How-to admin Galaxy

Tip: press P to view the presenter notes

Questions

- What is galaxy?
- Can I put my tool in galaxy?
- What are the options for using Galaxy?
- Should we have a local galaxy lab/EPFL?

Objectives

- Discover galaxy
- See if/how you can add your tool
- Discuss about galaxy at EPFL

Before starting

Package, dependency and environment management

Conda Terminology

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

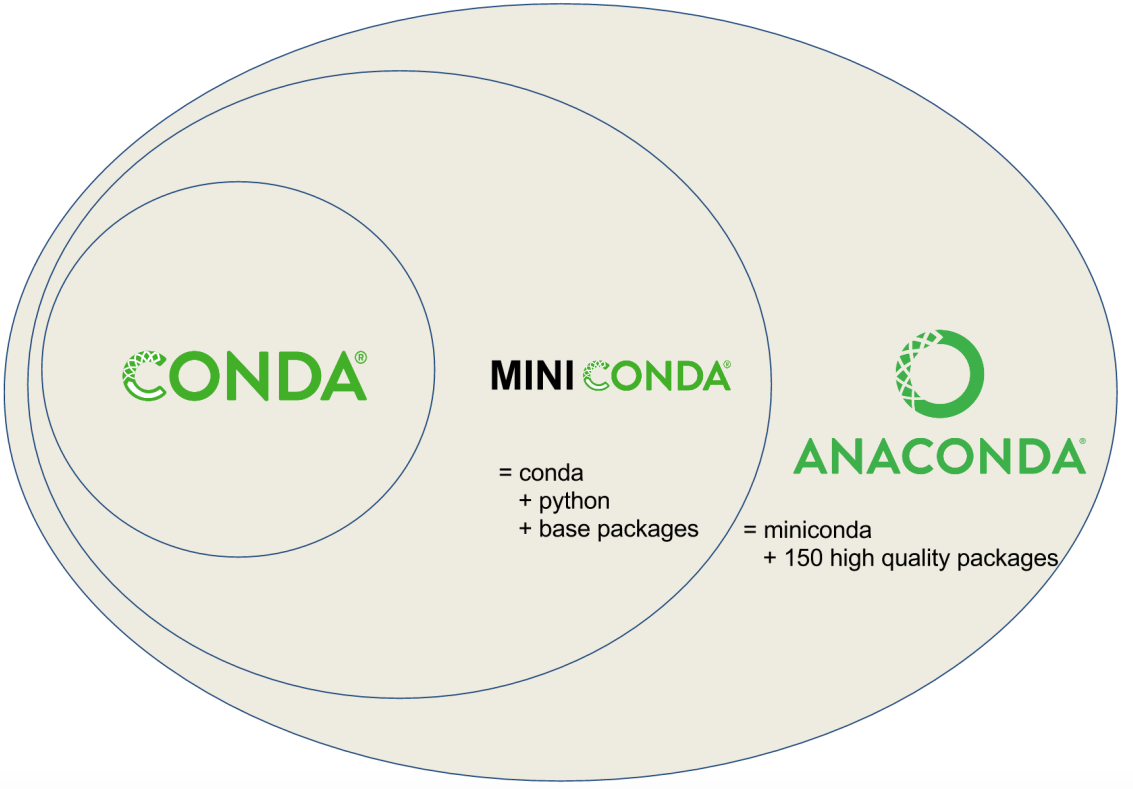
- **recipes** = flat directory with `meta.yml` a file with metadata, `build.sh` or `bld.bat` the script that install the package
- **packages** = software / library etc...
- **channel** = repository for compiled packages.

Conda Key Features

- No compilation at install time - *binaries* with their dependencies, 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
- Vibrant Communities

CONDA

Conda Distributions



- Useful channels which are not added by default:
  - conda-forge
  - bioconda

To add them:

```
conda config --add channel bioconda --add channel conda-forge
```

Using Conda

- Install some packages within an isolated environment

```
$ conda create -n planemo61 planemo=0.61.0
$ which planemo
$ conda env list
planemo61          /home/ldelisle/.conda/envs/planemo61
base            *  /opt/miniconda3
$ conda activate planemo61
(planemo61) $ which planemo
(planemo61) $ conda install myPreferedSoftware
```

What is Galaxy?

Galaxy

- **Web-based** platform for computational biomedical research (analysis and data integration)
  - Developed at Penn State, Johns Hopkins, OHSU and Cleveland Clinic with substantial outside contributions
  - **Open source** under Academic Free License
- More than 7,500 citations
- More than 150 public Galaxy resources
  - 120+ public servers, many more non-public
  - Both general-purpose and domain-specific

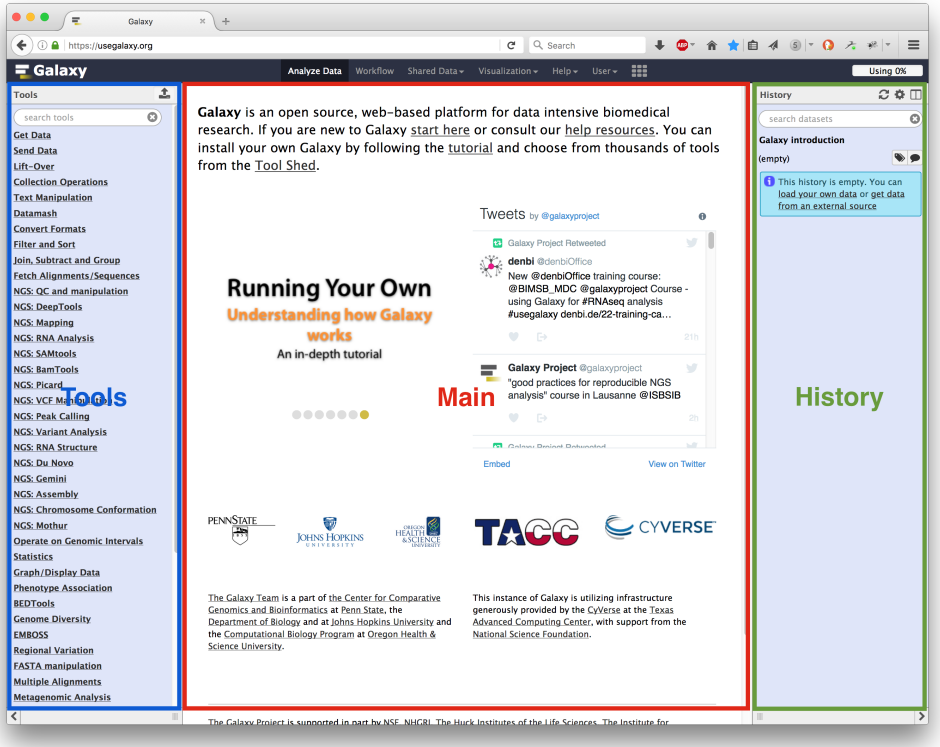
Core values

- **Accessibility**
  - Users without programming experience can easily upload/retrieve data, run complex tools and workflows, and visualize data
- **Reproducibility**
  - Galaxy captures information so that any user can understand and repeat a complete computational analysis
- **Transparency**
  - Users can share or publish their analyses (histories, workflows, visualizations)
  - Users can generate Pages: online Methods for your paper

