_password TEXT          Password for Tool Shed auth (required unless
                              shed_key is specified).
-t, --shed_target TEXT        Tool Shed to target (this can be 'toolshed',
                              'testtoolshed', 'local' (alias for
                              http://localhost:9009/), an arbitrary url or
                              mappings defined ~/.planemo.yml.
-m, --message TEXT            Commit message for tool shed upload.
--force_repository_creation   If a repository cannot be found for the specified
                              user/repo name pair, then automatically create
                              the repository in the toolshed.
--check_diff                  Skip uploading if the shed_diff detects there
                              would be no 'difference' (only attributes
                              populated by the shed would be updated.)
--skip_upload                 Skip upload contents as part of operation, only
                              update metadata.
--skip_metadata               Skip metadata update as part of operation, only
                              upload new contents.
--help                        Show this message and exit.
```

13.48 shed_upload command

This section is auto-generated from the help text for the planemo command `shed_upload`. This help message can be generated with `planemo shed_upload --help`.

Usage:

```
planemo shed_upload [OPTIONS] PROJECT
```

Help

Low-level command to upload tarballs.

Generally, `shed_update` should be used instead since it also updates both tool shed contents (via tar ball generation and upload) as well as metadata (to handle metadata changes in `.shed.yml` files).

```
% planemo shed_upload --tar_only  ~/
% tar -tzf shed_upload.tar.gz
test-data/blastdb.loc
...
tools/ncbi_blast_plus/tool_dependencies.xml
% tar -tzf shed_upload.tar.gz | wc -l
117
```

Options:

```
-r, --recursive               Recursively perform command for nested repository
                              directories.
--fail_fast                   If multiple repositories are specified and an
                              error occurs stop immediately instead of
                              processing remaining repositories.
--owner TEXT                  Tool Shed repository owner (username).
--name TEXT                   Tool Shed repository name (defaults to the
```

(continues on next page)

(continued from previous page)

```
                             inferred tool directory name).
--shed_email TEXT            E-mail for Tool Shed auth (required unless
                             shed_key is specified).
--shed_key TEXT              API key for Tool Shed access. An API key is
                             required unless e-mail and password is specified.
                             This key can be specified with either --shed_key
                             or --shed_key_from_env.
--shed_key_from_env TEXT     Environment variable to read API key for Tool
                             Shed access from.
--shed_password TEXT         Password for Tool Shed auth (required unless
                             shed_key is specified).
-t, --shed_target TEXT       Tool Shed to target (this can be 'toolshed',
                             'testtoolshed', 'local' (alias for
                             http://localhost:9009/), an arbitrary url or
                             mappings defined ~/.planemo.yml.
-m, --message TEXT           Commit message for tool shed upload.
--force_repository_creation  If a repository cannot be found for the specified
                             user/repo name pair, then automatically create
                             the repository in the toolshed.
--check_diff                 Skip uploading if the shed_diff detects there
                             would be no 'difference' (only attributes
                             populated by the shed would be updated.)
--tar_only                   Produce tar file for upload but do not publish to
                             a tool shed.
--tar FILE                   Specify a pre-existing tar file instead of
                             automatically building one as part of this
                             command.
--help                       Show this message and exit.
```

13.49 syntax command

This section is auto-generated from the help text for the planemo command `syntax`. This help message can be generated with `planemo syntax --help`.

Usage:

```
planemo syntax [OPTIONS]
```

Help

Open tool config syntax page in web browser. **Options**:

```
--help  Show this message and exit.
```

13.50 `test` command

This section is auto-generated from the help text for the planemo command `test`. This help message can be generated with `planemo test --help`.

Usage:

```
planemo test [OPTIONS] TOOL_PATH
```

Help

Run specified tool or workflow tests within Galaxy.

All referenced tools (by default all the tools in the current working directory) will be tested and the results quickly summarized.

To run these tests planemo needs a Galaxy instance to utilize, planemo will search parent directories to see if any is a Galaxy instance - but one can pick the Galaxy instance to use with the –galaxy_root option or force planemo to download a disposable instance with the `--install_galaxy` flag.

In addition to to quick summary printed to the console - various detailed output summaries can be configured. `tool_test_output.html` (settable via `--test_output`) will contain a human consumable HTML report describing the test run. A JSON file (settable via `--test_output_json` and defaulting to `tool_test_output.json`) will also be created. These files can can be disabled by passing in empty arguments or globally by setting the values `default_test_output` and/or `default_test_output_json` in `~/.planemo.yml` to `null`. For continuous integration testing a xUnit-style report can be configured using the `--test_output_xunit`.

planemo uses temporarily generated config files and environment variables to attempt to shield this execution of Galaxy from manually launched runs against that same Galaxy root - but this may not be bullet proof yet so please careful and do not try this against production Galaxy instances.

Options:

```
--failed                        Re-run only failed tests. This command will
                                read tool_test_output.json to determine which
                                tests failed so this file must have been
                                produced with the same set of tool ids
                                previously.
--polling_backoff INTEGER       Poll resources with an increasing interval
                                between requests. Useful when testing against
                                remote and/or production instances to limit
                                generated traffic.
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
```

(continues on next page)

(continued from previous page)

```
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
```

(continues on next page)

(continued from previous page)

```
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
```

(continues on next page)

(continued from previous page)

```
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--paste_test_data_paths / --no_paste_test_data_paths
                                By default Planemo will use or not use
                                Galaxy's path paste option to load test data
                                into a history based on the engine type it is
                                targeting. This can override the logic to
                                explicitly enable or disable path pasting.
--update_test_data              Update test-data directory with job outputs
                                (normally written to directory
                                --job_output_files if specified.)
--test_output PATH              Output test report (HTML - for humans)
                                defaults to tool_test_output.html.
--test_output_text PATH         Output test report (Basic text - for display
                                in CI)
--test_output_markdown PATH     Output test report (Markdown style - for
                                humans & computers)
--test_output_xunit PATH        Output test report (xunit style - for CI
                                systems
--test_output_junit PATH        Output test report (jUnit style - for CI
                                systems
--test_output_allure DIRECTORY  Output test allure2 framework resutls
--test_output_json PATH         Output test report (planemo json) defaults to
                                tool_test_output.json.
--job_output_files DIRECTORY    Write job outputs to specified directory.
--summary [none|minimal|compact]
                                Summary style printed to planemo's standard
                                output (see output reports for more complete
                                summary). Set to 'none' to disable completely.
--test_timeout INTEGER          Maximum runtime of a single test in seconds.
--engine [galaxy|docker_galaxy|cwltool|toil|external_galaxy]
                                Select an engine to run or test artifacts such
                                as tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container or the CWL reference implementation
                                'cwltool' and 'toil' be selected.
--non_strict_cwl                Disable strict validation of CWL.
--no-container, --no_container  If cwltool engine is used, disable Docker
                                container usage.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--ignore_dependency_problems    When installing shed repositories for
```

(continues on next page)

(continued from previous page)

```
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--install_tool_dependencies / --no_install_tool_dependencies
                                Turn on installation of tool dependencies
                                using classic toolshed packages.
--install_resolver_dependencies / --no_install_resolver_dependencies
                                Skip installing tool dependencies through
                                resolver (e.g. conda).
--install_repository_dependencies / --no_install_repository_dependencies
                                Skip installing the repository dependencies.
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.
--galaxy_admin_key TEXT         Admin key to use with external Galaxy engine.
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
--history_name TEXT             Name to give a Galaxy history, if one is
                                created.
--no_wait                       After invoking a job or workflow, do not wait
                                for completion.
--help                          Show this message and exit.
```

13.51 `test_reports` command

This section is auto-generated from the help text for the planemo command `test_reports`. This help message can be generated with `planemo test_reports --help`.

Usage:

```
planemo test_reports [OPTIONS] FILE_PATH
```

Help

Generate human readable tool test reports.

Creates reports in various formats (HTML, text, markdown) from the structured test output (tool_test_output.json).

Options:

```
--test_output PATH              Output test report (HTML - for humans)
                                defaults to tool_test_output.html.
--test_output_text PATH         Output test report (Basic text - for display
                                in CI)
--test_output_markdown PATH     Output test report (Markdown style - for
                                humans & computers)
--test_output_xunit PATH        Output test report (xunit style - for CI
```

(continues on next page)

(continued from previous page)

```
                                systems
--test_output_junit PATH        Output test report (jUnit style - for CI
                                systems
--test_output_allure DIRECTORY  Output test allure2 framework resutls
--help                          Show this message and exit.
```

13.52 tool_init command

This section is auto-generated from the help text for the planemo command `tool_init`. This help message can be generated with `planemo tool_init --help`.

Usage:

```
planemo tool_init [OPTIONS]
```

Help

Generate tool outline from given arguments. **Options**:

```
-i, --id TEXT                 Short identifier for new tool (no whitespace)
-f, --force                   Overwrite existing tool if present.
-t, --tool FILE               Output path for new tool (default is <id>.xml)
-n, --name TEXT               Name for new tool (user facing)
--version TEXT                Tool XML version.
-d, --description TEXT        Short description for new tool (user facing)
-c, --command TEXT            Command potentially including cheetah variables
                              ()(e.g. 'seqtk seq -A $input > $output')
--example_command TEXT        Example to command with paths to build Cheetah
                              template from (e.g. 'seqtk seq -A 2.fastq >
                              2.fasta'). Option cannot be used with --command,
                              should be used --example_input and --example_output.
--example_input TEXT          For use with --example_command, replace input file
                              (e.g. 2.fastq with a data input parameter).
--example_output TEXT         For use with --example_command, replace input file
                              (e.g. 2.fastq with a tool output).
--named_output TEXT           Create a named output for use with command block for
                              example specify --named_output=output1.bam and then
                              use '-o $output1' in your command block.
--input TEXT                  An input description (e.g. input.fasta)
--output TEXT                 An output location (e.g. output.bam), the Galaxy
                              datatype is inferred from the extension.
--help_text TEXT              Help text (reStructuredText)
--help_from_command TEXT      Auto populate help from supplied command.
--doi TEXT                    Supply a DOI (http://www.doi.org/) easing citation
                              of the tool for Galxy users (e.g. 10.1101/014043).
--cite_url TEXT               Supply a URL for citation.
--test_case                   For use with --example_commmand, generate a tool
                              test case from the supplied example.
--macros                      Generate a macros.xml for reuse across many tools.
--version_command TEXT        Command to print version (e.g. 'seqtk --version')
--requirement TEXT            Add a tool requirement package (e.g. 'seqtk' or
```

(continues on next page)

(continued from previous page)

```
                          'seqtk@1.68').
--container TEXT          Add a Docker image identifier for this tool.
--cwl                     Build a CWL tool instead of a Galaxy tool.
--autopygen TEXT          Option for automatic generation of tool file, from
                          python source code that uses argparse. Parameter is
                          a path to source file containing definition of the
                          parser
--help                    Show this message and exit.
```

13.53 training_fill_data_library command

This section is auto-generated from the help text for the planemo command `training_fill_data_library`. This help message can be generated with `planemo training_fill_data_library --help`.

Usage:

```
planemo training_fill_data_library [OPTIONS] TOOL_PATH
```

Help

Build training template from workflow. **Options**:

```
--topic_name TEXT     Name (directory name) of the topic to create or in which
                      a tutorial should be created or updates  [required]
--tutorial_name TEXT  Name (directory name) of the tutorial to modify
                      [required]
--zenodo_link TEXT    Zenodo URL with the input data
--help                Show this message and exit.
```

13.54 training_generate_from_wf command

This section is auto-generated from the help text for the planemo command `training_generate_from_wf`. This help message can be generated with `planemo training_generate_from_wf --help`.

Usage:

```
planemo training_generate_from_wf [OPTIONS] TOOL_PATH
```

Help

Create tutorial skeleton from workflow. **Options**:

```
--topic_name TEXT               Name (directory name) of the topic to create
                                or in which a tutorial should be created or
                                updates  [required]
--tutorial_name TEXT            Name (directory name) of the tutorial to
                                modify  [required]
--workflow PATH                 Workflow of the tutorial (locally)
--galaxy_url TEXT               URL of a Galaxy instance with the workflow
--galaxy_api_key TEXT           API key on the Galaxy instance with the
```

(continues on next page)

(continued from previous page)

```
                                workflow
--workflow_id TEXT              ID of the workflow on the Galaxy instance
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
```

(continues on next page)

(continued from previous page)

```
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--non_strict_cwl                Disable strict validation of CWL.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
```

(continues on next page)

(continued from previous page)

```
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--daemon                        Serve Galaxy process as a daemon.
--pid_file FILE                 Location of pid file is executed with
                                --daemon.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--skip_client_build             Do not build Galaxy client when serving
                                Galaxy.
```

(continues on next page)

(continued from previous page)

```
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--help                          Show this message and exit.
```

13.55 `training_init` command

This section is auto-generated from the help text for the planemo command `training_init`. This help message can be generated with `planemo training_init --help`.

Usage:

```
planemo training_init [OPTIONS] TOOL_PATH
```

Help

Build training template from workflow. **Options**:

```
--topic_name TEXT               Name (directory name) of the topic to create
                                or in which a tutorial should be created or
                                updates  [required]
--topic_title TEXT              Title of the topic to create
--topic_summary TEXT            Summary of the topic
--topic_target [use|admin-dev|instructors]
                                Target audience for the topic
--tutorial_name TEXT            Name (directory name) of the tutorial to
                                create or to modify
--tutorial_title TEXT           Title of the tutorial
--hands_on                      Add hands-on for the new tutorial
--slides                        Add slides for the new tutorial
--workflow PATH                 Workflow of the tutorial (locally)
--galaxy_url TEXT               URL of a Galaxy instance with the workflow
--galaxy_api_key TEXT           API key on the Galaxy instance with the
                                workflow
--workflow_id TEXT              ID of the workflow on the Galaxy instance
--zenodo_link TEXT              Zenodo URL with the input data
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
```

(continues on next page)

(continued from previous page)

```
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
```

(continues on next page)

(continued from previous page)

```
--non_strict_cwl                Disable strict validation of CWL.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
```

(continues on next page)

(continued from previous page)

```
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--daemon                        Serve Galaxy process as a daemon.
--pid_file FILE                 Location of pid file is executed with
                                --daemon.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--skip_client_build             Do not build Galaxy client when serving
                                Galaxy.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--help                          Show this message and exit.
```

13.56 `upload_data` command

This section is auto-generated from the help text for the planemo command `upload_data`. This help message can be generated with `planemo upload_data --help`.

Usage:

```
planemo upload_data [OPTIONS] RUNNABLE_PATH_OR_ID JOB_PATH NEW_JOB_PATH
```

Help

Planemo command for uploading data to an external Galaxy server.