User Interface

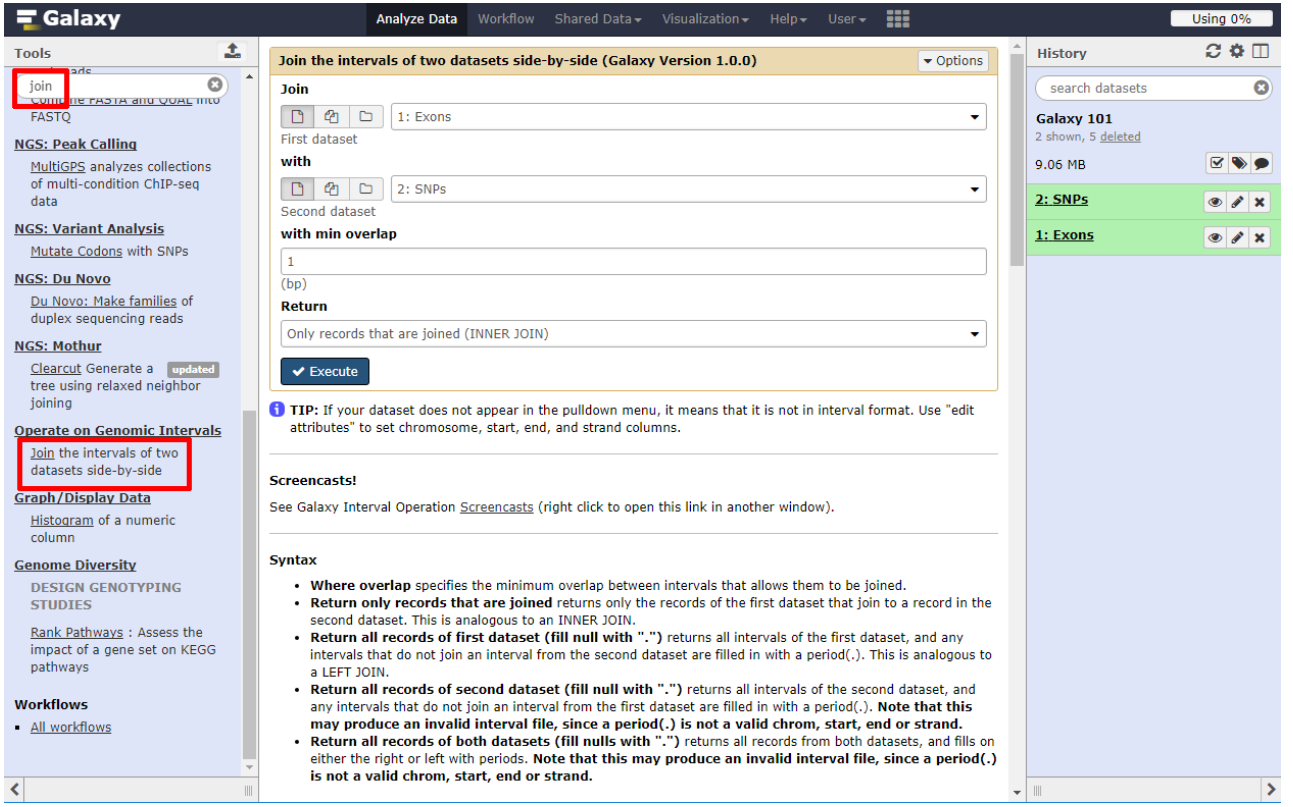
Visit https://usegalaxy.eu

Main Galaxy interface



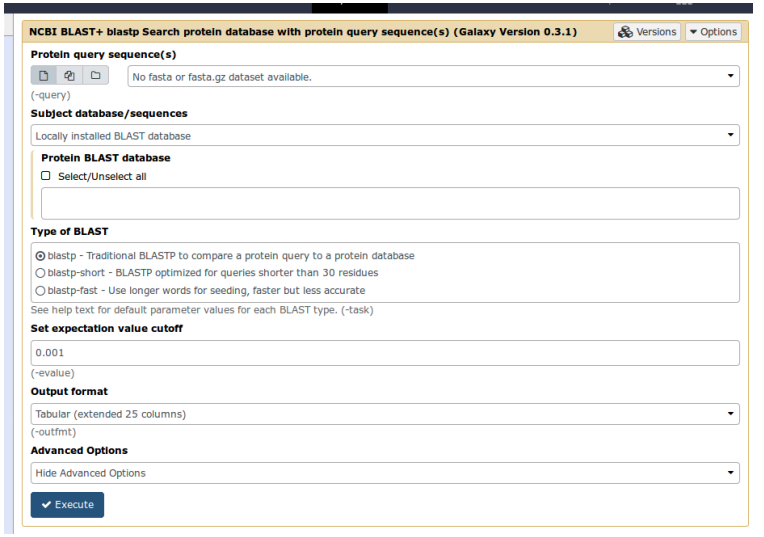
Home page divided into 3 panels

Tools



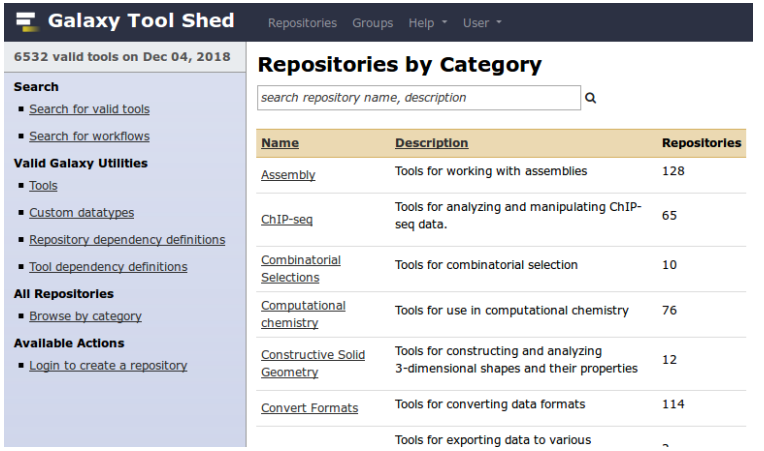
- The tool search helps in finding a tool in a crowded toolbox

Tool interface



- A tool form contains:
  - input datasets and parameters
  - help, citations, metadata
  - an `Execute` button to start a job, which will add some output datasets to the history
- New tool versions can be installed without removing old ones to ensure reproducibility

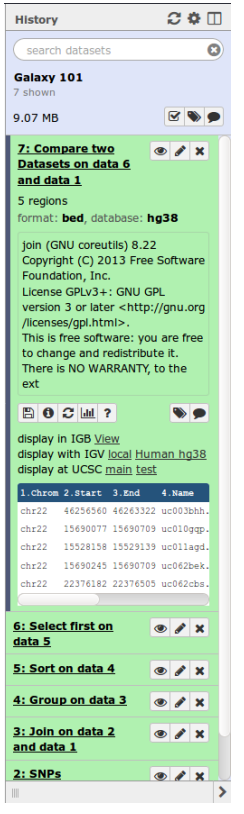
Tool Shed



- Free "app" store: Galaxy Tool Shed
  - Thousands of tools already available
  - Most software can be integrated
    - If a tool is not available, ask the Galaxy community for help or write it yourself!
  - Only a Galaxy admin can install tools