```
% planemo upload_data wf.ga wf-job.yml new-wf-job.yml --profile profile
```

Running this subcommand requires a workflow file or identifier and a job file, just as for `planemo run`. In addition, a third argument is required, for the location of a new job file which Planemo will write. The data will be uploaded to the specified external Galaxy server and the job file will be recreated at the specified location, with all instances of `path` or `location` for input datasets and collections replaced by `galaxy_id`. The new job file can then be used to run the workflow separately from the already completed data upload.

Options:

```
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
```

(continues on next page)

(continued from previous page)

```
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
```

(continues on next page)

(continued from previous page)

```
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--tags TEXT                     Comma-separated list of tags to add to the
                                created history.
--help                          Show this message and exit.
```

13.57 workflow_convert command

This section is auto-generated from the help text for the planemo command `workflow_convert`. This help message can be generated with `planemo workflow_convert --help`.

Usage:

```
planemo workflow_convert [OPTIONS] WORKFLOW_PATH_OR_ID
```

Help

Convert Format 2 workflows to native Galaxy workflows, and vice-versa. **Options**:

```
-f, --force                     Overwrite existing files if present.
-o, --output FILE
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
```

(continues on next page)

(continued from previous page)

```
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--non_strict_cwl                Disable strict validation of CWL.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
```

(continues on next page)

(continued from previous page)

```
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
```

(continues on next page)

(continued from previous page)

```
                                Galaxy from running this way.
--daemon                        Serve Galaxy process as a daemon.
--pid_file FILE                 Location of pid file is executed with
                                --daemon.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--skip_client_build             Do not build Galaxy client when serving
                                Galaxy.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--help                          Show this message and exit.
```

13.58 `workflow_edit` command

This section is auto-generated from the help text for the planemo command `workflow_edit`. This help message can be generated with `planemo workflow_edit --help`.

Usage:

```
planemo workflow_edit [OPTIONS] WORKFLOW_PATH_OR_ID
```

Help

Open a synchronized Galaxy workflow editor. **Options**:

```
--galaxy_root DIRECTORY         Root of development galaxy directory to
                                execute command with.
--galaxy_python_version [3|3.7|3.8|3.9|3.10|3.11]
                                Python version to start Galaxy under
--extra_tools PATH              Extra tool sources to include in Galaxy's tool
                                panel (file or directory). These will not be
                                linted/tested/etc... but they will be
                                available to workflows and for interactive
                                use.
--install_galaxy                Download and configure a disposable copy of
                                Galaxy from github.
--galaxy_branch TEXT            Branch of Galaxy to target (defaults to
                                master) if a Galaxy root isn't specified.
--galaxy_source TEXT            Git source of Galaxy to target (defaults to
                                the official galaxyproject github source if a
                                Galaxy root isn't specified.
--skip_venv                     Do not create or source a virtualenv
                                environment for Galaxy, this should be used to
                                preserve an externally configured virtual
```

(continues on next page)

(continued from previous page)

```
                                environment or conda environment.
--no_cache_galaxy               Skip caching of Galaxy source and dependencies
                                obtained with --install_galaxy. Not caching
                                this results in faster downloads (no git) - so
                                is better on throw away instances such with
                                TravisCI.
--no_cleanup                    Do not cleanup temp files created for and by
                                Galaxy.
--galaxy_email TEXT             E-mail address to use when launching single-
                                user Galaxy server.
--docker / --no_docker          Run Galaxy tools in Docker if enabled.
--docker_cmd TEXT               Command used to launch docker (defaults to
                                docker).
--docker_sudo / --no_docker_sudo
                                Flag to use sudo when running docker.
--docker_host TEXT              Docker host to target when executing docker
                                commands (defaults to localhost).
--docker_sudo_cmd TEXT          sudo command to use when --docker_sudo is
                                enabled (defaults to sudo).
--docker_run_extra_arguments TEXT
                                Extra arguments to pass to docker run.
--mulled_containers, --biocontainers
                                Test tools against mulled containers (forces
                                --docker). Disables conda resolution unless
                                any conda option has been set explicitly.
--galaxy_startup_timeout INTEGER RANGE
                                Wait for galaxy to start before assuming
                                Galaxy did not start.  [x>=1]
--job_config_file FILE          Job configuration file for Galaxy to target.
--tool_dependency_dir DIRECTORY
                                Tool dependency dir for Galaxy to target.
--tool_data_path DIRECTORY      Directory where data used by tools is located.
                                Required if tests are run in docker and should
                                make use of external reference data.
--port INTEGER                  Port to serve Galaxy on (default is 9090).
--host TEXT                     Host to bind Galaxy to. Default is 127.0.0.1
                                that is restricted to localhost connections
                                for security reasons set to 0.0.0.0 to bind
                                Galaxy to all ports including potentially
                                publicly accessible ones.
--engine [galaxy|docker_galaxy|external_galaxy]
                                Select an engine to serve artifacts such as
                                tools and workflows. Defaults to a local
                                Galaxy, but running Galaxy within a Docker
                                container.
--non_strict_cwl                Disable strict validation of CWL.
--docker_galaxy_image TEXT      Docker image identifier for docker-galaxy-
                                flavor used if engine type is specified as
                                ``docker-galaxy``. Defaults to
                                quay.io/bgruening/galaxy.
--docker_extra_volume PATH      Extra path to mount if --engine docker or
                                `--biocontainers` or `--docker`.
```

(continues on next page)

(continued from previous page)

```
--test_data DIRECTORY           test-data directory to for specified tool(s).
--tool_data_table PATH          tool_data_table_conf.xml file to for specified
                                tool(s).
--dependency_resolvers_config_file FILE
                                Dependency resolver configuration for Galaxy
                                to target.
--brew_dependency_resolution    Configure Galaxy to use plain brew dependency
                                resolution.
--shed_dependency_resolution    Configure Galaxy to use brewed Tool Shed
                                dependency resolution.
--no_dependency_resolution      Configure Galaxy with no dependency resolvers.
--conda_prefix DIRECTORY        Conda prefix to use for conda dependency
                                commands.
--conda_exec FILE               Location of conda executable.
--conda_channels, --conda_ensure_channels TEXT
                                Ensure conda is configured with specified
                                comma separated list of channels.
--conda_use_local               Use locally built packages while building
                                Conda environments.
--conda_dependency_resolution   Configure Galaxy to use only conda for
                                dependency resolution.
--conda_auto_install / --no_conda_auto_install
                                Conda dependency resolution for Galaxy will
                                attempt to install requested but missing
                                packages.
--conda_auto_init / --no_conda_auto_init
                                Conda dependency resolution for Galaxy will
                                auto install conda itself using miniconda if
                                not availabe on conda_prefix.
--simultaneous_uploads / --no_simultaneous_uploads
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, upload multiple files
                                simultaneously without waiting for the
                                previous file upload to complete.
--check_uploads_ok / --no_check_uploads_ok
                                When uploading files to Galaxy for tool or
                                workflow tests or runs, check that the history
                                is in an 'ok' state before beginning tool or
                                workflow execution.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--postgres                      Use postgres database type.
--database_type [postgres|postgres_docker|sqlite|auto]
                                Type of database to use for profile - 'auto',
                                'sqlite', 'postgres', and 'postgres_docker'
                                are available options. Use postgres to use an
                                existing postgres server you user can access
                                without a password via the psql command. Use
                                postgres_docker to have Planemo manage a
                                docker container running postgres. Data with
                                postgres_docker is not yet persisted past when
```

(continues on next page)

(continued from previous page)

```
                                you restart the docker container launched by
                                Planemo so be careful with this option.
--postgres_psql_path TEXT       Name or or path to postgres client binary
                                (psql).
--postgres_database_user TEXT   Postgres username for managed development
                                databases.
--postgres_database_host TEXT   Postgres host name for managed development
                                databases.
--postgres_database_port TEXT   Postgres port for managed development
                                databases.
--file_path DIRECTORY           Location for files created by Galaxy (e.g.
                                database/files).
--database_connection TEXT      Database connection string to use for Galaxy.
--shed_tool_conf TEXT           Location of shed tools conf file for Galaxy.
--shed_tool_path TEXT           Location of shed tools directory for Galaxy.
--galaxy_single_user / --no_galaxy_single_user
                                By default Planemo will configure Galaxy to
                                run in single-user mode where there is just
                                one user and this user is automatically logged
                                it. Use --no_galaxy_single_user to prevent
                                Galaxy from running this way.
--daemon                        Serve Galaxy process as a daemon.
--pid_file FILE                 Location of pid file is executed with
                                --daemon.
--ignore_dependency_problems    When installing shed repositories for
                                workflows, ignore dependency issues. These
                                likely indicate a problem but in some cases
                                may not prevent a workflow from successfully
                                executing.
--skip_client_build             Do not build Galaxy client when serving
                                Galaxy.
--shed_install / --no_shed_install
                                By default Planemo will attempt to install
                                repositories needed for workflow testing. This
                                may not be appropriate for production servers
                                and so this can disabled by calling planemo
                                with --no_shed_install.
--help                          Show this message and exit.
```

13.59 `workflow_job_init` command

This section is auto-generated from the help text for the planemo command `workflow_job_init`. This help message can be generated with `planemo workflow_job_init --help`.

Usage:

```
planemo workflow_job_init [OPTIONS] WORKFLOW_PATH_OR_ID
```

Help

Initialize a Galaxy workflow job description for supplied workflow.

Be sure to your lint your workflow with `workflow_lint` before calling this to ensure inputs and outputs comply with best practices that make workflow testing easier.

Jobs can be run with the planemo run command (`planemo run workflow.ga job.yml`). Planemo run works with Galaxy tools and CWL artifacts (both tools and workflows) as well so this command may be renamed to to job_init at something along those lines at some point.

Options:

```
-f, --force                     Overwrite existing files if present.
-o, --output FILE
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
--from_invocation / --from_uri  Build a workflow test or job description from
                                an invocation ID run on an external Galaxy.A
                                Galaxy URL and API key must also be specified.
                                This allows test data to be downloadedand
                                inputs and parameters defined automatically.
                                Alternatively, the default is to build
                                thedescriptions from a provided workflow URI.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--help                          Show this message and exit.
```

13.60 `workflow_lint` command

This section is auto-generated from the help text for the planemo command `workflow_lint`. This help message can be generated with `planemo workflow_lint --help`.

Usage:

```
planemo workflow_lint [OPTIONS] TARGET
```

Help

Check workflows for syntax errors and best practices. **Options**:

```
--report_level [all|warn|error]
--report_xunit PATH             Output an XUnit report, useful for CI testing
--fail_level [warn|error]
-s, --skip TEXT                 Comma-separated list of lint tests to skip
                                (e.g. passing --skip 'citations,xml_order'
                                would skip linting of citations and best-
                                practice XML ordering.
--help                          Show this message and exit.
```

13.61 workflow_test_init command

This section is auto-generated from the help text for the planemo command `workflow_test_init`. This help message can be generated with `planemo workflow_test_init --help`.

Usage:

```
planemo workflow_test_init [OPTIONS] WORKFLOW_PATH_OR_ID
```

Help

Initialize a Galaxy workflow test description for supplied workflow.

Be sure to lint your workflow with `workflow_lint` before calling this to ensure inputs and outputs comply with best practices that make workflow testing easier.

Options:

```
-f, --force                     Overwrite existing files if present.
-o, --output FILE
--split_test / --no_split_test  Write workflow job and test definitions to
                                separate files.
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
--from_invocation / --from_uri  Build a workflow test or job description from
                                an invocation ID run on an external Galaxy.A
                                Galaxy URL and API key must also be specified.
                                This allows test data to be downloadedand
                                inputs and parameters defined automatically.
                                Alternatively, the default is to build
                                thedescriptions from a provided workflow URI.
--profile TEXT                  Name of profile (created with the
                                profile_create command) to use with this
                                command.
--help                          Show this message and exit.
```

13.62 workflow_test_on_invocation command

This section is auto-generated from the help text for the planemo command `workflow_test_on_invocation`. This help message can be generated with `planemo workflow_test_on_invocation --help`.

Usage:

```
planemo workflow_test_on_invocation [OPTIONS] TEST.YML INVOCATION_ID
```

Help

Run defined tests against existing workflow invocation. **Options**:

```
--galaxy_url TEXT               Remote Galaxy URL to use with external Galaxy
                                engine.  [required]
--galaxy_user_key TEXT          User key to use with external Galaxy engine.
                                [required]
```

(continues on next page)

(continued from previous page)

```
--test_index INTEGER            Select which test to check. Counting starts at
                                1
--update_test_data              Update test-data directory with job outputs
                                (normally written to directory
                                --job_output_files if specified.)
--test_output PATH              Output test report (HTML - for humans)
                                defaults to tool_test_output.html.
--test_output_text PATH         Output test report (Basic text - for display
                                in CI)
--test_output_markdown PATH     Output test report (Markdown style - for
                                humans & computers)
--test_output_xunit PATH        Output test report (xunit style - for CI
                                systems
--test_output_junit PATH        Output test report (jUnit style - for CI
                                systems
--test_output_allure DIRECTORY  Output test allure2 framework resutls
--test_output_json PATH         Output test report (planemo json) defaults to
                                tool_test_output.json.
--job_output_files DIRECTORY    Write job outputs to specified directory.
--summary [none|minimal|compact]
                                Summary style printed to planemo's standard
                                output (see output reports for more complete
                                summary). Set to 'none' to disable completely.
--test_timeout INTEGER          Maximum runtime of a single test in seconds.
--help                          Show this message and exit.
```

13.63 workflow_upload command

This section is auto-generated from the help text for the planemo command `workflow_upload`. This help message can be generated with `planemo workflow_upload --help`.

Usage:

```
planemo workflow_upload [OPTIONS] TARGET
```

Help

Upload workflows to github organization. **Options**:

```
--namespace TEXT      Organization or username under which to create or update
                      workflow repository. Must be a valid github username or
                      organization
--github_branch TEXT  GitHub branch to use for the action. Default is main.
--dry_run             Don't execute action, show preview of action.
--help                Show this message and exit.
```

CHAPTER

FOURTEEN

FAQS

Below are a list of frequently asked questions (and answers) for users new to Planemo.

14.1 What is Planemo?

Planemo is a command-line application which is primarily used for developing Galaxy and CWL tools, workflows and training materials. It's also used for deploying the tools and workflows once developed (tools to the Galaxy ToolShed; workflows to the Intergalactic Workflow Commission) and executing Galaxy-based analyses via the command-line, if you prefer or need to use that rather than Galaxy's graphical interface.

14.2 Is Planemo the right tool for me?

Yes, if you are comfortable using the command-line and one of the following is true:

- You want to use a piece of scientific software not currently available in Galaxy and therefore need to wrap it yourself.
- You want to develop a workflow from individual Galaxy tools and submit it to a community repository such as the Intergalactic Workflow Commission.
- You want to develop CWL tools or workflows.
- You're writing a tutorial for the Galaxy Training Network and would like to use a pre-prepared template.
- You want to run Galaxy workflows from the command line.

14.3 What is the difference between Galaxy and CWL? Which should I use?

Galaxy is a web-based scientific analysis platform which provides access to scientific software via a graphical interface. It is also a scientific workflow management system and allows chaining multiple individual tools together to form a complex pipeline, which can be executed as a single tool. If you place a high value on a graphical interface, usability and developing flexible, reusable tools which can easily be adapted for other purposes, Galaxy is the best choice for you.

CWL (Common Workflow Language) is a workflow specification, rather than a workflow management system. It aims to allow the textual description of scientific pipelines in a way that is independent of the particular workflow manager used; if this is important to you, using CWL would be an excellent idea. CWL workflows are supported by a variety of workflow managers, including Toil, cwltool, Arvados and Galaxy.

Both Galaxy and CWL place a high value on community, open-source code and FAIR data analysis.

14.4 Can I develop Galaxy tools without Planemo?

Yes, you can, but it is more complicated to do so and will take more time. Planemo encourages best practices in software development and follows guidelines agreed on by the Galaxy community, particularly test-driven development.

14.5 Where can I learn to write tools?

We have a tutorial in this documentation and another incorporated into the Galaxy Training Network, which should get you off to a good start.

If you use the Visual Studio Code editor, we recommend using Planemo in conjunction with the Galaxy Language Server, which encourages development best practices and should greatly improve the speed at which you can develop new Galaxy tools.

14.6 How do I set up deployment to the ToolShed with Planemo?

We recommend using continuous integration (e.g. GitHub Actions) to achieve this and provide a template repository which you can use to set up a GitHub repository which automatically deploys tools to the ToolShed when a pull request is merged.

If you don't want to handle tool deployment yourself, you can also submit your tool wrappers to a community repository such as the one provided by the IUC. This also has the advantage that your wrappers will be reviewed by experienced Galaxy users and developers.

14.7 How can I automate workflow execution with Planemo?

Have a look at this section of the documentation, or this training provided by the Galaxy Training Network.

14.8 How can I contribute to the project?

We would love to see new contributions to Planemo! Please have a look here to see the different ways you can help.

layout: true class: inverse, middle

---

class: title

CHAPTER

FIFTEEN

GALAXY TOOL FRAMEWORK CHANGES

15.1 John Chilton

This document describes changes to Galaxy's tooling framework over recent releases.

15.1.1 16.04

Full Galaxy changelog.

Tool Profile Version (PR #1688)

Tools may (and should) now declare a `profile` version (e.g. `<tool profile="16.04" ...>`).

This allows Galaxy to fire a warning if a tool uses features too new for the current version and allows us to migrate away from some undesirable default behaviors that were required for backward compatiblity.

`set -e` by default (d020522)

From the IUC best practices documentation:

> *"If you need to execute more than one shell command, concatenate them with a double ampersand (&&), so that an error in a command will abort the execution of the following ones."*

The job script generated with profile `16.04+` tools will include a `#set -e` statement causing this behavior by default.

Older-style tools can enable this behavior by setting `strict="true"` on the tool `command` XML block.

Using Exit Codes for Error Detection (b92074e)

Previously the default behavior was for Galaxy to ignore exit codes and declare a tool in error if issued any output on standard error. This was a regretable default behavior and so all tools were encouraged to declare special XML blocks to force the use of exit codes.

For any tool that declares a profile version of `16.04` or greater, the default is now just to use exit codes for error detection.

Unrobust Features Removed (b92074e)

A few tool features have ben removed from tools that declare a version of `16.04` or newer.

- The `interepreter=` attribute on `command` blocks has been eliminated. Please use `$__tool_directory__` from within the tool instead.
- `format="input"` on output datasets has been eliminated, please use `format_source=` to specify an exact input to derive datatype from.
- Disables extra output file discovery by default, tools must explicitly describe the outputs to collect with `discover_dataset` tags.
- Tools require a `version` attribute - previously an implicit default to `1.0.0` would be used.
- `$param_file` has been eliminated.

Clean Working Directories

Previously, Galaxy would fill tool working directories with files related to metadata and job metric collection. Tools will no longer be executed in the same directory as these files.

This applies to all tools not just profile `16.04+` tools.

15.1.2 16.01

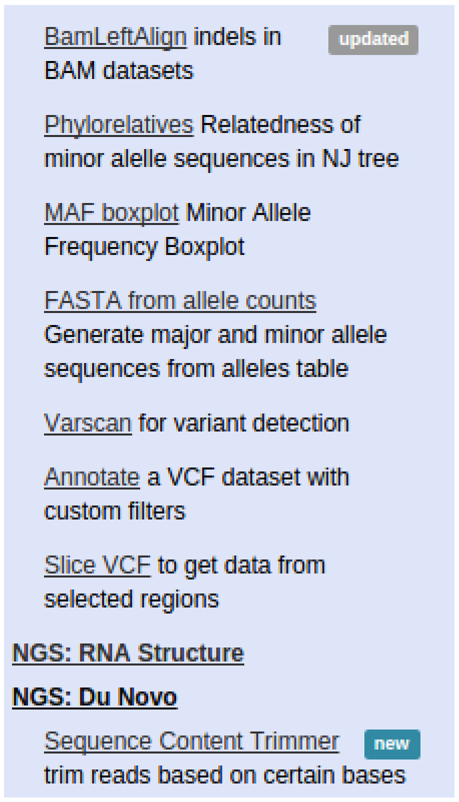
Full Galaxy changelog.

Conda Dependency Resolution (PR #1345)

```xml
<tool>
  ...
  <requirements>
    <requirement type="package" version="0.11.4">FastQC</requirement>
  </requirements>
  ...
</tool>
```

- Dependency resolvers tell Galaxy how to translate requirements into jobs.
- The Conda dependency resolver forces Galaxy to create a conda environment for the job with `FastQC` at version `0.11.4` installed.
- Only dependency resolver that can be installed at runtime - great for Docker images, heterogeneous clusters, and testing tools.
- Links Conda and BioConda

ToolBox Enhancements - Labels (PR #1012)



BamLeftAlign indels in BAM datasets updated

Phylorelatives Relatedness of minor alelle sequences in NJ tree

MAF boxplot Minor Allele Frequency Boxplot

FASTA from allele counts Generate major and minor allele sequences from alleles table

Varscan for variant detection

Annotate a VCF dataset with custom filters

Slice VCF to get data from selected regions

NGS: RNA Structure

NGS: Du Novo

Sequence Content Trimmer new trim reads based on certain bases

---

ToolBox Enhancements - Monitoring (PR #1398)

- The Galaxy toolbox can be reloaded from the Admin interface.
- Tool conf files (e.g. `tool_conf.xml`) can be monitored and automatically reloaded by Galaxy.
- Tool conf files can now be specified as YAML (or JSON).

Process Inputs as JSON (PR #1405)

```
    <command>python "$__tool_directory/script.py" "$json_inputs"</command>
    <configfiles>
        <inputs name="json_inputs" />
    </configfiles>
```

This will produce a file referenced as `$json_inputs` that contains a nested JSON structure corresponding to the tools inputs. Of limitted utility for simple command-line tools - but complex tools with many repeats, conditional, and nesting could potentially benefit from this.

For instance, the JBrowse tool generates a complex JSON data structure using a `configfile` inside the XML. This is a much more portable way to deal with that.

Collections

- `data_collection` tool parameters (`params`) can now specify multiple `collection_types` for consumption (PR #1308).
  - This mirrors the `format` attribute which allows a comma-separated list of potential format types.
- Multiple collections can now be supplied to a `multiple="true"` data parameter (PR #805).
- Output collections can specify a `type_source` attribute (again mirroring `format_source`) (PR #1153).

15.1.3 15.10

Full Galaxy changelog.

Collections

- Tools may now produce explicit nested outputs PR #538. This enhances the `discover_dataset` XML tag to allow this.
- Allow certain `output` actions on collections. PR #544.
- Allow `discover_dataset` tags to use `format` instead of `ext` when referring to datatype extensions/formats.
- Allow `min`/`max` attributes on multiple data input parameters PR #765.

Whitelist Tools that Generate HTML (PR #510)

Galaxy now contains a plain text file that contains a list of tools whose output can be trusted when rendering HTML.

15.1.4 15.07

Full Galaxy changelog.

Parameterized XML Macros (PR #362)

Macros now allow defining tokens to be consumed as XML attributes. For instance, the following definition

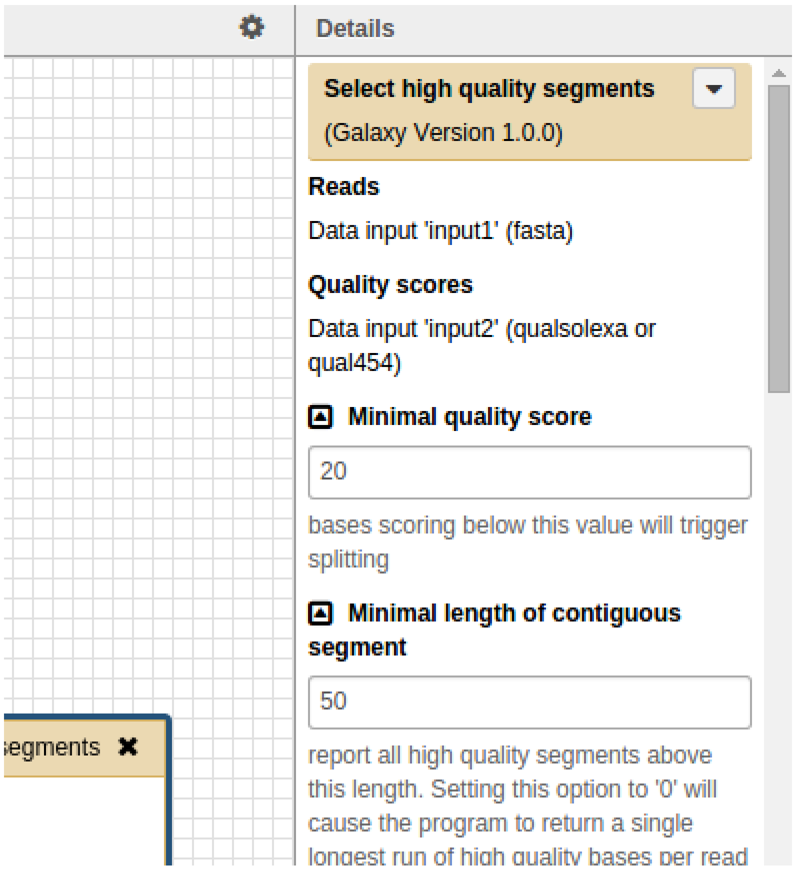
```
<tool>
    <expand macro="inputs" foo="hello" />
    <expand macro="inputs" foo="world" />
    <expand macro="inputs" />
    <macros>
        <xml name="inputs" token_foo="the_default">
            <inputs>@FOO@</inputs>
        </xml>
    </macros>
</tool>
```

would expand out as

```
<tool>
    <inputs>hello</inputs>
    <inputs>world</inputs>
    <inputs>the_default</inputs>
</tool>
```

Tool Form

The workflow editor was updated to the use Galaxy's newer frontend tool form.



Details

Select high quality segments (Galaxy Version 1.0.0)

Reads

Data input 'input1' (fasta)

Quality scores

Data input 'input2' (qualsolexa or qual454)

Minimal quality score

20

bases scoring below this value will trigger splitting

Minimal length of contiguous segment

50

report all high quality segments above this length. Setting this option to '0' will cause the program to return a single longest run of high quality bases per read

Environment Variables (PR #395)

Tools may now use `inputs` to define environment variables that will be set during tool execution. The new `environment_variables` XML block is used to define this.

```
<command>
  echo "\$INTVAR"  >  $out_file1;
  echo "\$FORTEST" >> $out_file1;
</command>
<environment_variables>
  <environment_variable name="INTVAR">$inttest</environment_variable>
  <environment_variable name="FORTEST">#for i in ['m', 'o', 'o']#$i#end for#</
↪environment_variable>
  </environment_variables>
  ...
```

Test tool demonstrating the use of the `environment_variables` tag.

Collections

- Explicit output collections can now be used in workflows. (PR #311)
- The `filter` tag has been implemented for output dataset collections (PR #455. See the example tool output_collection_filter.xml.

CHAPTER SIXTEEN

CODE OF CONDUCT

As part of the Galaxy Community, this project is committed to providing a welcoming and harassment-free experience for everyone. We therefore expect participants to abide by our Code of Conduct, which can be found at:

https://galaxyproject.org/community/coc/

CHAPTER

SEVENTEEN

CONTRIBUTING

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

17.1 Types of Contributions

17.1.1 Report Bugs

Report bugs at https://github.com/galaxyproject/planemo/issues.

If you are reporting a bug, please include:

- Your operating system name and version, versions of other relevant software such as Galaxy or Docker.
- Links to relevant tools.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

17.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

17.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" is open to whoever wants to implement it.

17.1.4 Write Documentation

Planemo could always use more documentation, whether as part of the official Planemo docs, in docstrings, or even on the web in blog posts, articles, and such.

17.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/galaxyproject/planemo/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- This will hopefully become a community-driven project and contributions are welcome :)

17.2 Get Started!

Ready to contribute? Here's how to set up *planemo* for local development.

1. Fork the *planemo* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/planemo.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ make setup-venv
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass `flake8` and the tests

   ```
   $ make lint
   $ make test
   ```

   If the modification doesn't affect code that configures and runs Galaxy - skipping a couple tests that will cause Galaxy and its dependencies to be downloaded results in a significant speed up. This subset of tests can be run with `make quick-test`.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

17.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
2. The pull request should work for Python >=3.7. Check https://travis-ci.org/galaxyproject/planemo/pull_requests and make sure that the tests pass for all supported Python versions.

17.4 Tips

To run a subset of tests:

```
% make tox ENV=py37-unit ARGS='--tests tests/test_shed_upload.py'
```

This will use Tox to run the specified tests using Python 3.7. `ENV` here can be used to specify different Python version (e.g. `py36` or `py37`).

Even more granularity is also possible by specifying specific test methods.:

```
make tox ENV=py37-unit ARGS='--tests tests/test_shed_upload.py:ShedUploadTestCase.test_
↪tar_from_git'
```

`tox` can be used to run tests directly also (use `. .venv/bin/activate` to ensure it is on your `PATH`).

```
tox -e py37-unit -- --tests tests/test_shed_upload.py
```

Tox itself is configured to wrap nose. One can skip Tox and run `nosetests` directly.

```
pytest tests/test_shed_upload.py
```

17.4.1 Tox

Tox is a tool to automate testing across different Python versions. The `tox` executable can be supplied with a `-e` argument to specify a testing environment. Planemo defines the following environments:

`py37-lint`
Lint the planemo code using Python 3.7.

`py37-lint_docs`
Lint the docs reStructuredText.

`py37-lint_docstrings`
Lint the project Python docstrings (doesn't pass currently).

`py37-unit-quick`
Run the fastest unit tests (with least external dependencies) on Python 3.7.

`py37-unit-nonredundant-noclientbuild-gx-2005`
Run tests that are marked as targeting a Galaxy branch and test against Galaxy 20.05. Skip tests that are marked as redundant or that require a Galaxy client build.

`py37-unit-gx-dev`
Run tests that are marked as targeting a Galaxy branch and test against Galaxy's dev branch.

17.4.2 Pre-commit Hooks

Planemo pull requests are automatically linted and tested using TravisCI. A git pre-commit hook can be setup to lint and/or test Planemo before committing to catch problems that would be detected by TravisCI as early as possible.

The following command will install a pre-commit hook that lints the Planemo code:

```
make setup-git-hook-lint
```

To also run the faster planemo tests, the following command can be used to setup a more rigorous pre-commit hook:

```
make setup-git-hook-lint-and-test
```

CHAPTER EIGHTEEN

PROJECT GOVERNANCE

This document informally outlines the organizational structure governing the Planemo code base hosted at https://github.com/galaxyproject/planemo. This governance extends to code-related activities of this repository such as releases and packaging and related Planemo projects such planemo-machine. This governance does not include any other Galaxy-related projects belonging to the `galaxyproject` organization on GitHub.

18.1 Benevolent Dictator for Now (BDFN)

John Chilton (@jmchilton) is the benevolent dictator for now (BDFN) and is solely responsible for setting project policy. The BDFN is responsible for maintaining the trust of the developer community and so should be consistent and transparent in decision making processes and request comment and build consensus whenever possible.

The BDFN position only exists because the developers of the project believe it is currently too small to support full and open governance at this time. In order to keep things evolving quickly, it is better to keep procedures and process to a minimum and centralize important decisions with a trusted developer. The BDFN is explicitly meant to be replaced with a more formal and democratic process if the project grows to a sufficient size or importance.

The *committers* group is the group of trusted developers and advocates who manage the Planemo code base. They assume many roles required to achieve the project's goals, especially those that require a high level of trust.

The BDFN will add committers as he or she see fits, usually after a few successful pull requests. Committers may commit directly or merge pull requests at their discretion, but everyone (including the BDFN) should open pull requests for larger changes.

In order to encourage a shared sense of ownership and openness, any committer may decide at any time to request a open governance model for the project be established and the BDFN must replace this informal policy with a more formal one and work with the project committers to establish a consensus on these procedures.

18.2 Committers

- Dannon Baker (@dannon)
- Bérénice Batut (@bebatut)
- Simon Bray (@simonbray)
- Martin Cech (@martenson)
- John Chilton (@jmchilton)
- Peter Cock (@peterjc)
- Björn Grüning (@bgruening)

- Helena Rasche (@hexylena)
- Nicola Soranzo (@nsoranzo)
- Marius van den Beek (@mvdbeek)

CHAPTER

NINETEEN

RELEASE CHECKLIST

This page describes the process of releasing new versions of Planemo.

- Review `git status` for missing files.
- Verify the latest Travis CI builds pass.
- Update `HISTORY.rst` with the help of `scripts/bootstrap_history.py`
- `make open-docs` and review changelog.
- Ensure the target release is set correctly in `planemo/__init__.py` (`version` will be a `devN` variant of target release).
- `make clean && make lint && make test`
- Commit outstanding changes.
- Update version and history, commit, add tag, mint a new version and push everything upstream with `make release`
- The new tag should automatically push the new release to PyPI via the `deploy` job of the GitHub Actions workflow defined in `.github/workflows/ci.yaml` . If this didn't work, you can `git checkout` the tag and push to PyPI by executing `make release-artifacts`

CHAPTER

TWENTY

HISTORY

20.1 0.75.22 (2024-04-04)

- Fix parsing of training `metadata.yaml` files (thanks to @nsoranzo). Pull Request 1439
- Fix markdown template: add missing closing details tag (thanks to @bernt-matthias). Pull Request 1440
- Misc workflow linting improvements (thanks to @bernt-matthias). Pull Request 1437
- Don't skip requirements with an environment marker (thanks to @nsoranzo). Pull Request 1433
- Add *–host* parameter to allow listening on non-default hosts (thanks to @selten). Pull Request 1430

20.2 0.75.21 (2024-02-01)

- Add option to pass extra arguments to *docker run* (thanks to @bernt-matthias). Pull Request 1428
- Fix workflow_edit (thanks to @mvdbeek). Pull Request 1427
- Update to black 2024 stable style (thanks to @nsoranzo). Pull Request 1425

20.3 0.75.20 (2024-01-30)

- Fix use of *package_name* attribute on *CondaTarget* objects (thanks to @nsoranzo). Pull Request 1424
- Don't crash autoupdate on non-PEP440-compliant tool versions (thanks to @nsoranzo). Pull Request 1422
- Add now mandatory readthedocs config files (thanks to @nsoranzo). Pull Request 1419
- Update action versions (thanks to @nsoranzo). Pull Request 1416

20.4 0.75.19 (2023-12-06)

- Update upper bound for galaxy packages to < 23.2 (thanks to @bernt-matthias). Pull Request 1388
- Fix workflow download when using instance id (thanks to @mvdbeek). Pull Request 1412
- Remove introduction header from training init (thanks to @hexylena). Pull Request 1410

20.5 0.75.18 (2023-11-16)

- Pretty-print workflow and use correct version of linked workflow in `workflow_test_init` (thanks to @mvdbeek). Pull Request 1408
- Fix running workflow tests when there are multiple tests (thanks to @mvdbeek). Pull Request 1407
- Fix matching of tool ids to autoupdate (thanks to @nsoranzo). Pull Request 1406

20.6 0.75.17 (2023-11-01)

- Implement running tests against existing invocation (thanks to @mvdbeek). Pull Request 1401
- Fix test result download (thanks to @mvdbeek). Pull Request 1402

20.7 0.75.16 (2023-11-01)

- No changes

20.8 0.75.15 (2023-10-29)

- Change info message for markdown readme in repo (thanks to @bernt-matthias). Pull Request 1398
- Make test reports more resilient to failing invocations (thanks to @mvdbeek). Pull Request 1400
- Fix zenodo test, drop explicit datatype mapping (thanks to @mvdbeek). Pull Request 1399
- Fix linting rule selection (thanks to @bernt-matthias). Pull Request 1396
- Add missing `Astronomy`, `CLIP-seq` and `Materials science` TS categories (thanks to @bernt-matthias). Pull Request 1397

20.9 0.75.14 (2023-10-19)

- Fix workflow_lint for json output values (thanks to @mvdbeek). Pull Request 1395

20.10 0.75.13 (2023-10-18)

- Add –skip to shed_lint (thanks to @bernt-matthias). Pull Request 1394
- Remove API key requirement from training_init (thanks to @hexylena). Pull Request 1393
- Try to fix planemo test workflow when output is collection with identical name (thanks to @lldelisle). Pull Request 1391

20.11 0.75.12 (2023-09-18)

- Track subworkflow invocations after main invocation is scheduled (thanks to @mvdbeek). Pull Request 1389

20.12 0.75.11 (2023-09-14)

- Implement automatic tool generation based on the source code of the tool (thanks to @Kulivox). Pull Request 1263
- Support for testing workflows with conditional steps (thanks to @mvdbeek). Pull Request 1387

20.13 0.75.10 (2023-09-01)

- Add pick_value to distro_tools (thanks to @mvdbeek). Pull Request 1385
- Allow missing conda_exec (thanks to @mstabrin). Pull Request 1384
- Fix profile version test (thanks to @bernt-matthias). Pull Request 1383
- Fix type comparisons (thanks to @bernt-matthias). Pull Request 1382
- Added a note that docker is required for docker and biocontainers option (thanks to @paulzierep). Pull Request 1355
- Optimize disk space usage for *planemo test* (thanks to @bernt-matthias). Pull Request 1378
- Fix for change of base_url in BioBlend 1.2.0 (thanks to @nsoranzo). Pull Request 1379

20.14 0.75.9 (2023-06-29)

- Update collection operation tool list (thanks to @mvdbeek). Pull Request 1377
- Officially support Python 3.10 and 3.11 (thanks to @nsoranzo). Pull Request 1375
- Fix links (thanks to @nsoranzo). Pull Request 1374
- Fix `test_run_gxtool_randomlines` test on Galaxy dev branch (thanks to @nsoranzo). Pull Request 1373

20.15 0.75.8 (2023-06-09)

- Bump galaxy package requirements to allow for 23.0 (thanks to @bernt-matthias). Pull Request 1372
- Sanitize and make output names unique (thanks to @mvdbeek). Pull Request 1371
- Increase bioblend retries for bad networks (thanks to @hexylena). Pull Request 1369
- Add `--biocontainers` option to shed_lint (thanks to @mvdbeek). Pull Request 1370
- fix regex of orcid + add test (thanks to @lldelisle). Pull Request 1364
- Drop *allow_none* for finding repository (thanks to @mvdbeek). Pull Request 1358

20.16 0.75.7 (2023-03-01)

- Drop call to `escape_non_unicode_symbols` (thanks to @nsoranzo). Pull Request 1357

20.17 0.75.6 (2023-02-22)

- Allow running autoupdate against external server (thanks to @mvdbeek). Pull Request 1265
- Fix extraction of orcid identifiers for .dockstore.yml (thanks to @lldelisle). Pull Request 1350

20.18 0.75.5 (2023-02-10)

- Bump galaxy packages to 22.05 (thanks to @bernt-matthias). Pull Request 1275

20.19 0.75.4 (2023-02-09)

- Store datasets by UUID (thanks to @bernt-matthias). Pull Request 1347
- Fix orcid regex for dockstore_init (thanks to @lldelisle). Pull Request 1348
- Fix virtualenv dir bookkeeping (thanks to @wm75). Pull Request 1346
- Deprecate introduction slides folder (thanks to @hexylena). Pull Request 1344
- Remove note recommending installing dev versions from *planemo run* tutorial (thanks to @simonbray). Pull Request 1338
- Rename configuration option removed in tox 4.0 (thanks to @nsoranzo). Pull Request 1337
- Remove travis-ci.org badge from README.rst (thanks to @SimonWaldherr). Pull Request 1334

20.20 0.75.3 (2022-11-30)

- Mount test data dir read-only in docker (thanks to @mvdbeek). Pull Request 1327
- Add traceback to report template (thanks to @bernt-matthias). Pull Request 1332
- Add indent and sort_keys to tool_test_json (thanks to @simonbray). Pull Request 1330
- Fix `planemo shed_test` (thanks to @mvdbeek). Pull Request 1329
- Add h5py dependency, required for comparing h5 files (thanks to @mvdbeek). Pull Request 1326
- Update GitHub action versions (thanks to @nsoranzo). Pull Request 1322
- Fix “glone” typo in error message (thanks to @bernt-matthias). Pull Request 1325
- Type annotation for input staging-related code (thanks to @nsoranzo). Pull Request 1320
- Add creator dockstore (thanks to @lldelisle). Pull Request 1314

20.21 0.75.2 (2022-11-02)

- Compare versions, not tool ids to find latest tool ids (thanks to @mvdbeek). Pull Request 1313
- Fix `for_paths` when path is directory of tools (thanks to @mvdbeek). Pull Request 1312
- Fix workflow_lint with list + check elements in collection (thanks to @lldelisle). Pull Request 1310
- Drop copy_tree workaround for tool sources (thanks to @mvdbeek). Pull Request 1308

20.22 0.75.1 (2022-10-31)

- Use *pytest.raises()* instead of ad-hoc *assert_raises_regexp()* context manager (thanks to @nsoranzo). Pull Request 1302
- Add planemo_ci_setup command (thanks to @mvdbeek). Pull Request 1304
- Don't fail planemo autoupdate if tool version not found in tool shed (thanks to @lldelisle). Pull Request 1305
- workflow_lint: ensure that tool shed tool ids are valid (thanks to @lldelisle). Pull Request 1306
- Fix recording of virtual_env_dir (thanks to @mvdbeek). Pull Request 1307

20.23 0.75.0 (2022-10-28)

- Restore running tool tests against directory (thanks to @mvdbeek). Pull Request 1303
- Update outdated cuffmerge url (thanks to @martenson). Pull Request 1247
- Set upstream branch when pushing workflows to GitHub (thanks to @simonbray). Pull Request 1249
- restore –no_cleanup to set cleanup_job to never (thanks to @bernt-matthias). Pull Request 1255
- Drop support for Python 3.6 (thanks to @simonbray). Pull Request 1257
- Replace CoC with link to GalaxyProject's one (thanks to @nsoranzo). Pull Request 1259
- Mains resource selector must be skipped (thanks to @hexylena). Pull Request 1260
- Ignore cloudflare 503 status when checking links (thanks to @bernt-matthias). Pull Request 1262
- Document the use of mandatory macro parameters and add named macro tokens (thanks to @bernt-matthias). Pull Request 1212
- Bump galaxy package requirements to 22.01 (thanks to @bernt-matthias). Pull Request 1264
- Run local galaxy via gravity (thanks to @mvdbeek). Pull Request 1232
- Lint randomlines.xml file (thanks to @simonbray). Pull Request 1270
- Check if main requirement is *None* (thanks to @bernt-matthias). Pull Request 1274
- Planemo type annotation: module planemo.commands.cmd_autoupdate (thanks to @gallardoalba). Pull Request 1278
- Planemo type annotation: module planemo.cli (thanks to @gallardoalba). Pull Request 1277
- Planemo type annotation: module cmd_clone (thanks to @gallardoalba). Pull Request 1279
- Planemo type annotation: module cmd_tool_init (thanks to @gallardoalba). Pull Request 1281
- Add type annotations to `planemo.autoupdate` module (thanks to @nsoranzo). Pull Request 1283

- Planemo type annotation: module cmd_normalize (thanks to @gallardoalba). Pull Request 1280
- Planemo type annotation: module planemo.conda (thanks to @gallardoalba). Pull Request 1284
- Add type annotations to `planemo.glob` and `planemo.virtualenv` (thanks to @nsoranzo). Pull Request 1287
- Drop `conda_lint` command (thanks to @nsoranzo). Pull Request 1288
- Type annotations for planemo.bioblend, planemo.git, and planemo.cwl.run (thanks to @adRn-s). Pull Request 1285
- Add FAQ page to docs (thanks to @simonbray). Pull Request 1271
- Add type annotations to `planemo.runnable` and `planemo.workflow_lint` (thanks to @nsoranzo). Pull Request 1291
- Make *–channels* also affect mulled channels and update/extend howto use bioconda artifacts (thanks to @bernt-matthias). Pull Request 1227
- Planemo type annotation: config, context and factory (thanks to @gallardoalba). Pull Request 1292
- [Training] update templates to use new, more accessible box style (thanks to @shiltemann). Pull Request 1293
- Pre-create expected output file on disk (thanks to @mvdbeek). Pull Request 1276
- Control publish setting in .dockstore.yml, fix first release not appearing on dockstore (thanks to @mvdbeek). Pull Request 1295
- Fix disclosure css for summary elements (thanks to @mvdbeek). Pull Request 1294
- Enable providing multiple *–tool_data_table* options (thanks to @mvdbeek). Pull Request 1296
- Make startup timeout configurable (thanks to @mvdbeek). Pull Request 1298
- Fix printing planemo test logs (thanks to @mvdbeek). Pull Request 1299
- Fix workflow test when input is optional but also workflow output (thanks to @mvdbeek). Pull Request 1297

20.24 0.74.11 (2022-06-08)

- Removing broken link, update it to current doc (thanks to @profgiuseppe). Pull Request 1244
- Implement nested collection inputs and outputs in workflow_test_init Pull Request 1242
- More fixes for auto-generating workflow tests Pull Request 1241
- Fix workflow_test_init for collection outputs Pull Request 1239

20.25 0.74.10 (2022-05-31)

- Allow specifying URL and API key with workflow autoupdate and docs improvements (thanks to @simonbray). Pull Request 1237
- Pin planemo to last known working major galaxy version Pull Request 1230
- Drop unused Python dependencies and upgrade syntax to Python 3.6 Pull Request 1228
- Update *best_practice_search()* for changes in galaxy-tool-util Pull Request 1224
- Make galaxy config *cleanup_job* depend on *–no_cleanup* (thanks to @bernt-matthias). Pull Request 1226
- Tool builder: add profile and suffix version (thanks to @gallardoalba). Pull Request 1222

- training-init: add the FAQ index page to tutorial folder (thanks to @shiltemann). Pull Request 1217
- Adding best practices and assertion checks to workflow_lint (thanks to @simonbray). Pull Request 1213
- Updates to workflow autoupdate required for IWC bot (thanks to @simonbray). Pull Request 1214
- Add test and fix for failing docker_galaxy engine (thanks to @simonbray). Pull Request 1215
- Generate workflow test from invocation id (thanks to @simonbray). Pull Request 1209
- Fixed minor typo in documentation (thanks to @stain). Pull Request 1206
- Add missing ToolShed categories Pull Request 1207