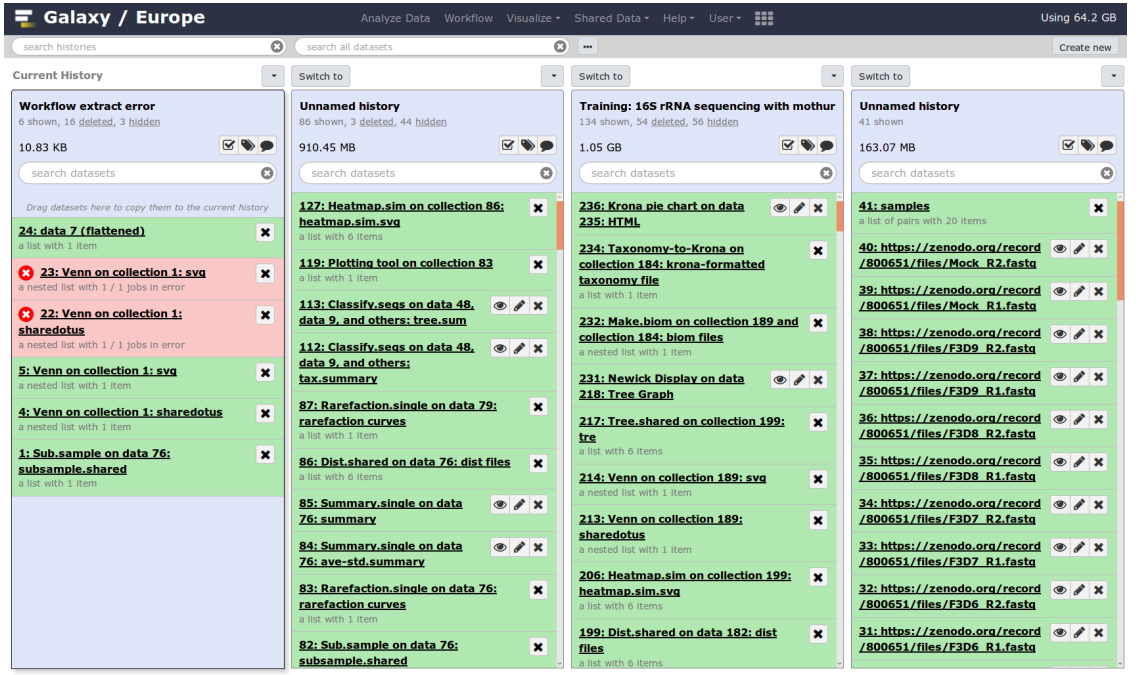
History

- Location of all analyses
  - collects all datasets produced by tools
  - collects all operations performed on the data
- For each dataset (the heart of Galaxy's reproducibility), the history tracks
  - name, format, size, creation time, datatype-specific metadata
  - tool id, version, inputs, parameters
  - standard output (`stdout`) and error (`stderr`)
  - state (waiting, running, success, failed)
  - hidden, deleted, purged



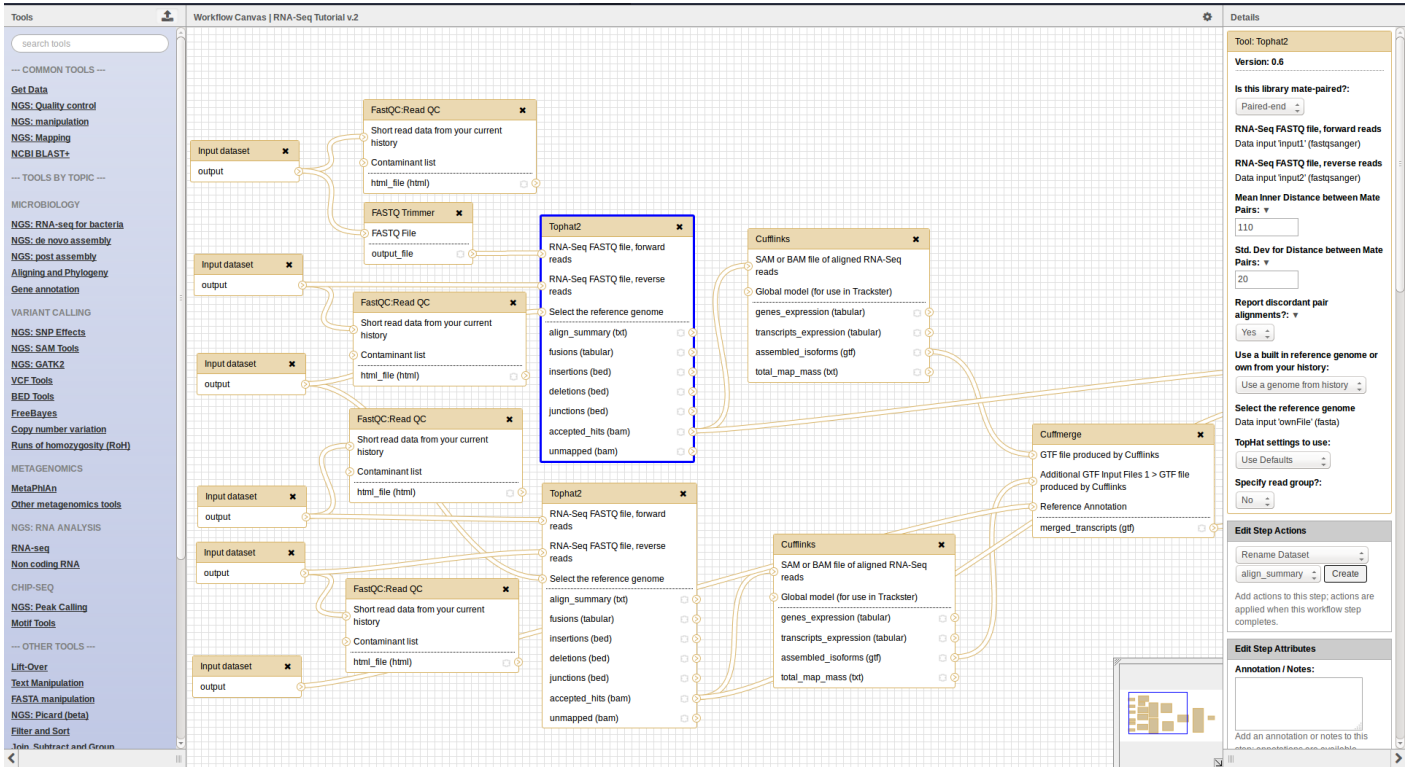
Multiple histories

- You can have as many histories as you want
  - each history should correspond to a **different analysis**
  - and should have a meaningful **name**



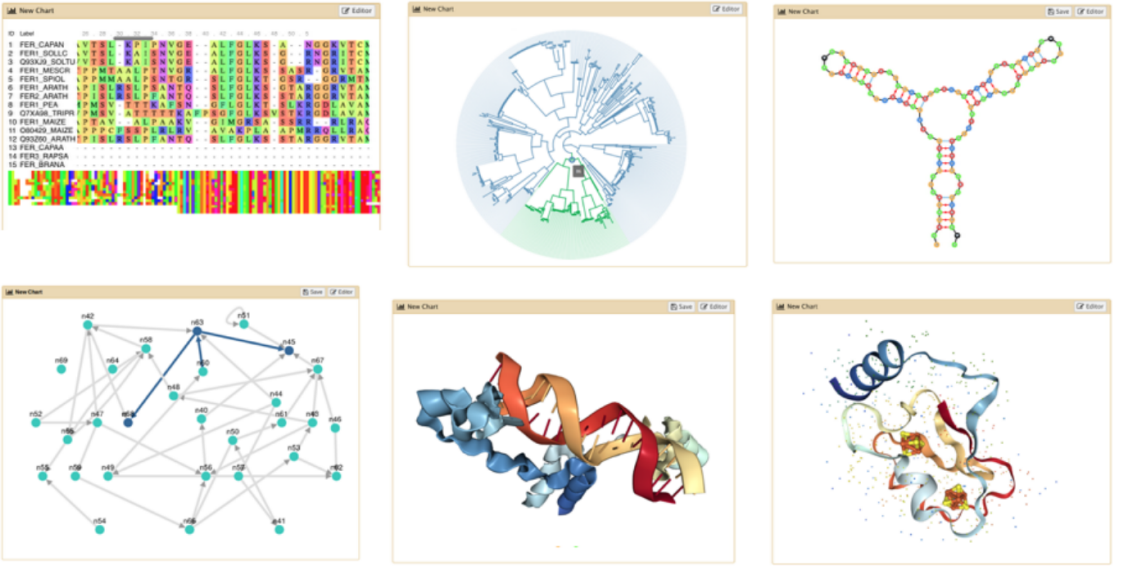
Workflows

Workflow Editor



- **Extracted** from a history
- **Built manually** by adding and configuring tools using the canvas
- **Imported** using an existing shared workflow

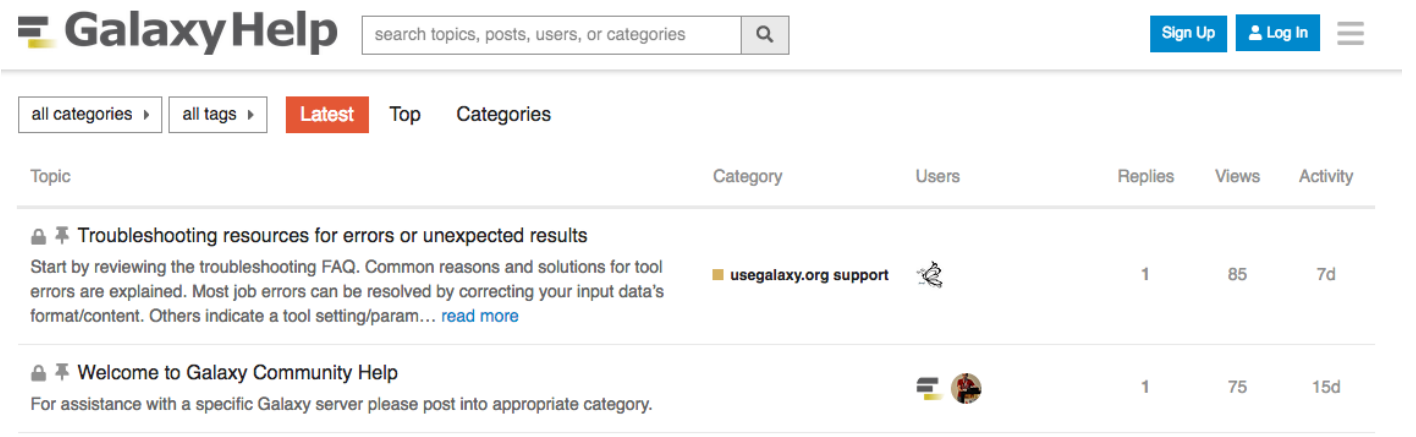
Visualizations



- Datatypes know what tools can be used to visualize datasets:
  - Sequencing data has a button for visualizing in IGV/UCSC
  - Tabular data will prompt you to build charts
  - Protein data can be seen in a 3D viewer
- Interactive environments: Jupyter, RStudio, etc

Community

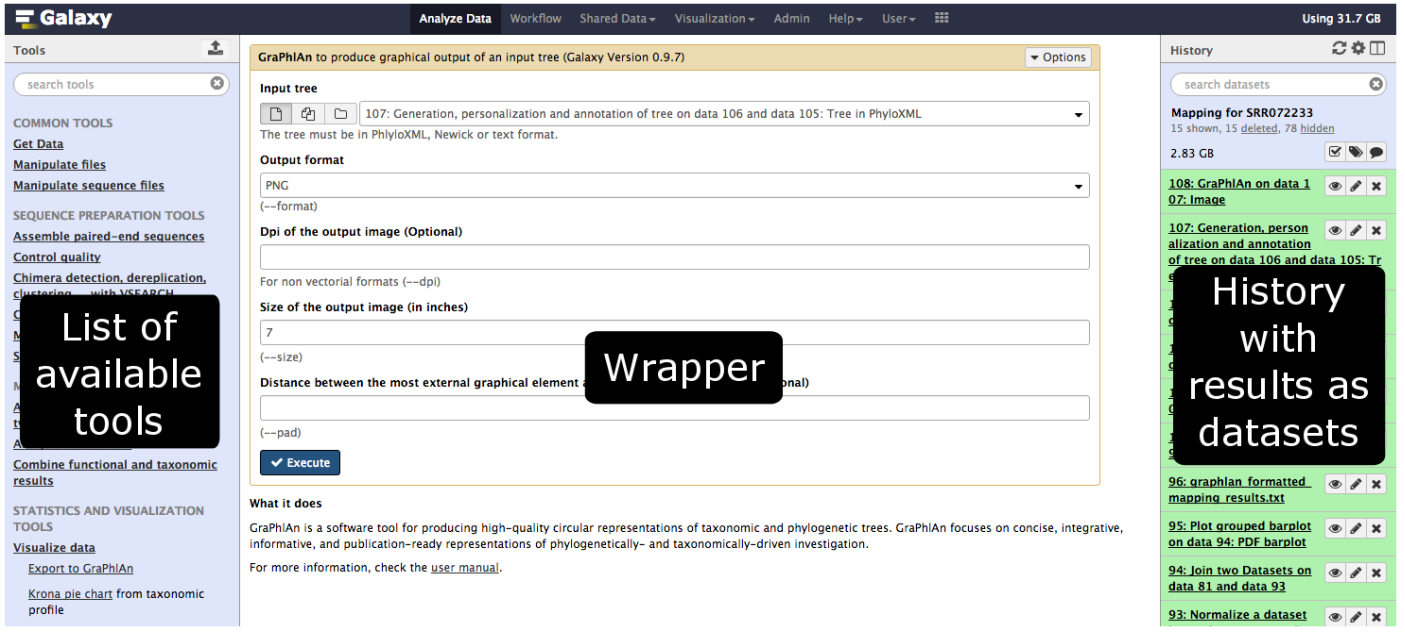
- Support forum: Galaxy Help



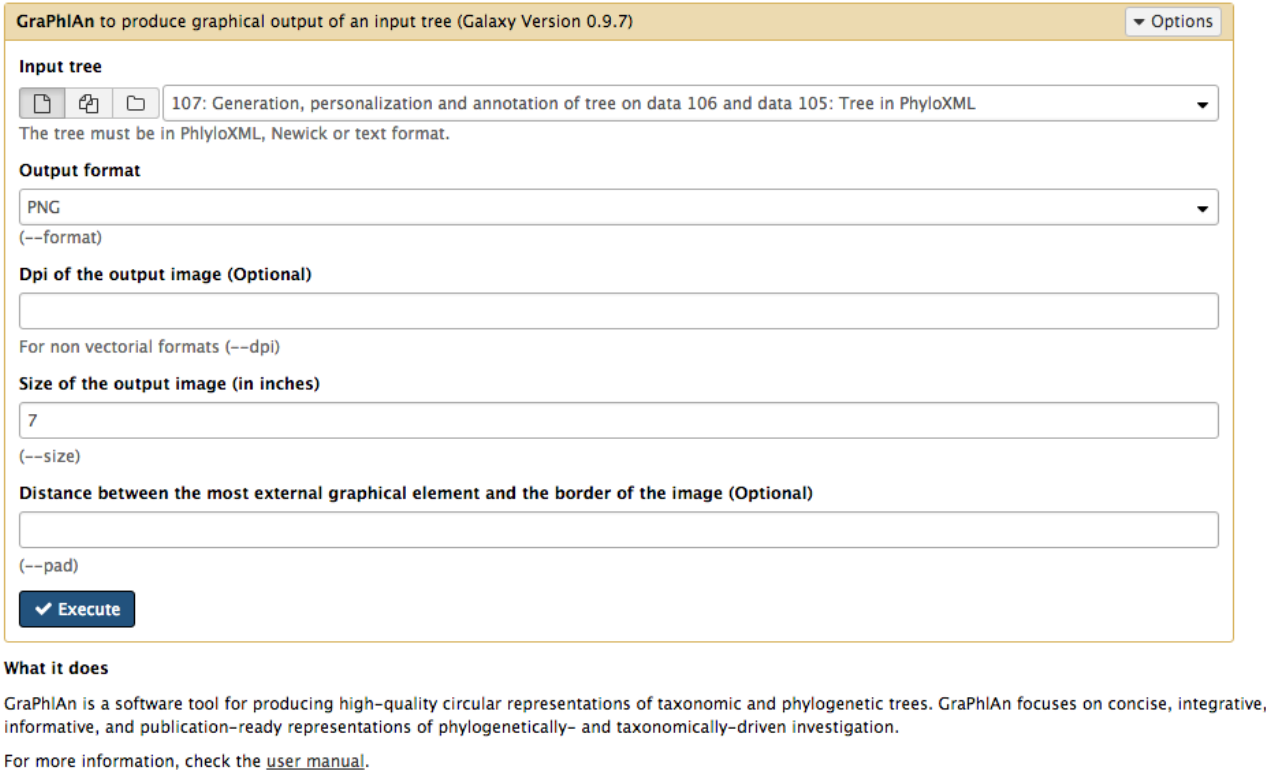
- Community curated documentation: Galaxy Community Hub
- Events all around the world
- Galaxy Training for scientists, developers, admins, instructors: Galaxy Training Community
  - Training questions? Chat with us on Gitter