- Use WorkflowId rather than StoredWorkflowId when autoupdating subworkflows (thanks to @simonbray). Pull Request 1205
- Always use random id_secret for testing (thanks to @bernt-matthias). Pull Request 1198
- Add rerun subcommand for rerunning jobs (thanks to @simonbray). Pull Request 1140

20.26 0.74.9 (2021-11-03)

- Fix rendering of subworkflow steps for workflow testing report (thanks to @simonbray). Pull Request 1200
- Replace Galaxy interactor galaxy_requests_post with make_post_request from BioBlend (thanks to @simonbray). Pull Request 1201

20.27 0.74.8 (2021-10-10)

- Exclude click 8.0.2. Pull Request 1196
- Add tool version numbers to autoupdate logging (thanks to @simonbray). Pull Request 1188
- Allow tool autoupdate without conda installation (thanks to @simonbray). Pull Request 1193
- use correct key execution_problem in template (thanks to @bernt-matthias). Pull Request 1195

20.28 0.74.7 (2021-09-21)

- Fix documentation to include `--download_outputs` flag (thanks to @simonbray). Pull Request 1184
- Select refgenie config based on Galaxy version Pull Request 1187
- Extend autoupdate subcommand to workflows (thanks to @simonbray). Pull Request 1151

20.29 0.74.6 (2021-07-23)

- Add JSON report for planemo run invocations (thanks to @simonbray). Pull Request 1153
- Ignore failure to download output datasets Pull Request 1179
- Allow location to point to url for outputs Pull Request 1180
- Fix –shed_install for gxformat2 workflows Pull Request 1182

20.30 0.74.5 (2021-06-25)

- Remove iuc from default channels Pull Request 1170
- Fix parsing of changelog for git release Pull Request 1171
- Remove legacy commands, egg handling Pull Request 1172
- Use bioblend's invoke_workflow Pull Request 1173
- Create more useful output for failed invocations Pull Request 1174
- Improve dockstore_init Pull Request 1177

20.31 0.74.4 (2021-06-01)

- Relicense under the MIT license Pull Request 1169
- Revise log levels (thanks to @bernt-matthias). Pull Request 1165
- Create upload_data subcommand (thanks to @simonbray). Pull Request 1164
- Create `--download_outputs` flag for the `run` command (thanks to @simonbray). Pull Request 1157
- Make simultaneous file upload configurable for the run and test commands (thanks to @simonbray). Pull Request 1156
- Add option to add tags to a history with the `run` command (thanks to @simonbray). Pull Request 1154
- Revise Allure reporting experience for workflows. Pull Request 1152

20.32 0.74.3 (2021-02-25)

- Load both cat1 versions when testing workflows Pull Request 1146
- Fix isolated virtualenv not getting activated Pull Request 1145
- Use bioblend's make_get_request for authenticated request Pull Request 1144
- Display live logs when Galaxy is run in background Pull Request 1142

20.33 0.74.2 (2021-02-21)

- Allow testing dir of workflows Pull Request 1095
- Fix container register for gh workflow Pull Request 1135, Pull Request 1133
- Don't fail URL linting if blocked by CloudFlare Pull Request 1134 1133`_
- Allow planemo run to stage exisiting datasets and relative paths (thanks to @simonbray). Pull Request 1128

20.34 0.74.1 (2021-01-03)

- Fix `ci_find_tools` and `ci_find_repos` commands. Pull Request 1127

20.35 0.74.0 (2020-12-30)

- Allow running Galaxy workflow tests against externally defined workflows. Pull Request 1126, Pull Request 1125, Pull Request 1123
- Require Python `tabulate` package for the `list_invocations` command. Pull Request 1124

20.36 0.73.0 (2020-12-28)

- Integrate important features from gxwf for running workflows - including building up profile commands for creating aliases, allowing referencing workflows by external IDs, and listing invocations (thanks to @simonbray). Pull Request 1076
- Documentation for using `planemo run` to execute workflows (thanks to @simonbray). Pull Request 1102
- Add `workflow_upload` command for publishing each workflow of a repository with many workflows to their own standalone repository. Pull Request 1091
- Update github commands to authenticate with a token rather than username/password (thanks to @simonbray). Pull Request 1083
- Document "advanced" tool test debugging (thanks to @bernt-matthias). Pull Request 1108
- Various fixes for workflow commands - including `workflow_convert`, `workflow_lint`, `workflow_job_init`, and `workflow_test_init` (thanks to @simonbray). Pull Request 1101, Pull Request 1118, Pull Request 1121, Pull Request 1116, Pull Request 1064
- Allow outputting test results as Allure framework results. Pull Request 1115
- Fix `run_tests.sh` invocation Pull Request 1099
- Tiny typo in debugging output (thanks to @abretaud). Pull Request 1066
- Fix typo in 'planemo test' help text for –skip_venv (thanks to @peterjc). Pull Request 1068
- Fixes for CLI when `external_galaxy` is used as the engine (thanks to @simonbray). Pull Request 1072
- Updating base image to 20.05 for training topics (thanks to @bedroesb). Pull Request 1074
- Changes to update_test_data testing mode (thanks to @simonbray). Pull Request 1079
- Fix docker options when filling `job_conf.xml` template. Pull Request 1086

- Explicit tests for Galaxy 20.09. Pull Request 1093
- Minor fix for `ci_find_repos` command. Pull Request 1094
- Fix a couple of Cheetah urls in Galaxy tool documentation (thanks to @martenson). Pull Request 1096
- Fix doc link from a redirect loop to a section (thanks to @martenson). Pull Request 1110
- Clarify `tutorial.md` usage of citations (thanks to @blankenberg). Pull Request 1114
- Fix `ZeroDivisionError` when no tests are executed (thanks to @simonbray). Pull Request 1120

20.37 0.72.0 (2020-08-04)

- More documentation/support around running workflows including new command to initialize workflow jobs `workflow_init_job`. Pull Request 1052
- Workflow tests and documentation for tagging inputs. Pull Request 1058
- Various documentation improvements. Pull Request 1061, Pull Request 1062
- Add mypy type checking. Pull Request 1060
- Progress decoupling Planemo's core from click & CLI interactions. Pull Request 1059
- Tests for workflow testing script. Pull Request 821

20.38 0.71.0 (2020-08-03)

- Drop Python 2 support. Pull Request 1026
- Rev Galaxy dependencies - including bumping bioblend to 0.14.0, galaxy-tool-util, and unpinning cwltool (last of these thanks to thanks to @TMiguelT). Pull Request 1038, Pull Request 1034
- Workflow linting, best practices, and tooling to assist in following them. Pull Request 1028, Pull Request 1049, Pull Request 1051 Pull Request 1044
- Substantial rewrites to Galaxy workflow input staging - including allow nested collection and composite inputs to Galaxy for `run` and `test`. Pull Request 900, Pull Request 1029
- Remove assorted older, likely unused commands. Pull Request 1043
- Update installation.rst (thanks to @mblue9). Pull Request 1032
- Automatic PyPI upload on tag using GitHub Actions. Pull Request 994
- Fix quay repository presence check for single target builds. Pull Request 993
- More fine grained options for `--shed_install` (thanks to @AndreasSko). Pull Request 1001
- Change default Python version for Galaxy (thanks to @bernt-matthias). Pull Request 1021
- Sort tests by id when merging (thanks to @bernt-matthias). Pull Request 1022
- Add `--group_tools` option to `ci_find_tools` (thanks to @bernt-matthias). Pull Request 1008
- Add shared data library path to the data upload box for training material (thanks to @shiltemann). Pull Request 1013
- Add support for tool versions to tutorial template generator (thanks to @shiltemann). Pull Request 1041
- Only copy test files if they don't exist. Pull Request 1037

- Improvements to loading stock tools for workflow testing and serving (add new stock tools to list and check subworkflows). Pull Request 1031
- Fix link for composite data type docs (thanks to @bernt-matthias). Pull Request 1020
- Do not use `gi._make_url()` internal BioBlend method.
- Switch CWL examples to use https://schema.org/version/latest/schema.rdf (thanks to @mr-c). Pull Request 1015
- Fix docs not to claim Galaxy can't run on Python 3. Pull Request 1023
- Improved abstractions around target Galaxy instance. Pull Request 1046
- Add empty refgenie config for tests (thanks to @blankenberg). Pull Request 1025
- Substantial reworking of testing infrastructure. Pull Request 1024, Pull Request 1003, Pull Request 1011, Pull Request 1006, Pull Request 1040, Pull Request 1036, Pull Request 1042

20.39 0.70.0 (2020-01-29)

- Temporarily add galaxy-util requirement Pull Request 991
- Make symlinks in tool tree work for planemo test Pull Request 988
- Reduce use of `shell=True` in subprocesses Pull Request 989
- Drop planemo database seed option Pull Request 985
- Don't execute `untar_to()` subprocesses through the shell Pull Request 984
- Allow setting database_connection for planemo test runs Pull Request 986
- Fix copy-paste mistakes Pull Request 983
- Add planemo list_repos command Pull Request 982
- Make container_register build files with headers and include base_image Pull Request 980
- Replace deprecated galaxy-lib requirement with galaxy-tool-util Pull Request 978
- Close all opened files (thanks to @bernt-matthias). Pull Request 979
- Build single requirement container, log if requirement not in best-practice channels Pull Request 977
- Use tojson jinja2 filter instead of json.dumps Pull Request 975
- Add merge_test_reports command Pull Request 974
- Implement github workflow and fix profile commands if psql unavailable Pull Request 976
- Fix planemo lint –biocontainers if no build number in container Pull Request 972
- Update a training command (thanks to @hexylena). Pull Request 973
- Allow passing through GALAXY_VIRTUAL_ENV variable to venv setup Pull Request 971
- Correct help text (thanks to @hexylena). Pull Request 970
- Remove unneeded html5lib requirement Pull Request 968

20.40 0.62.1 (2019-10-14)

- Init & update submodules when installing and creating packages. Stop distributing eggs (thanks to @nsoranzo). 1ab8530

20.41 0.62.0 (2019-10-11)

- Use `unicodify()` on exceptions and subprocess outputs (thanks to @nsoranzo) Pull Request 944
- Do not override `None` with empty string (thanks to @ic4f). Pull Request 950
- Update Docker template for training material generation (thanks to @bedroesb). Pull Request 958
- Add support for suite of repos with different owners (thanks to @nsoranzo). Pull Request 959
- Link for collection details updated in the docs (thanks to @martin-raden). Pull Request 963
- Move most tests to Python 3.7, drop Python 3.4 (thanks to @nsoranzo). Pull Request 964
- Remove confusing warning Pull Request 966

20.42 0.61.0 (2019-07-08)

- Training - fix empty repeat + some formatting (thanks to @bebatut). Pull Request 926
- Training - add bibliography to tutorial template (thanks to @shiltemann). Pull Request 938
- Training - support new class definition for input in workflow step (thanks to @bebatut). Pull Request 943
- Various tool tutorial fixes ahead of GCC 2019 (thanks to @nsoranzo). Pull Request 940
- Return validation error if doi is empty (thanks to @nsoranzo). Pull Request 937

20.43 0.60.0 (2019-05-31)

- Return validation error if doi is empty Pull Request 937
- Add junit as test reporter (thanks to @selten). Pull Request 935
- Update galaxy.xsd for new python 3 compatibility attribute (thanks to @martenson). Pull Request 931
- Documentation: add a little warning for <param … multiple="true"> (thanks to @bernt-matthias). Pull Request 930

20.44 0.59.0 (2019-05-09)

- Add ability to test data manager tools (thanks to @mvdbeek). Pull Request 912
- Update Training for new requirement definition (thanks to @bebatut). Pull Request 913
- Drop amqp workaround (thanks to @mvdbeek). Pull Request 917
- Use `yaml.safe_load()` instead of deprecated `load()` (thanks to @nsoranzo). Pull Request 921
- Allow converting `tool_test_report.json` to xunit (thanks to @mvdbeek). Pull Request 918

- Fix error if testcase.data.job does not exist (thanks to @mvdbeek). Pull Request 924
- Fix deprecated `getchildren()` (thanks to @nsoranzo). Pull Request 925

20.45 0.58.2 (2019-03-01)

- Fix display of tool ids in planemo html report (thanks to @mvdbeek). Pull Request 908
- Single quotes for file names (thanks to @bernt-matthias). Pull Request 909
- Fix doc linting (thanks to @mvdbeek). Pull Request 910
- Update TS categories (thanks to @nsoranzo). 07dc6e0
- Close tag in doc help, to help with copy&paste (thanks to @blankenberg). Pull Request 914
- Update the tool XSD file (thanks to @bgruening). Pull Request 915

20.46 0.58.1 (2019-01-03)

- Update galaxy-lib requirement to 18.9.2 to add Python 3.7 support (thanks to @nsoranzo). Pull Request 906
- Fix command run by *planemo test –skip_venv* (thanks to @nsoranzo). Pull Request 907

20.47 0.58.0 (2019-01-01)

- Remove deprecated `sudo:  false` from .travis.yml (thanks to @nsoranzo). Pull Request 902
- Do not skip Galaxy client build for `planemo serve`. Install Galaxy when the directory specified with `--galaxy_root` does not exist or is empty. (thanks to @nsoranzo). Pull Request 895, Issue 845

20.48 0.57.1 (2018-11-23)

- Fix username validation for shed linting (thanks to @martenson). Pull Request 899, Issue 898

20.49 0.57.0 (2018-11-19)

- Allow `workflow_convert` to convert a native `.ga` workflows to format 2 (yaml). Pull Request 896
- New command (`workflow_edit`) to open workflow in a synchronized graphical editor. Pull Request 894
- Conda tutorial fixes (thanks to @nsoranzo). Pull Request 876
- Enable `--conda_use_local` option for `planemo test` (thanks to @nsoranzo). Pull Request 876
- When testing, skip workflow outputs that do not have a *label* set (thanks to @bgruening). Pull Request 893
- Add `__repr__` for `TestCase` to improve debugging Planemo (thanks to @bgruening). Pull Request 892
- Increase IO polling interval over time (thanks to @martenson). Pull Request 891
- Sync galaxy xsd and fix tests (thanks to @mvdbeek). Pull Request 889
- Linting fix for `W605` (thanks to @martenson). Pull Request 888

- Add icon for repeat parameters in training (thanks to @bebatut). Pull Request 887

20.50 0.56.0 (2018-10-30)

- Allow selection of Python version when starting managed Galaxy (thanks to @mvdbeek). Pull Request 874
- Change the channel priority of conda (again). (thanks to @bgruening). Pull Request 867
- Some small english corrections (thanks to @hexylena). Pull Request 868
- Print the list of excluded paths when running `ci_find_repos` (thanks to @nsoranzo). Pull Request 877
- Improved XSD lint reporting. Pull Request 871
- Fix Planemo writing a file called `gx_venv_None`. Pull Request 870
- Update cwltool and galaxy-lib dependencies for Python 3.7 (thanks to @nsoranzo). Pull Request 864
- Fix to make workflow testing more robust. Pull Request 882

20.51 0.55.0 (2018-09-12)

- Add commands to create Galaxy training materials (thanks to @bebatut). Pull Request 861
- Fix *planemo test* when TEMP env variable contain spaces (thanks to @nsoranzo). Pull Request 851
- Support testing a completely remote galaxy instance (thanks to @hexylena). Pull Request 856
- Allow naming history from command line (thanks to @hexylena). Pull Request 860
- Sync galaxy.xsd from galaxy repo (thanks to @nsoranzo). Pull Request 866
- Fix ServeTestCase.test_shed_serve test (thanks to @*nsoranzo*). bad810a

20.52 0.54.0 (2018-06-06)

- Better support for testings against different versions of Galaxy efficiently and robustly. Pull Request 849
- New database version (thanks to @bgruening). Pull Request 847
- Hyperlink DOIs against preferred resolver (thanks to @katrinleinweber). Pull Request 850
- Tests for collection inputs to workflows. Pull Request 843
- Bring in Ephemeris sleep function - hopefully makes serve tests a bit more robust. b12b117