Galaxy tools

A tool in Galaxy



Galaxy tool / wrapper

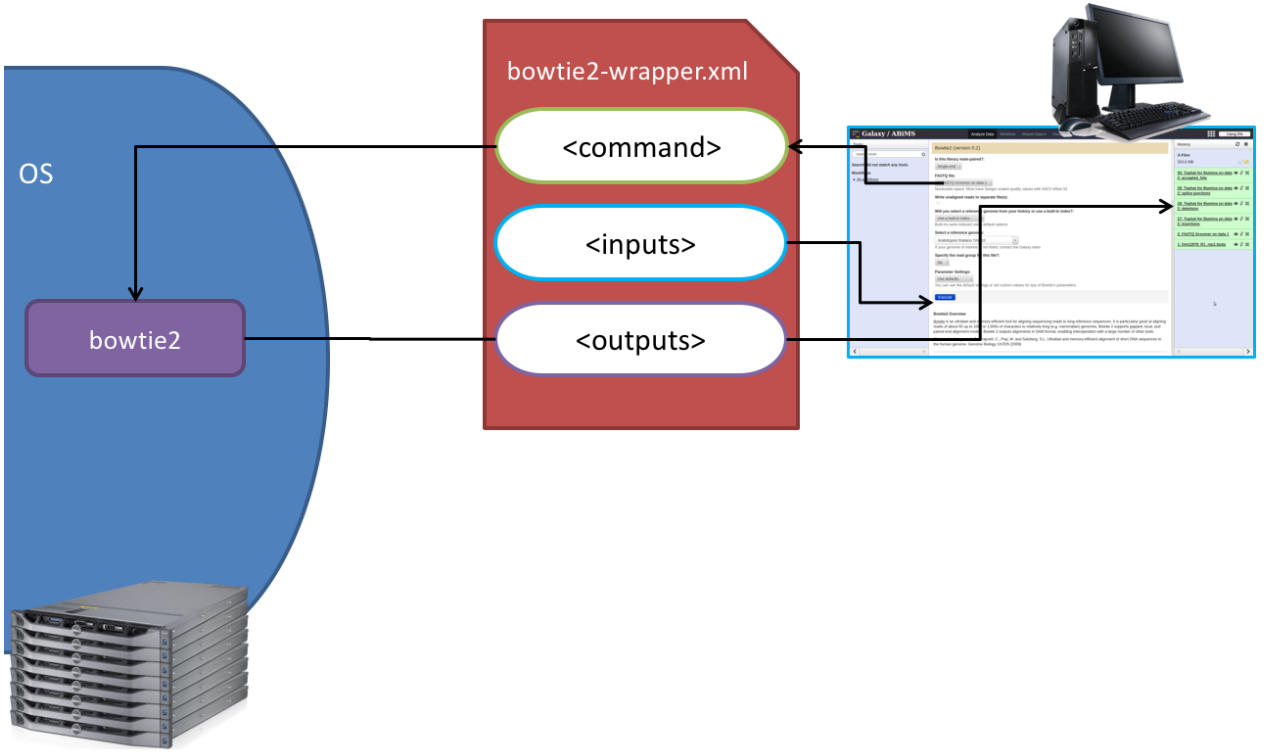



```
graphlan.py --format "png" --size 7 input_tree.txt png_image.png
```

Wrapper

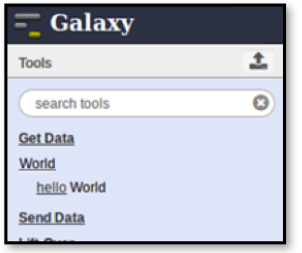
- Description of the user interface
- Link between the underlying tool and the Galaxy instance
- How to invoke the tool
- Which files and options to pass
- Which files the tool will produce as output

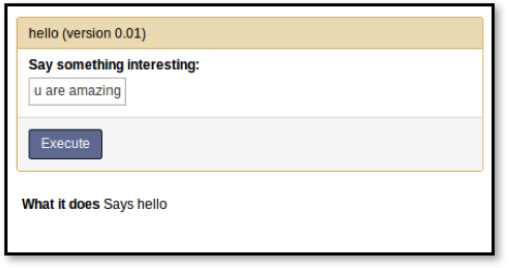
Wrapper


OS
bowtie2
bowtie2-wrapper.xml
<command>
<inputs>
<outputs>
Galaxy / ABIMS

Wrapper

```
<tool id="hello" name="hello" version="0.01">
    <description>World</description>
    <command><![CDATA[
        /bin/echo 'Hello World! $mystring' > '$output1'
    ]]></command>
    <inputs>
        <param name="mystring" type="text" label="Say something interesting"/>
    </inputs>
    <outputs>
        <data format="tabular" name="output1" label="hello_world"/>
    </outputs>
    <help><![CDATA[
        **What it does**
        Says hello
    ]]></help>
</tool>
```





Wrapper

```
<tool id="hello" name="hello" version="0.01">
    <description>World</description>
    <command><![CDATA[
        /bin/echo 'Hello World! $mystring' > '$output1'
    ]]></command>
    <inputs>
        <param name="mystring" type="text" label="Say something interesting"/>
    </inputs>
    <outputs>
        <data format="tabular" name="output1" label="hello_world"/>
    </outputs>
    <help><![CDATA[
        **What it does**
```

Job Command-Line:

```
/bin/echo 'Hello World You are amazing' > '/usr/local/galaxy/database/files/000/dataset_9.dat'
```

```
</tool>
```

Wrapper

```
<tool id="hello" name="hello" version="0.01">
    <description>World</description>
    <command><![CDATA[
        /bin/echo 'Hello World! $mystring' > '$output1'
    ]]></command>
    <inputs>
        <param name="mystring" type="text" label="Say something interesting"/>
    </inputs>
    <outputs>
        <data format="tabular" name="output1" label="hello_world"/>
    </outputs>
    <help><![CDATA[
        **What it does**
        Says hello
    ]]></help>
</tool>
```

Hello World! You are amazing

Wrapper

Galaxy tool XML format is formally defined in a XML Schema Definition (XSD), used to generate the relative online documentation

Today, I just give you the bases to be able to make simple tools.

command

How to invoke the tool?

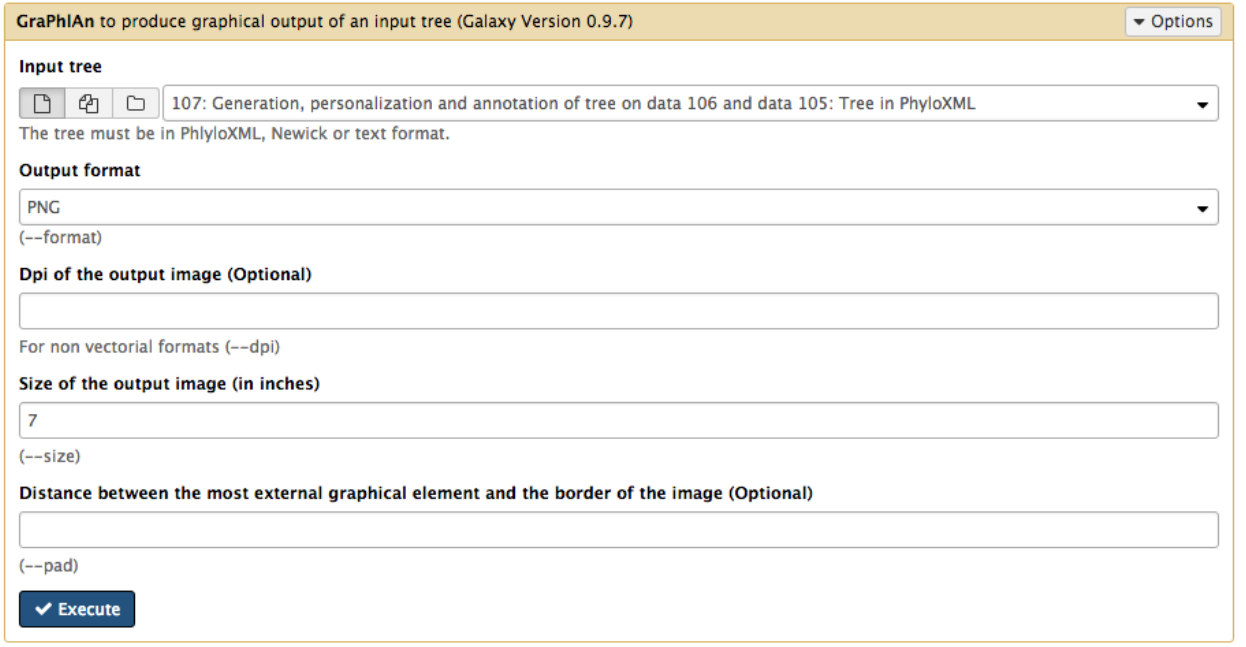
```
<command><![CDATA[
graphlan.py
--format $format
#if str($dpi):
    --dpi $dpi
#end if
--size $size
'$input_tree'
]]></command>
```

If the script is provided with the wrapper xml

```
<command><![CDATA[
python '$__tool_directory__/graphlan.py'
# command params...
]]></command>
```

command

How to pass the parameters?



inputs > param to command

How to pass the parameters?

```
<inputs>
    <param name="input_tree" type="data" label="..."/>

    <param argument="--dpi" type="integer" optional="true" label="..."
        help="For non vectorial formats" />
</inputs>
```

inputs > param to command

Directly linked to <command>

```
<command><![CDATA[
graphlan.py
...
#if str($dpi):
    --dpi $dpi
#end if
'$input_tree'
...
]]></command>
```

inputs > param > data

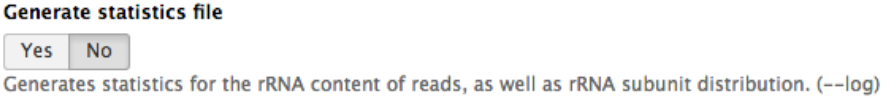

```
<param name="..." type="data" format="txt" label="..." help="..." />
```

List of possible formats

inputs > param > integer | float | text | boolean

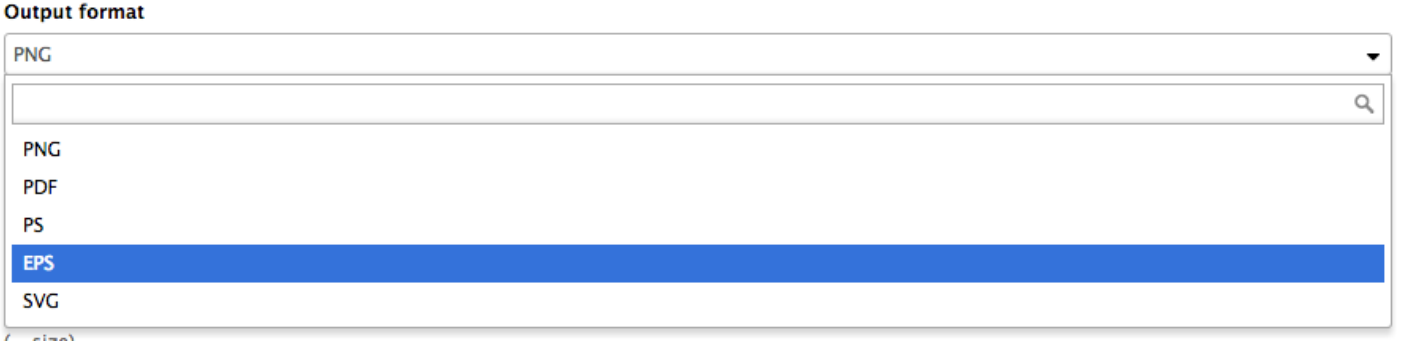

```
<param name="..." type="float" min="0" max="10" value="1" label="..."
    help="..."/>
```


```
<param name="..." type="text" value="..." label="..." help="..."/>
```




```
<param name="..." type="boolean" checked="false" truevalue="--log" falsevalue=""
    label="..." help="..." />
```

inputs > param > select

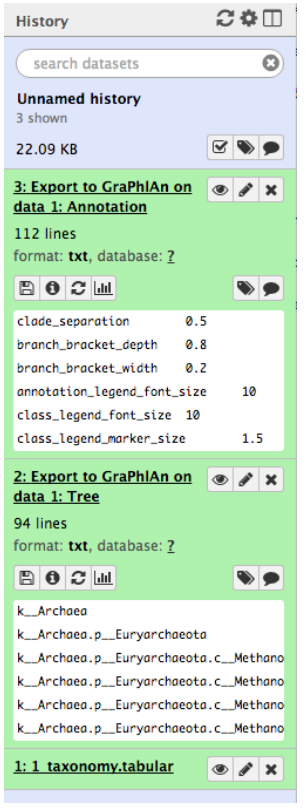



```
<param name="..." type="select" label="..." help="...">
    <option value="png" selected="true">PNG</option>
    <option value="pdf">PDF</option>
    <option value="ps">PS</option>
    <option value="eps">EPS</option>
    <option value="svg">SVG</option>
</param>
```

If no `option` has `selected="true"`, the first one is selected by default.

outputs

Which files the tool will produce as output?



outputs

```
<outputs>
    <data name="tree" format="txt" label="${tool.name} on ${on_string}: Tree" />
    <data name="annotation" format="txt"
        label="${tool.name} on ${on_string}: Annotation" />
</outputs>
```

Tips and best practices

Use `&&` to concatenate multiple commands

```
<command><![CDATA[
graphlan.py
--format '$format'
&&
echo "Yeah it worked!"
]]></command>
```

The job will exit on the first error encountered.

PLANEMO

Command-line utilities to assist in building and publishing Galaxy tools.