- More tutorial testing, tutorial updates. 016b923, 324c776, 2002b49
- More isolated `test_shed_upload.py` tests. 72d2ca7
- Add filetype support for workflow test inputs (thanks to @bgruening). Pull Request 842
- Add `--no_shed_install` option, to prevent shed installs as part of workflow testing. Pull Request 841
- Small docs fix (thanks to @hexylena). Pull Request 848

20.53 0.53.0 (2018-05-22)

- Make Planemo testing easier for CWL tools and workflows in various ways and update tutorials to reflect these simplifications. Pull Request 837
- Test and fix running workflow tests against externally managed Galaxy servers. Pull Request 833, Pull Request 836
- Allow using URIs for inputs of workflow test. Pull Request 840
- Slide Galaxy testing window to include 18.05 and drop 17.09. Pull Request 838

20.54 0.52.0 (2018-05-20)

- Allow optional disabling of Galaxy single user mode. Pull Request 835
- Fix for path pasting options during workflow testing. Pull Request 834

20.55 0.51.0 (2018-05-19)

- Fix essentially all Conda and BioContainers related functionality to allow parity between CWL and existing Galaxy functionality - fixes and enhances many commands including `lint`, `conda_install`, `conda_env`, `test`, `run`, and `mull`. Pull Request 828
- Add two new tutorials for Conda and Container development with CWL tools that mirrors the existing tutorials for Galaxy tools - including new CWL exercises, answers, and example project templates. 347c622
- Improve the CWL generated by the `tool_init` command to properly deal with `SoftwareRequirement` s and generate more idiomatic CWL. Pull Request 820, a5c72e3
- Add new engine type (`--engine toil`) for testing and running CWL tools (requires manually installing Toil with `pip install toil` in Planemo's environment). Pull Request 831
- Add documentation for the Galaxy Workflow and CWL test format files (includes information on configuring various test engines). Pull Request 832
- Better default logging config for CWL development. Pull Request 830
- Various fixes for the `conda_search` command. Pull Request 826
- Fix test coverage configuration. Pull Request 822
- Reorganize .travis.yml for clarity. Pull Request 829
- More isolated, robust unit tests that use git. Pull Request 827, Pull Request 818
- Fix default list of best-practice Conda channels. Pull Request 825
- Refactor tests to speed up quick tests - fewer buggy URLs fetched in "quick" mode. Pull Request 823
- Fix upload configuration of workflow testing to default (overrideable) external Galaxies to not use path pasting. Pull Request 816
- Fix test number parsing for workflow tests. Pull Request 817

20.56 0.50.1 (2018-05-11)

- Fix the process of waiting on Galaxy to boot up for the Docker Galaxy container `--engine`.

20.57 0.50.0 (2018-05-10)

- Fixes and small CLI tweaks to get the Docker Galaxy container working as an `--engine` for the run, serve, and test commands.

20.58 0.49.2 (2018-05-09)

- Various small fixes for new external Galaxy engine type.

20.59 0.49.1 (2018-05-06)

- Fix PyPI README rendering for 0.49.0 release changes.

20.60 0.49.0 (2018-05-06)

- Implement external Galaxy engine. Pull Request 781
- Restructure serve testing code for reuse. Pull Request 795
- Improve test report handling for JSON generated via galaxy-lib testing script. Pull Request 799
- Improve how various branches of Galaxy are tested. Pull Request 800
- Added documentation for `GALAXY_MEMORY_MB` (thanks to @bernt-matthias). Pull Request 801
- Log tool config in verbose logging mode. Pull Request 802
- Replace `r` channel with `conda-forge` (thanks to @bgruening). Pull Request 805
- Sync `galaxy.xsd` with latest Galaxy updates (thanks to @nsoranzo). Pull Request 806
- Use `requests.get()` when validating http URLs (thanks to @nsoranzo). Pull Request 809
- Do not consider tools with “deprecated” in the path (thanks to @bgruening). Pull Request 810
- Automatically load tools shipped with Galaxy when testing, running, or serving workflows that reference these tools. Pull Request 790
- Revise README and touch up documentation in general. Pull Request 787
- Various small changes to testing and test framework. Pull Request 792
- Various Python 3 fixes. 8cfe9e9, 41f7df1
- Fixes for Galaxy 18.0X releases. Pull Request 803, dc443d6

20.61 0.48.0 (2018-02-28)

- Run all CI tests against Python 3 (thanks to @nsoranzo). Pull Request 768 and Pull Request 774
- Python 3 fix - subprocess with `universal_newlines=True` (thanks to @peterjc). Pull Request 764
- Record CWL conformance test results using JUnit xml (thanks to @mr-c). Pull Request 756
- Restore run test case for simple Galaxy tools. Pull Request 769
- Enhancements to Galaxy profiles and workflow testing. Pull Request 773
- Fix resolving & installing shed repositories from workflows for `test` and `run` commands. Pull Request 776
- Implement planemo command to convert format 2 workflows into .ga workflows. Pull Request 771
- Add a native Galaxy workflow (.ga) testing test. Pull Request 770
- Drop Brew support but add more detailed install instructions. Pull Request 761
- Clean up CWL conformance test execution. Pull Request 753
- Assorted small CWL and deamon serve fixes. Pull Request 759

20.62 0.47.0 (2017-11-18)

- Update to the latest Galaxy tool XSD (thanks to @nsoranzo). Pull Request 747
- Re-fix problem when shed_update would fail if nothing to update (thanks to @nsoranzo). Pull Request 747
- Update instructions for installation via conda (thanks to @nsoranzo) . Pull Request 743
- Bug fix for MacOS *chmod* doesn't support *–recursive* flag. (thanks to @dfornika). Pull Request 739
- Bug fix to also *socket.error* when linting URLs (thanks to @nsoranzo). Pull Request 738
- Disable broken tests. Pull Request 745

20.63 0.46.1 (2017-09-26)

- Rev to latest versions of bioblend and galaxy-lib for various fixes related to CWL.

20.64 0.46.0 (2017-09-15)

- Change behavior of `--docker` flag, for a few releases it would require Galaxy use a container for every non-upload tool. This breaks various conversion tools for instance and so was reverted. Pull Request 733
- Add 'Accept' header when linting doc URLs (thanks to @nsoranzo). Pull Request 725
- Fix *–conda_auto_install* help (thanks to @nsoranzo). Pull Request 727
- Incremental progress toward CWL support via Galaxy. Pull Request 729, Pull Request 732
- Update galaxy-lib to latest version to fix various issues. Pull Request 730
- Fix lint detected problems with documentation. Pull Request 731

20.65 0.45.0 (2017-09-06)

- Update to the latest galaxy-lib for Conda fixes. (thanks @nsoranzo) and updated CWL utilities. Pull Request 716, Pull Request 723
- Update Conda channel order to sync with Bioconda (thanks to @nsoranzo). Pull Request 715
- Experimental support running CWL workflows through the CWL fork of Galaxy.
- Mention `planemo command --help` in main help (thanks to @peterjc). Pull Request 709
- Bugfix handle `None` requirement versions when registering containers (thanks to @bgruening). Pull Request 704
- Bugfix for dependencies by pinning ruamel.yaml version (thanks to @mvdbeek). Pull Request 720

20.66 0.44.0 (2017-06-22)

- Fix and improve Galaxy root option specification options. Pull Request 701, 8a608e0
- Update *planemo mull* to use a default action of *build-and-test* since *build* no longer cleans up itself. ecc1bc2
- Add a command to pre-install Involucro. Pull Request 702

20.67 0.43.0 (2017-06-22)

- Remove stdio from generated tools - just use exit_code for everything. 91b6fa0
- Implement some ad-hoc documentation tests. Pull Request 699
- A large number of small enhancements and fixes for the documentation and example projects.

20.68 0.42.1 (2017-06-16)

- Fix Readme typos (thanks to @manabuishii) 904d77a
- Fix *container_register* to create pull requests against the newly finalized home of the multi-package-containers registry repository. 9636682
- Fix *use_global_config* and *use_env_var* for options with unspecified defaults. 475104c

20.69 0.42.0 (2017-06-15)

- Conda/Container documentation and option naming improvements. Pull Request 684
- Sync *galaxy.xsd* with latest upstream Galaxy updates (thanks to @nsoranzo). Pull Request 687
- Fix *ci_find_repos* command to not filter repos whose only modifications where in subdirs (thanks to @nsoranzo). Pull Request 688
- Update *container_register* for mulled version 2 and repository name changes. Pull Request 689
- Better pull request messages for the *container_register* command. Pull Request 690

20.70 0.41.0 (2017-06-05)

- Fix `shed_update` not fail if there is nothing to update (thanks to @nsoranzo). Issue 494, Pull Request 680
- Conda documentation and option naming improvements. Pull Request 683
- Implement `container_register` for tool repositories. Pull Request 675
- Fix `hub` binary installation for Mac OS X. Pull Request 682

20.71 0.40.1 (2017-05-03)

- Fix data manager configuration to not conflict with original Galaxy at `galaxy_root` (thanks to @nsoranzo). Pull Request 662
- Fix `filter_paths()` to not partial match paths when filtering shed repositories (thanks to @nsoranzo). Pull Request 665
- Fix description when creating `.shed.yml` files (thanks to @RJMW). Pull Request 664

20.72 0.40.0 (2017-03-16)

- Implement instructions and project template for GA4GH Tool Execution Challenge Phase 1. 84c4a73
- Eliminate Conda hack forcing `/tmp` as temp directory. b4ae44d
- Run dependency script tests in isolated directories. 32f41c9
- Fix OS X bug in `planemo run` by reworking it to wait using urllib instead of sockets. 3129216

20.73 0.39.0 (2017-03-15)

- Implement documentation and examples for Conda-based dependency development (under “Advanced” topics). Pull Request 642, Pull Request 643
- Implement documentation and examples for container-based dependency development (under “Advanced” topics). 0a1abfe
- Implement a `planemo conda_search` command for searching best practice channels from the command line. Pull Request 642
- Allow Planemo to work with locally built Conda packages using the `--conda_use_local` command. Pull Request 643, Issue 620
- Implement an `open` (or just `o`) command to quickly open the last test results (or any file if supplied). Pull Request 641
- Linting improvements and fixes due to galaxy-lib update. * WARN on test output names not found or not matching. * INFO correct information about stdio if profile version is found. * WARN if profile version is incorrect. * INFO profile version * Fix `assert_command` not detected as a valid test (fixes Issue 260).
- Have `lint --conda_requirements` check that at least one actual requirement is found. 6638caa
- Allow `conda_install` to work with packages as well as just tools. 8faf661

- Add `--global` option to conda_install to install requirements into global Conda setup instead of using an environment. 8faf661
- Implement `planemo lint --biocontainer` that checks that a tool has an available BioContainer registered. 0a1abfe
- Add more options and more documentation to the `planemo mull` command. 0a1abfe
- Hack around a bug in Conda 4.2 that makes it so `planemo mull` doesn't work out of the box on Mac OS X. 0a1abfe
- Allow URIs to be used instead of paths for a couple operations. ce0dc4e
- Implement non-strict CWL parsing option. 4c0f100
- Fixes for changes to cwltool and general CWL-relate functionality. 3c95b7b, 06bcf19, 525de8f, 9867e56, 9ab4a0d
- Eliminate deprecated XML-based abstraction from `planemo.tools`. 04238d3
- Fix `MANIFEST.in` entry that was migrated to galaxy-lib. ced5ce2
- Various fixes for the command `conda_env`. Pull Request 640
- Improved command help - both formatting and content. Pull Request 639
- Implement a `--no_dependency_resolution` option disabling conda dependency resolver. Pull Request 635, Issue 633
- Tests for new linting logic. Pull Request 638
- Fix bug where tool IDs needs to be lowercase for the shed (thanks to @bgruening). Pull Request 649
- Update seqtk version targetted by intro docs. e343b67
- Various other Conda usability improvements. Pull Request 634

20.74 0.38.1 (2017-02-06)

- Fix bug with `shed_lint --urls` introduced in 0.38.0. 84ebc1f

20.75 0.38.0 (2017-02-06)

- Trim down the default amount of logging during testing. Pull Request 629, Issue 515
- Improved log messages during shed operations. 08c067c
- Update tool XSD against latest Galaxy. fca4183, 03c9658
- Fix bug where `shed_lint --tools` for a suite lints the same tools multiple times. Issue 564, Pull Request 628

20.76 0.37.0 (2017-01-25)

- Update to the latest galaxy-lib release. This means new installs start with Miniconda 3 instead of Minicoda 2 and at a newer version. This fixes many Conda related bugs.
- Change defaults so that Conda automatically initializes and performs tool installs by default from within the spawned Galaxy server. The trio of flags `--conda_dependency_resolution`, `--conda_auto_install`, and `--conda_auto_init` are effectively enabled by default now. 4595953
- Use the Galaxy cached dependency manager by default (thanks to @abretaud). Pull Request 612
- Test Conda dependency resolution for more versions of Galaxy including the forthcoming release of 17.01.
- Update to the latest Galaxy tool XSD for various tool linting fixes. 32acd68
- Fix pip ignores for `bioconda_scripts` (thanks to @nturaga) Pull Request 614

20.77 0.36.1 (2016-12-12)

- Fix move error when using `project_init`. Issue 388, Pull Request 610
- Improved integration testing for `test` command. Pull Request 609
- Update CWL links to v1.0 (thanks to @mr-c). Pull Request 608

20.78 0.36.0 (2016-12-11)

- Bring in latest tool XSD file from Galaxy (thanks to @peterjc). Pull Request 605
- PEP8 fixes for various linting problems (thanks to @peterjc). Pull Request 606
- Update tool syntax URL to new URL (thanks to @mvdbeek). Pull Request 602

20.79 0.35.0 (2016-11-14)

- Native support for building bioconductor tools and recipes (thanks to @nturaga). Pull Request 570
- Fixes for running Galaxy via docker-galaxy-stable (thanks to @bgruening). 50d3c4a
- Import order linting fixes (thanks to @bgruening).

20.80 0.34.1 (2016-10-12)

- Mimic web browser to validate user help URLs fixing Issue 578 (thanks to @peterjc). Pull Request 591
- Fix for Bioconda recipes depending on `conda-forge` (thanks to @nsoranzo). Pull Request 590

20.81 0.34.0 (2016-10-05)

- Implement `mull` command to build containers for tools based on Conda recipes matching requirement definitions. 08cef54
- Implement `--mulled_containers` flag on `test`, `serve`, and `run` commands to run tools in "mulled" containers. Galaxy will first search locally cache containers (such as ones built with `mull`), then search the mulled namespace of quay.io, and finally build one on-demand if needed using galaxy-lib and Involucro developed by @thriqon.
- Implement `--conda_requirements` flag on `lint` command to ensure requirements can be resolved in best practice channels. 9da8387
- Allow `conda_install` command over multiple tool paths. 2e4e5fc
- Update pip as part of setting virtual environment in `Makefile` target. 19b2ee9
- Add script to auto-update Bioconda recipe for Planemo and open a pull request. f0da66f

20.82 0.33.2 (2016-09-28)

- Fix HISTORY.rst link problem that prevented correct display of content on PyPI.

20.83 0.33.1 (2016-09-28)

- Fix `lint --urls` false positives by being more restrictive with what is considered a URL (fixed by @hexylena after detailed report from @peterjc). Issue 573, Pull Request 579

20.84 0.33.0 (2016-09-23)

- Enable XSD validation of tools by default (restore old behavior with `planemo lint --no_xsd`). 1ef05d2
- Implement a `conda_lint` command to lint Conda recipes based on anaconda-verify. 6a6f164