- Documentation
- Tutorial

PLANEMO

planemo tool_init

Creates a skeleton of xml file

```
$ mkdir new_tool
$ cd new_tool
$ planemo tool_init --id 'some_short_id' --name 'My super tool'
```

Complicated version:

```
$ 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'
```

planemo lint

Checks the syntax of a wrapper

```
$ planemo lint
```

```
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.
```

planemo serve

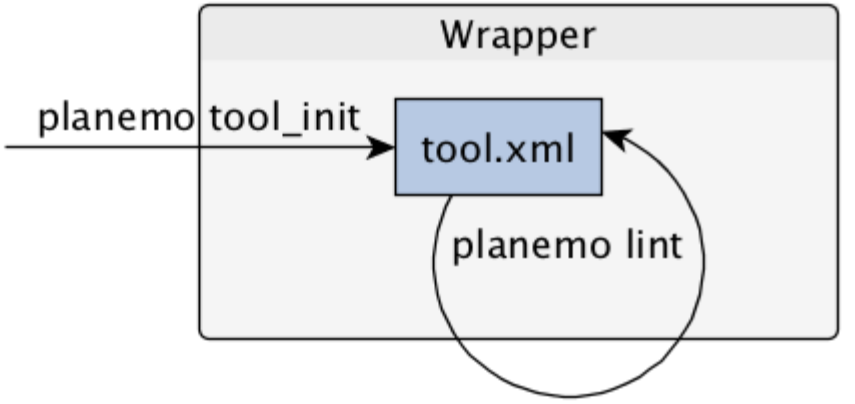
View your new tool in a local Galaxy instance

```
$ planemo serve
```

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

PLANEMO

Building Galaxy Tools



Hands-on

Hands-on

In a new directory:

Write a simple script (bash/python/perl) which takes 2 inputs:

- a number (x)
- a text

And write x times the text in a new output.

Write a wrapper for your tool:

- `planemo tool_init`
- you can also use the hello world wrapper as a template (available here).

Check it with planemo (`planemo lint`)

Test it with planemo (`planemo serve`*)

If you finish before others, write another tool which take a text file and extract a given line.

*`planemo serve` will create a local minimal galaxy instance. This is quite resource demanding and I advice you to save everything before in case your computer crashes.

Galaxy Dependencies

These slides mirror the section on "Dependencies and Conda" in the Planemo Documentation.

Example Tool (1 / 2)

From Planemo docs - the following example builds a tool for the `seqtk seq` 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.fast
                    --example_input 2.fastq \
                    --example_output 2.fasta \
                    --test_case \
                    --cite_url 'https://github.com/lh3/seqtk' \
                    --help_from_command 'seqtk seq'
```

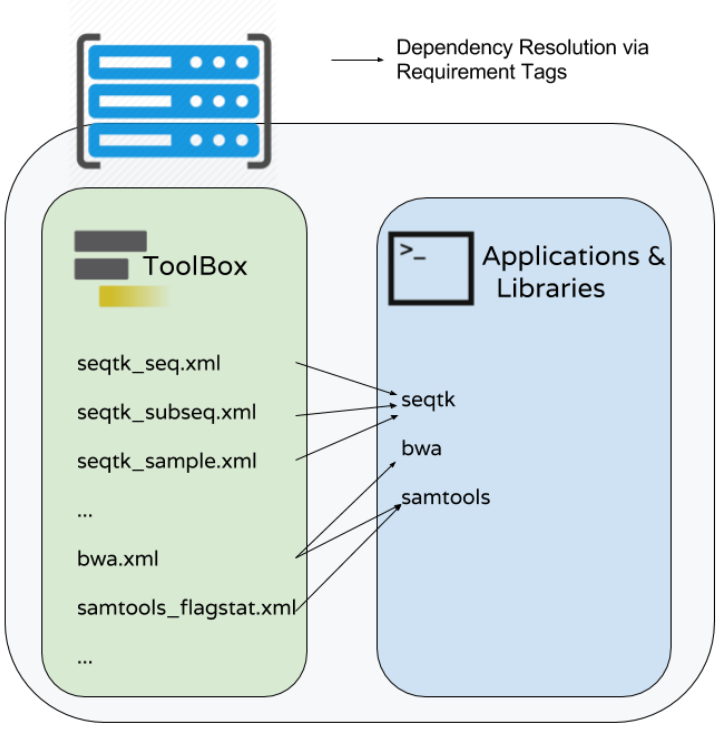
Notice the `--requirement seqtk@1.2`.

Example Tool (2 / 2)

The `--requirement seqtk@1.2` gets translated into the following Galaxy tool XML:

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

Dependency Resolution



Conda and Galaxy

Galaxy now automatically installs Conda when first launched and will use Bioconda and other channels for package resolution.

Linting Conda Dependencies

The next few slides will use the seqtk example from Planemo's documentation - this can be downloaded to follow along using the following command:

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

Planemo can check if the requirements of a tool are available in best practice Conda channels using the `--conda_requirements` flag of `planemo lint`.

```
$ planemo lint --conda_requirements
Linting tool /home/ldelisle/Documents/mygit/test/seqtk_example/seqtk_seq.xml
  ...
Applying linter requirements_in_conda... CHECK
.. INFO: Requirement [seqtk@1.2] matches target in best practice Conda channel [ht
```

Notice Planemo indicates this tool is available and shows the channel it is available in.

The Planemo `conda_install` command

```
$ planemo conda_install seqtk_seq.xml
Install conda target CondaTarget[seqtk,version=1.2]
/home/ldelisle/miniconda3/bin/conda create -y --quiet --override-channels --channel iuc --channel conda-forge --channel bioc
Collecting package metadata: ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: /home/ldelisle/miniconda3/envs/__seqtk@1.2

  added / updated specs:
    - seqtk=1.2


The following NEW packages will be INSTALLED:

  _libgcc_mutex      pkgs/main/linux-64::_libgcc_mutex-0.1-main
  libgcc-ng          pkgs/main/linux-64::libgcc-ng-9.1.0-hdf63c60_0
  seqtk              bioconda/linux-64::seqtk-1.2-1
  zlib               conda-forge/linux-64::zlib-1.2.11-h516909a_1006


Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
$ which seqtk
seqtk not found
```

Notice seqtk hasn't been placed on the PATH, an environment has been setup that Galaxy (when used through Planemo) can leverage.

Using the Tool Environment

Now that we have verified the Conda environment setup with `conda_install` works properly on the command-line, we can use our tool!

`planemo test` and `planemo serve` will use this environment by default now for this tool.

Planemo `test`

```
$ planemo test seqtk_seq.xml
...
2019-10-14 12:24:41,414 INFO  [galaxy.jobs.handler] (2) Job dispatched
2019-10-14 12:24:41,524 DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda
2019-10-14 12:24:41,525 DEBUG [galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda
2019-10-14 12:24:41,538 INFO  [galaxy.jobs.command_factory] Built script [/tmp/tmpe8nrdkxv/job_working_directory/000/2/tool_
MAX_TRIES=3
COUNT=0
while [ $COUNT -lt $MAX_TRIES ]; do
...
done ; seqtk seq -a '/tmp/tmpe8nrdkxv/files/7/e/9/dataset_7e9b6bad-75e9-4039-aa6e-127698cc6426.dat' > '/tmp/tmpe8nrdkxv/file
2019-10-14 12:24:41,572 DEBUG [galaxy.tools.deps] Using dependency bcftools version 1.5 of type conda
2019-10-14 12:24:41,572 DEBUG [galaxy.tools.deps] Using dependency bcftools version 1.5 of type conda
ok

----------------------------------------------------------------------
XML: /tmp/tmpe8nrdkxv/xunit.xml
----------------------------------------------------------------------
Ran 1 test in 14.254s

OK
...
Testing complete. HTML report is in "/home/ldelisle/Documents/mygit/test/seqtk_example/tool_test_output.html".
All 1 test(s) executed passed.
seqtk_seq[0]: passed
```

The following line indicates the seqtk package was found:

`[galaxy.tools.deps] Using dependency seqtk version 1.2 of type conda`

Hands-on

Hands-on

The Goal

- Use the Planemo commands `conda_install`, `conda_env`, and `test` to practice the Galaxy tool dependency development lifecycle.

Hands-on

Steps

Run the following commands to practice working with Galaxy tools, Planemo, and Conda.

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