- Implement `clone` and `pull_request` commands to ease PRs (with documentation fixes from @martenson). e925ba1, ea5324f
- Update galaxy.xsd to allow version_command's to have an interpreter attribute. 7cca2e4
- Apply improvement from @nsoranzo for Planemo's use of git diff. 6f91719
- Pull in downstream refactoring of `tool_init` code from @nturaga's Bioconductor work. ccdd2d5
- Update to latest Tool Factory code from tools-iuc. ca88b0c
- Small code cleanups. b6d8294, d6da3a8
- Fixup docs in `planemo.xml.validation`.
- Allow skipping newly required lxml dependency in setup.py. 34538de

20.85 0.32.0 (2016-09-16)

- Enhance `planemo lint --xsd` to use a fairly complete and newly official XSD definition. Pull Request 566
- Migrate and update documentation related to tool XML macros and handling multiple outputs from the Galaxy wiki (with help from @bgruening, @mvdbeek, and @nsoranzo). Pull Request 559
- Documentation fixes (thanks to @ramezrawas). Pull Request 561
- Do not fail URL linting in case of too many requests (thanks to @nsoranzo). Pull Request 565

20.86 0.31.0 (2016-09-06)

- Implement new commands to `ci_find_repos` and `ci_find_tools` to ease CI scripting. Pull Request 555

20.87 0.30.2 (2016-09-01)

- Fix another problem with Conda prefix handling when using `--conda_dependency_resolution`. f7b6c7e

20.88 0.30.1 (2016-09-01)

- Fix a problem with Conda prefix handling when using `--conda_dependency_resolution`. f7b6c7e
- Fix for quote problem in `update_planemo_recipe.bash`. 6c03de8
- Fix to restore linting of `tests/` directory and fix import order throughout module. ef4b9f4

20.89 0.30.0 (2016-09-01)

- Update to the latest galaxy-lib release and change Conda semantics to match recent updates to Galaxy. For the most robust Conda usage - use planemo 0.30+ with Galaxy 16.07 or master. 07d94bd
- Implement the `--conda_auto_init` flag for `conda_install`. ca19910
- Allow the environment variable `PLANEMO_CONDA_PREFIX` to set a default for `--conda_prefix`. 24008ab
- Fixup documentation regarding installs and Conda. ce44e87
- Fix and lint Python module import order throughout project. Pull Request 550
- Use `cp` rather than symlink to `$DOWNLOAD_CACHE` in the `dependency_script` command (thanks to @peterjc). c2204b3
- Fixes for the Homebrew recipe updater. c262b6d

20.90 0.29.1 (2016-08-19)

- Improved handling of Python 2.7 specific dependencies.

20.91 0.29.0 (2016-08-19)

- Look for sha256sum checksums during shed_lint (thanks to @peterjc). Pull Request 539
- An assortment fixes and enhancements to the `dependency_script` command (thanks to @peterjc). Pull Request 541, Pull Request 545
- Fix shed_build to respect exclude: in .shed.yml (thanks to @nsoranzo). Pull Request 540
- Fix linting of tool URLs (thanks to @nsoranzo). Pull Request 546

20.92 0.28.0 (2016-08-17)

- Fixes for bioblend v0.8.0 (thanks to @nsoranzo). 9fdf490
- Enable shed repo type update (thanks to @nsoranzo). 3ceaa40
- Create suite repositories with repository_suite_definition type by default (thanks to @nsoranzo). 057f4f0
- Include `shed_lint` in script run by `travis_init` (thanks to @peterjc). Pull Request 528
- Minor polish to the `travis_init` command (thanks to @peterjc). Pull Request 512
- Update pip and setuptools on TravisCI; fix travis_init (thanks to @peterjc). Pull Request 521
- Shorten command one line descriptions for main help (thanks to @peterjc). Pull Request 510
- Use `planemo test --no_cache_galaxy` under TravisCI (thanks to @peterjc). Pull Request 513
- Improve and fix docs ahead of GCC 2016 (thanks to @martenson). Pull Request 498, 725b232
- Add description of `expect_num_outputs` to planemo FAQ. a066afb
- Revise planemo tools docs to be more explicit about collection identifiers. a811e65
- Add more docs on existing dynamic tool output features. Pull Request 526
- Fix serve command doc (thanks to @nsoranzo). 8c088c6
- Fix *make lint-readme* (RST link errors) (thanks to @peterjc). Pull Request 525
- Add union bedgraph example to project templates (for GCC demo example). d53bcd6
- Add Flow Cytometry Analysis, Data Export, and Constructive Solid Geometry as shed categories (thanks to @bgruening, @gregvonkuster, and @nsoranzo). e890ab5, 08bb354, e2398fb
- Remove duplicated attribute in docs/writing/bwa-mem_v5.xml (thanks to Paul Stewart @pstew). Pull Request 507

20.93 0.27.0 (2016-06-22)

- Use ephemeris to handle syncing shed tools for workflow actions. 1c6cfbb
- More planemo testing enhancements for testing artifacts that aren't Galaxy tools. Pull Request 491
- Implement `docker_galaxy` engine type. eb039c0, Issue 15
- Enhance profiles to be Dockerized Galaxy-aware. Pull Request 488
- Add linter for DOI type citation - thanks to @mvdbeek. Pull Request 484

20.94 0.26.0 (2016-05-20)

- Implement `Engine` and `Runnable` abstractions - Planemo now has beta support for testing Galaxy workflows and CWL tools with Galaxy and any CWL artifact with cwltool. Pull Request 454, 7be1bf5
- Fix missing command_line in test output json. e38c436
- More explicit Galaxy `job_conf.xml` handling, fixes bugs caused by `galaxy_root` having existing and incompatible `job_conf.xml` files and makes it possible to specify defaults with fixed server name. c4dfd55
- Introduce profile commands (`profile_create`, `profile_delete`, and `profile_list`) and profile improvements (automatic postgres database creation support). Pull Request 480, a87899b
- Rework Galaxy test reporting to use structured data instead of XUnit data. 4d29bf1
- Refactor Galaxy configuration toward support for running Galaxy in docker-galaxy-stable. Pull Request 479

20.95 0.25.1 (2016-05-11)

- Tweak dependencies to try to fix cwltool related issues - such as Issue 475.

20.96 0.25.0 (2016-05-11)

- Implement Galaxy "profiles" - the ability to configure perisistent, named environments for `serve` and `test`. 5d08b67
- Greatly improved `serve` command - make `test-data` available as an FTP folder, (on 16.07) automatically log in an admin user, and many more options (such as those required for "profiles" and a `--daemon` mode).
- Two fixes to ensure more consistent, dependable `test` output. Pull Request 472, f3c6917
- Add code and documentation for linting (`lint`) and building (`tool_init`) CWL tools. a4e6958, b0b867e, 4cd571c
- If needed for Conda workaround, shorten `config_directory` path (thanks to @mvdbeek). efc5f30
- Fix `--no_cache_galaxy` option (thanks to Gildas Le Corguillé). d8f2038
- Target draft 3 of CWL instead of draft 2. 775bf49
- Fix `cwltool` dependency version - upstream changes broke compatibility. 65b999d
- Add documentation section and slides about recent Galaxy tool framework changes (with fix from @remimarenco). 069e7ba

- Add IUC standards to Planemo docs. 2ae2b49
- Improve collection-related contents in documentation (thanks in part to @martenson). fea51fc, 13a5ae7
- Add documentation on `GALAXY_SLOTS` and running planemo on a cluster. 45135ff, e0acf91
- Revise command-line handling framework for consistency and extension - allow extra options to be configured as defaults `~/.planemo.yml` including `--job_config_file` and Conda configuration options. e769118, 26e378e
- Fix `tool_init` commans options typos (thanks to Nitesh Turaga). 826d371
- Refactor galaxy-related modules into submodules of a new `planemo.galaxy` package. 8e96864
- Fix error message typo (thanks to @blankenberg). b1c8f1d
- Update documentation for recent command additions. 3f4ab44
- Rename option `--galaxy_sqlite_database` option to `--galaxy_database_seed` and fix it so it actually works. f7554d1
- Add `--extra_tools` option to `serve` command. 02a08a0
- Update project testing to include linting documentation (`docs/`), Python import order, and docstrings. a13a120, 6e1e726, 95d5cba

20.97 0.24.2 (2016-04-25)

- Revert "check `.shed.yml` owner against credentials during shed creation", test was incorrect and preventing uploads. Pull Request 425, Issue 246

20.98 0.24.1 (2016-04-08)

- Fix test summary report. Pull Request 429
- Improve error reporting when running `shed_test`. ce8e1be
- Improved code comments and tests for shed related functionality. 89674cb
- Rev galaxy-lib dependency to 16.4.1 to fix wget usage in newer versions of wget. d76b489

20.99 0.24.0 (2016-03-29)

- Drop support for Python 2.6. 93b7bda
- A variety of fixes for `shed_update`. Pull Request 428, Issue 416
- Fix reporting of metadata updates for invalid shed updates. Pull Request 426, Issue 420
- Check `.shed.yml` owner against credentials during shed creation. Pull Request 425, Issue 246
- Fix logic error if there is a problem with `shed_create`. 358a42c
- Tool documentation improvements. 0298510, a58a3b8

20.100 0.23.0 (2016-02-15)

- Fix duplicated attributes with Conda resolver (thanks to Björn Grüning). Pull Request 403
- Upgrade to latest version of galaxy-lib for more linting.
- Attempt to better handle conditional dependency on cwltool.

20.101 0.22.2 (2016-01-14)

- Fixed bug targetting forthcoming release of Galaxy 16.01.

20.102 0.22.1 (2016-01-14)

- Fixed problem with PyPI build artifacts due to submodule's not being initialized during previous release.

20.103 0.22.0 (2016-01-13)

- Add `--skip_venv` to support running Galaxy 16.01 inside of conda environments. 9f3957d
- Implement conda support. f99f6c1, ad3b2f0, 5e0b6d1
- Update LICENSE for Planemo to match Galaxy. 15d33c7
- Depend on new galaxy-lib on PyPI instead of previous hacks. . . . Pull Request 394
- Fix egg caching against master/15.10. 6d0f502
- Fix bug causing shed publishing of `.svn` directories. Issue 391
- Bug fixes for Conda support thanks to @bgruening. 63e456c
- Fix document issues thanks to @einon. Pull Request 390
- Improve client for shed publishing to support newer shed backend being developed by @hexylena. Pull Request 394
- Tool Shed `repo_id` change, @hexylena. Pull Request 398
- Various other small changes to testing, project structure, and Python 3 support.

20.104 0.21.1 (2015-11-29)

- Fix serious regression to `test` command. 94097c7
- Small fixes to release process. 4e1377c, 94645ed

20.105 0.21.0 (2015-11-29)

- If `virtualenv` not on `PATH`, have Planemo create one for Galaxy. 5b97f2e
- Add documentation section on testing tools installed in an existing Galaxy instance. 1927168
- When creating a virtualenv for Galaxy, prefer Python 2.7. e0577e7
- Documentation fixes and improvements thanks to @martenson. 0f8cb10, 01584c5, b757791
- Specify a minimum `six` version requirement. 1c7ee5b
- Add script to test a planemo as a wheel. 6514ff5, Issue 184
- Fix empty macro loading. Issue 362
- Fix an issue when you run `shed_diff --shed_target local` thanks to Gwendoline Andres and Gildas Le Corguillé at ABiMS Roscoff. Pull Request 375
- Fix `shed_diff` printing to stdout if `-o` isn't specified. f3394e7
- Small `shed_diff` improvements to XML diffing and XUnit reporting. af7448c, 83e227a
- More logging of `shed_diff` results if `--verbose` flagged. 9427b47
- Add `test_report` command for rebuilding reports from structured JSON. 99ee51a
- Fix option bug with Click 6.0 thanks to @bgruening. 2a7c792
- Improved error messages for test commands. fdce74c
- Various fixes for Python 3. 2f66fc3, 7572e99, 8eda729, 764ce01
- Use newer travis container infrastructure for testing. 6d81a94
- Test case fixes. 98fdc8c, 0e4f70a

20.106 0.20.0 (2015-11-11)

- More complete I/O capturing for XUnit. 6409449
- Check for select parameter without options when linting tools. Issue 373
- Add `--cwl_engine` argument to `cwl_run` command. dd94ddc
- Fixes for select parameter linting. 8b31850
- Fix to demultiplexing repositories after tool uploads. Issue 361
- Fix to update planemo for Galaxy wheels. 25ef0d5
- Various fixes for Python 2.6 and Python 3. c1713d2, 916f610, c444855

20.107 0.19.0 (2015-11-03)

- Initial implementation of `cwl_run` command that runs a CWL tool and job file through Galaxy. 49c5c1e
- Add `--cwl` flag to `serve` to experimentally serve CWL tools in Galaxy. Pull Request 339
- Implement highly experimental `cwl_script` command to convert a CWL job to a bash script. 508dce7
- Add name to all XUnit reports (thanks to @hexylena). Pull Request 343
- Capture stdout and stderr for `shed_diff` and `shed_update` XUnit reports. Pull Request 344
- More tool linting (conditionals) thanks to @hexylena. Pull Request 350
- UTF-8 fixes when handling XUnit reports. Pull Request 345
- Add *Epigenetics* as Tool Shed category. Pull Request 351
- Merge changes to common modules shared between Galaxy, Planemo, and Pulsar (thanks to @natefoo). Pull Request 356
- Add `--cite_url` to `tool_init`. fdb1b51
- `tool_init` bug fix. f854138
- Fix setup.py for cwltool and bioblend changes. 1a157d4
- Add option to specify template sqlite database locally. c23569f
- Add example IPython notebooks to docs. c8640b6

20.108 0.18.1 (2015-10-22)

- Fix issue with test reporting not being populated. 19900a6

20.109 0.18.0 (2015-10-20)

- Improvements to `docker_shell` usability (thanks to @kellrott). Pull Request 334
- Add docker pull attempt when missing Dockerfile (thanks to @kellrott). Pull Request 333
- Fix bug inferring which files are tool files (thanks to @hexylena). Pull Request 335, Issue 313
- Initial work toward automating brew recipe update. 4d6f7d9, Issue 329

20.110 0.17.0 (2015-10-19)

- Implement basic XUnit report option for `shed_update` (thanks to @martenson). Pull Request 322
- Fix issues with producing test outputs. 572e754
- Xunit reporting improvements - refactoring, times, diff output (thanks to @hexylena). Pull Request 330
- Implement project governance policy and update developer code of conduct to match that of the Galaxy project. Pull Request 316
- Update filters for account for new `.txt` and `.md` test outputs (thanks to @hexylena). Pull Request 327
- Add verbose logging to galaxy test output handling problems. 5d7db92

- Flake8 fixes (thanks to @martenson). 949a36d
- Remove uses of deprecated `mktemp` Python standard library function (thanks to @hexylena). Pull Request 330

20.111 0.16.0 (2015-10-07)

- Adding new command `dependency_script` to convert Tool Shed dependencies into shell scripts - thanks to @peterjc. Pull Request 310, f798c7e, Issue 303
- Implement profiles in sheds section of the `~/.planemo.yml`. Pull Request 314

20.112 0.15.0 (2015-10-01)

- Template framework for reporting including new markdown and plain text reporting options for testing - thanks to @hexylena. Pull Request 304
- XUnit style reporting for `shed_diff` command - thanks to @hexylena. Pull Request 305
- Add new `shed_build` command for building repository tarballs - thanks to @kellrott. Pull Request 297
- Fix exit code handling for `lint` commands - thanks to @mvdbeek. Pull Request 292
- Improved documentation for `serve` command - thanks to @lparsons. Pull Request 312
- Tiny backward compatible Python 3 tweaks for Tool Factory - thanks to @peterjc. dad2d9d
- Fixed detection of virtual environment in `Makefile` - thanks to @lparsons. Pull Request 311
- Updates to Galaxy XSD - thanks to @mr-c. Pull Request 309
- Allow reading shed key option from an environment variable. Pull Request 307
- Allow specifying host to serve Galaxy using `-host` - thanks in part to @chambm. Pull Request 301
- Allow specifying defaults for `-host` and `--port` in `~/.planemo.yml`. Pull Request 301
- Improve `~/.planemo.yml` sample comments - thanks to @martenson. Pull Request 287
- Update tool shed categories - thanks to @bgruening. Pull Request 285
- Improved output readibility for `diff` command - thanks to @martenson. Pull Request 284

20.113 0.14.0 (2015-08-06)

- Allow `-t` as shorthand for `--shed_target` (thanks to Peter Cock). Pull Request 278
- Fix `tool_init` command to use `from_work_dir` only if file in command (thanks to bug report and initial fix outline by Gildas Le Corguillé). Pull Request 277
- Various documentation fixes (thanks in part to Peter Cock and Daniel Blankenberg). Pull Request 256, Pull Request 253, Pull Request 254, Pull Request 255, Pull Request 251, Issue 272

20.114 0.13.2 (2015-07-06)

- Fix project_init for missing files. cb5b906
- Various documentation improvements.

20.115 0.13.1 (2015-07-01)

- Fix for `shed_init` producing non-standard type hints. Issue 243, f0610d7
- Fix tool linting for parameters that define an `argument` but not a `name`. Issue 245, aad1eed
- Many doc updates including a tutorial for developing tools in a test-driven fashion and instructions for using the planemo appliance through Kitematic (with Kitematic screenshots from E. Rasche).

20.116 0.13.0 (2015-06-28)

- If planemo cannot find a Galaxy root, it will now automatically fetch one (specifing `--galaxy_install` will still force a fetch). Pull Request 235
- Docuementation has been updated to reflect new and vastly improved Docker and Vagrant virtual appliances are now available, as well as a new VirtualBox OVA variant.
- Update linting for new tool XML features (including `detect_errors` and output collections). Issue 233, 334f2d4
- Fix `shed_test` help text. Issue 223
- Fix code typo (thanks to Nicola Soranzo). Pull Request 230
- Improvements to algorithm used to guess if an XML file is a tool XML file. Issue 231
- Fix configuration file handling bug. Issue 240

20.117 0.12.2 (2015-05-23)

- Fix `shed_test` and `shed_serve` for test and local tool sheds. f3cafaa

20.118 0.12.1 (2015-05-21)

- Fix to ensure the tab completion script is in the Python source tarball (required for setting up tab-completion for Homebrew). 6b4e7a6

20.119 0.12.0 (2015-05-21)

- Implement a `--failed` flag for the `test` command to rerun previously faied tests. Pull Request 210
- Implement `shed_update` to upload contents and update repository metadata. Pull Request 216
- Implement `shed_test` and `shed_serve` commands to test and view published artifacts in the Tool Shed. Pull Request 213, Issue 176
- Add shell tab-completion script. 37dcc07
- Many more commands allow specifing multiple tool and/or repository targets. Issue 150
- Add -m as alias for –message in planemo shed_upload (thanks to Peter Cock). Pull Request 200
- Add `--ensure_metadata` option to `shed_lint` to ensure `.shed.yml` files contain many repository. Pull Request 215
- More developer documentation, additional `make` targets including ones for setting up git pre-commit hooks. cc8abb6, Issue 209
- Small README improvement (thanks to Martin Čech) b53006d
- Fixes for shed operation error handling (thanks to Martin Čech). Pull Request 203, Pull Request 206
- Fix for "smart" `shed_diff` not in the repository root directory (thanks to Peter Cock). Pull Request 207, Issue 205
- Recursive `shed_diff` with directories not yet in Tool Shed. Pull Request 208
- Improve error handling and reporting for problematic `--shed_target` values. Issue 217
- Fix typos in lint messages. Issue 211

20.120 0.11.1 (2015-05-12)

- Fix default behavior for `planemo lint` to use current directory if explicit paths are not supplied. 1e3668a

20.121 0.11.0 (2015-05-12)

- More compact syntax for defining multiple custom inclusions in `.shed.yml` files - thanks to Peter Cock. Issue 180, Pull Request 185, Pull Request 196
- Prevent ambigous destinations when defining custom inclusions in `.shed.yml`- thanks to Peter Cock. Pull Request 186
- `lint` now warns if tool ids contain whitespace. Pull Request 190
- Handle empty tar-balls gracefully on older Python versions - thanks to Peter Cock. Pull Request 187
- Tweak quoting in `cp` command - thanks to Peter Cock. 6bcf699
- Fix regression causing testing to no longer produce "pretty" test results under certain circumstances. Issue 188
- Fix for recursive `shed_diff` folder naming. Issue 192
- Fix output definitions to `tool_init` command. Issue 189

20.122 0.10.0 (2015-05-06)

- Extend `shed_lint` to check for valid actions in tool_dependencies.xml files. 8117e03
- Extend `shed_lint` to check for required files based on repository type. Issue 156
- Ignore common editor backup files during `shed_upload`. Issue 179
- Fix missing file when installing from source via PyPI. Issue 181
- Fix `lint` to verify `data` inputs specify a `format` attribute. 8117e03
- Docstring fix thanks to @peterjc. fe7ad46

20.123 0.9.0 (2015-05-03)

- Add new logo to the README thanks to @petrkadlec from puradesign.cz and @carlfeberhard from the Galaxy Project. Issue 108
- Implement smarter `shed_diff` command - it now produces a meaningful exit codes and doesn't report differences if these correspond to attributes that will be automatically populated by the Tool Shed. Issue 167
- Use new smarter `shed_diff` code to implement a new `--check_diff` option for `shed_upload` - to check for meaningful differences before updating repositories. Issue 168
- Record git commit hash during `shed_upload` if the `.shed.yml` is located in a git repository. Issue 170
- Allow `shed_` operations to operate on git URLs directly. Issue 169
- Fail if missing file inclusion statements encountered during `.shed.yml` repository resolution - bug reported by @peterjc. Issue 158
- Improved exception handling for tool shed operations including new `--fail_fast` command-line option. * Issue 114, Pull Request 173
- Implement more validation when using the `shed_init` command. 1cd0e2d
- Add `-r/--recursive` option to `shed_download` and `shed_diff` commands and allow these commands to work with `.shed.yml` files defining multipe repositories. 40a1f57
- Add `--port` option to the `serve` and `tool_factory` commands. 15804be
- Fix problem introduced with setup.py during the 0.9.0 development cycle - thanks to @peterjc. Pull Request 171
- Fix clone bug introduced during 0.9.0 development cycle - thanks to @bgruening. Pull Request 175

20.124 0.8.4 (2015-04-30)

- Fix for Travis CI testing picking up invalid tests (reported by @takadonet). Issue 161
- Fix tar ordering for consistency (always sort by name) - thanks to @peterjc. Pull Request 164, Issue 159
- Fix exception handling related to tool shed operations - thanks to @peterjc. Pull Request 155, b86fe1f

20.125 0.8.3 (2015-04-29)

- Fix bug where `shed_lint` was not respecting the `-r/--recursive` flag. 9ff0d2d
- Fix bug where planemo was producing tar files incompatible with the Tool Shed for package and suite repositories. a2ee135

20.126 0.8.2 (2015-04-29)

- Fix bug with `config_init` command thanks to @bgruening. Pull Request 151
- Fix unnessecary `lint` warning about `parallelism` tag reported by @peterjc. 9bf1eab

20.127 0.8.1 (2015-04-28)

- Fixes for the source distribution to allow installation of 0.8.0 via Homebrew.

20.128 0.8.0 (2015-04-27)

- Implement the new `shed_lint` command that verifies various aspects of tool shed repositories - including XSD validation of `repository_dependencies.xml` and `tool_dependencies.xml` files, best practices for README files, and the contents of `.shed.yml` files. This requires the lxml library to be available to Planemo or the application xmllint to be on its `PATH`. Pull Request 130 Issue 89 Issue 91 912df02 d26929e 36ac6d8
- Option to enable experimental XSD based validation of tools when `lint` is executed with the new `--xsd` flag. This validation occurs against the unofficial Galaxy Tool XSD project maintained by @JeanFred. This requires the lxml library to be available to Planemo or the application xmllint to be on its `PATH`. Pull Request 130 912df02
- Allow skipping specific linters when using the `lint` command using the new `--skip` option. 26e3cdb
- Implement sophisticated options in `.shed.yml` to map a directory to many, custom Tool Shed repositories during shed operaitons such `shed_upload` including automatically mapping tools to their own directories and automatically building suites repositories. Pull Request 143
- Make `shed_upload` more intelligent when building tar files so that package and suite repositories may have README files in source control and they will just be filtered out during upload. 53edd99
- Implement a new `shed_init` command that will help bootstrap `.shed.yml` files in the specified directory. cc1a447
- Extend `shed_init` to automatically build a `repository_rependencies.xml` file corresponding to a Galaxy workflow (`.ga` file). Issue 118 988de1d
- In addition to a single file or directory, allow `lint` to be passed multiple files. 343902d Issue 139
- Add `-r/--recursive` option to `shed_create` and `lint` commands. 63cd431 01f2af9
- Improved output formatting and option to write diffs to a file for the `shed_diff` command. 965511d
- Fix lint problem when using new Galaxy testing features such as expecting job failures and verifing job output. Issue 138
- Fix typo in `test` help thanks to first time contributor @pvanheus. Pull Request 129 1982076
- Fix NPE on empty `help` element when linting tools. Issue 124

- Fix `lint` warnings when `configfiles` are defined in a tool. 1a85493
- Fix for empty `.shed.yml` files. b7d9e96
- Fix the `test` command for newer versions of nose. 33294d2
- Update help content and documentation to be clear `normalize` should not be used to update the contents of tool files at this time. 08de8de
- Warn on unknown `command` attributes when linting tools (anything but `interpreter`). 4f61025
- Various design, documentation (including new documentation on Tool Shed publishing), and testing related improvements (test coverage has risen from 65% to over 80% during this release cycle).

20.129 0.7.0 (2015-04-13)

- Implement *shed_create* command to create Tool Shed repositories from `.shed.yml` files (thanks to E. Rasche). Pull Request 101
- Allow automatic creation of missing repositories during `shed_upload` with the new `--force_repository_creation` flag (thanks to E. Rasche). Pull Request 102
- Allow specifying files to exclude in `.shed.yml` when creating tar files for `shed_upload` (thanks to Björn Grüning). Pull Request 99
- Resolve symbolic links when building Tool Shed tar files with `shed_upload` (thanks to Dave Bouvier). Pull Request 104
- Add a Contributor Code of Conduct. Pull Request 113
- Omit `tool_test_output.json` from Tool Shed tar file created with `shed_upload` (thanks to Dave Bouvier). Pull Request 111
- Update required version of bioblend to `0.5.3`. Fixed Issue 88.
- Initial work on implementing tests cases for Tool Shed functionality. 182fe57
- Fix incorrect link in HTML test report (thanks to Martin Čech). 4c71299
- Download Galaxy from the new, official Github repository. 7c69bf6
- Update travis_test to install stable planemo from PyPI. 39fedd2
- Enable caching on `--install_galaxy` by default (disable with `--no_cache_galaxy`). d755fe7

20.130 0.6.0 (2015-03-16)

- Many enhancements to the tool building documentation - descriptions of macros, collections, simple and conditional parameters, etc…
- Fix `tool_init` to quote file names (thanks to Peter Cock). Pull Request 98.
- Allow ignoring file patterns in `.shed.yml` (thanks to Björn Grüning). Pull Request 99
- Add `--macros` flag to `tool_init` command to generate a macro file as part of tool generation. ec6e30f
- Add linting of tag order for tool XML files. 4823c5e
- Add linting of `stdio` tags in tool XML files. 8207026
- More tests, much higher test coverage. 0bd4ff0

20.131 0.5.0 (2015-02-22)

- Implement `--version` option. Issue 78
- Implement `--no_cleanup` option for `test` and `serve` commands to persist temp files. 2e41e0a
- Fix bug that left temp files undeleted. Issue 80
- More improvements to release process. fba3874

20.132 0.4.2 (2015-02-21)

- Fix setup.py for installing non-Python data from PyPI (required newer for `tool_factory` command and reStructuredText linting). Thanks to Damion Dooley for the bug report. Issue 83

20.133 0.4.1 (2015-02-16)

- Fix README.rst so it renders properly on PyPI.

20.134 0.4.0 (2015-02-16)

- Implement `tool_init` command for bootstrapping creation of new tools (with tutorial.) 78f8274
- Implement `normalize` command for reorganizing tool XML and macro debugging. e8c1d45
- Implement `tool_factory` command to spin up Galaxy pre-configured the Tool Factory. 9e746b4
- Added basic linting of `command` blocks. b8d90ab
- Improved linting of `help` blocks, including verifying valid *reStructuredText*. 411a8da
- Fix bug related to `serve` command not killing Galaxy properly when complete. 53a6766
- Have `serve` command display tools at the top level instead of in shallow sections. badc25f
- Add additional dependencies to `setup.py` more functionality works out of the box. 85b9614
- Fix terrible error message related to bioblend being unavailable. Issue 70
- Various smaller documentation and project structure improvements.

20.135 0.3.1 (2015-02-15)

- Fixes to get PyPI workflow working properly.

20.136 0.3.0 (2015-02-13)

- Add option (`-r`) to the `shed_upload` command to recursively upload subdirectories (thanks to E. Rasche). Pull Request 68
- Fix diff formatting in test reports (thanks to E. Rasche). Pull Request 63
- Grab updated test database to speed up testing (thanks to approach from E. Rasche and Dannon Baker). Issue 61, dff4f33
- Fix test data command-line argument name (was `test-data` now it is `test_data`). 834bfb2
- Use `tool_data_table_conf.xml.sample` file if `tool_data_table_conf.xml.test` is unavailable. Should allow some new tools to be tested without modifying Galaxy's global `tool_data_table_conf.xml` file. ac4f828

20.137 0.2.0 (2015-01-13)

- Improvements to way Planemo loads its own copy of Galaxy modules to prevent various conflicts when launching Galaxy from Planemo. Pull Request 56
- Allow setting various test output options in `~/.planemo.yml` and disabling JSON output. 21bb463
- More experimental Brew and Tool Shed options that should not be considered part of Planemo's stable API. See bit.ly/gxbrew1 for more details.
- Fix `project_init` for BSD tar (thanks to Nitesh Turaga for the bug report.) a4110a8
- Documentation fixes for tool linting command (thanks to Nicola Soranzo). Pull Request 51

20.138 0.1.0 (2014-12-16)

- Moved repository URL to https://github.com/galaxyproject/planemo.
- Support for publishing to the Tool Shed. Pull Request 6
- Support for producing diffs (`shed_diff`) between local repositories and the Tool Shed (based on scripts by Peter Cock). Pull Request 33
- Use tool's local test data when available - add option for configuring `test-data` target. Pull Request 1
- Support for testing tool features dependent on cached data. 44de95c
- Support for generating XUnit tool test reports. 82e8b1f
- Prettier HTML reports for tool tests. 05cc9f4
- Implement `share_test` command for embedding test result links in pull requests. Pull Request 40
- Fix for properly resolving links during Tool Shed publishing (thanks to Dave Bouvier). Pull Request 29
- Fix for citation linter (thanks to Michael Crusoe for the bug report). af39061
- Fix tool scanning for tool files with fewer than 10 lines (thanks to Dan Blankenberg). a2c13e4
- Automate more of Travis CI testing so the scripts added to tool repository can be smaller. 20a8680
- Documentation fixes for Travis CI (thanks to Peter Cock). Pull Request 22, Pull Request 23
- Various documentation fixes (thanks to Martin Čech). 36f7cb1, b9232e5

- Various smaller fixes for Docker support, tool linting, and documentation.

20.139 0.0.1 (2014-10-04)

- Initial work on the project - commands for testing, linting, serving Galaxy tools - and more experimental features involving Docker and Homebrew. 7d07782

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- modindex
- search