```
$ cd seqtk_example
```

```
$ planemo conda_install seqtk_seq.xml
```

```
$ planemo test seqtk_seq.xml
```

Finding the Correct Requirements & Packages

The previous example worked because a published Bioconda recipe named `seqtk` at version `1.2` was previously published, but how can these be found?

Two easy approaches are

1. using `planemo conda_search`
2. using the Anaconda web search.

Using the Planemo `conda_search` Command

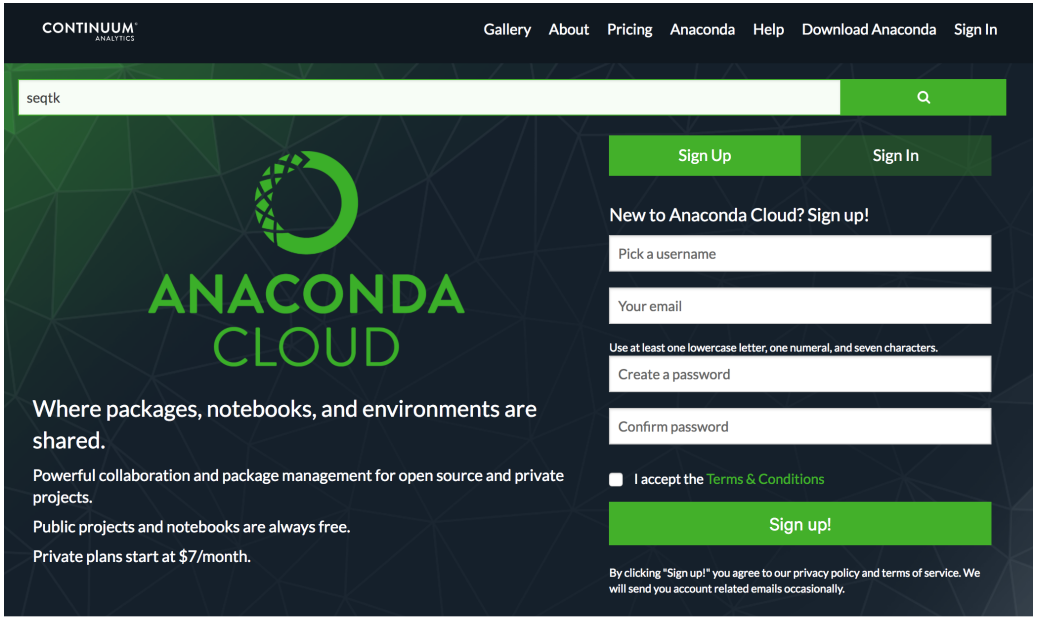
The Planemo `conda_search` command is a shortcut around `conda search` that searches best practice channels that Galaxy is configured to work with:

```
$ planemo conda_search seqtk
/home/ldelisle/miniconda3/bin/conda search --override-channels --channel iuc --cha
Loading channels: done
# Name                       Version           Build  Channel
fusioncatcher-seqtk              1.2      h84994c4_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
seqtk                            1.3      h84994c4_1  bioconda
seqtk                            1.3      ha92aebf_0  bioconda
seqtk                            1.3      hed695b0_2  bioconda    planemo conda_se
```

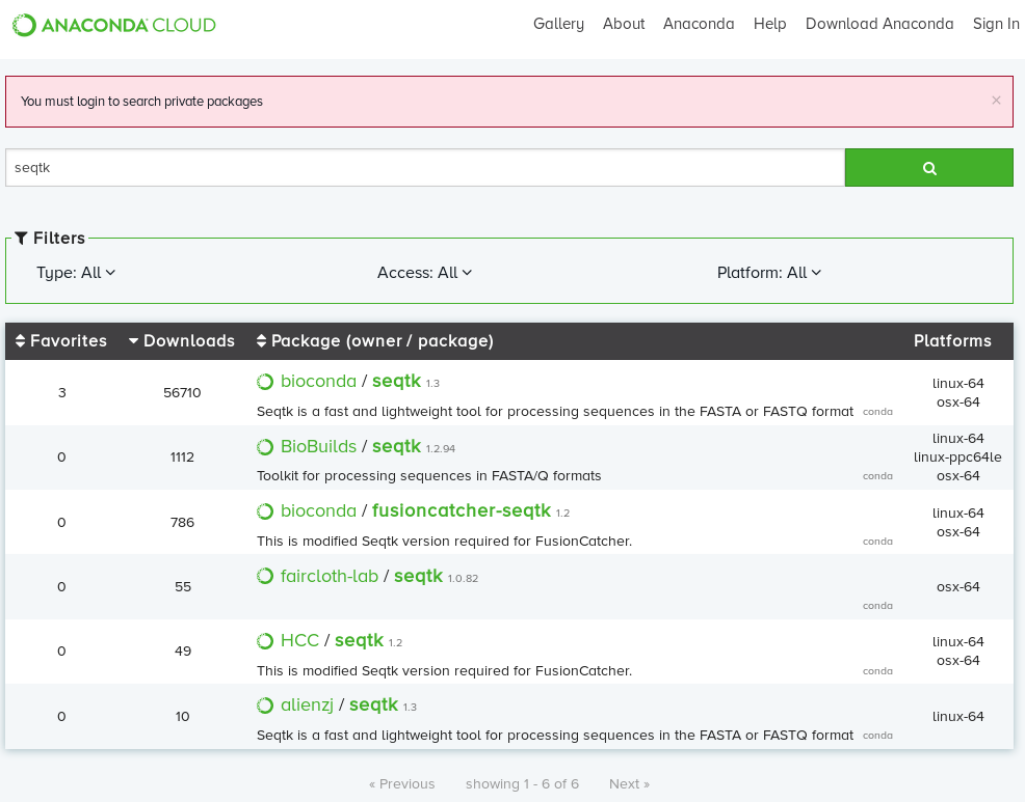
Alternatively, `conda` can be used directly:

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

Using Anaconda Search - https://anaconda.org



Using Anaconda Search - https://anaconda.org



Hands-on

Hands-on

The Goal

- Find the correct package and version for a tool in a best practice channel.
- Add a requirement to a tool to allow Galaxy to find, install, and use a Conda package.

Hands-on

Steps

1. Run the following commands to download an example tool to modify.

```
$ planemo project_init --template conda_exercises conda_exercises
$ cd conda_exercises/exercise1
$ ls
pear.xml              test-data
```

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

Writing a Conda Recipe

If searching best practice channels fails, you may need to build a Conda recipe.

There are skeleton for packages available in PyPI, cpan, CRAN, Bioconductor.

Documentation about how to write conda recipes from scratch
Bioconda guidelines

Options for using Galaxy

Options for using Galaxy

Galaxy is available in *many* different ways and some of those ways are better suited for some tasks than others. Here are some guidelines to help you decide how to use Galaxy.

Which Galaxy instance to use?

	UseGalaxy Servers	Public Servers	TIaaS	Academic Cloud Services	Commercial Cloud Services	Containers	VMs	Local
Free to use	Yes	Yes	Yes	Yes[1]	No	Yes	Yes	Yes
Uses your local compute infrastructure	No	No	No	No	No	Yes[2]	Yes[2]	Yes
Your have large datasets with many intermediate datasets (> 250GB)	No	?	Yes	Yes	Yes	?[3]	?[3]	Yes
Your computational requirements are similarly large	No	?	Yes	Yes	Yes	?[3]	?[3]	Yes
You want to share your Galaxy objects with others outside your organization	Yes	Yes	Yes	Yes	Yes	Yes[4]	Yes[4]	?[5]
You have control over the tools and reference genomes that are installed	No	No	No	Yes[5]	Yes	Yes	Yes	Yes
You have absolute data security requirements	No	No	No	?	?	?	?	Yes

[1] If you qualify for the service.
[2] Although these technologies can also be deployed on clouds.
[3] Depends on the size of the system you are running it on.
[4] With these technologies you can save the server and share the entire platform with them.
[5] Depends on configuration.

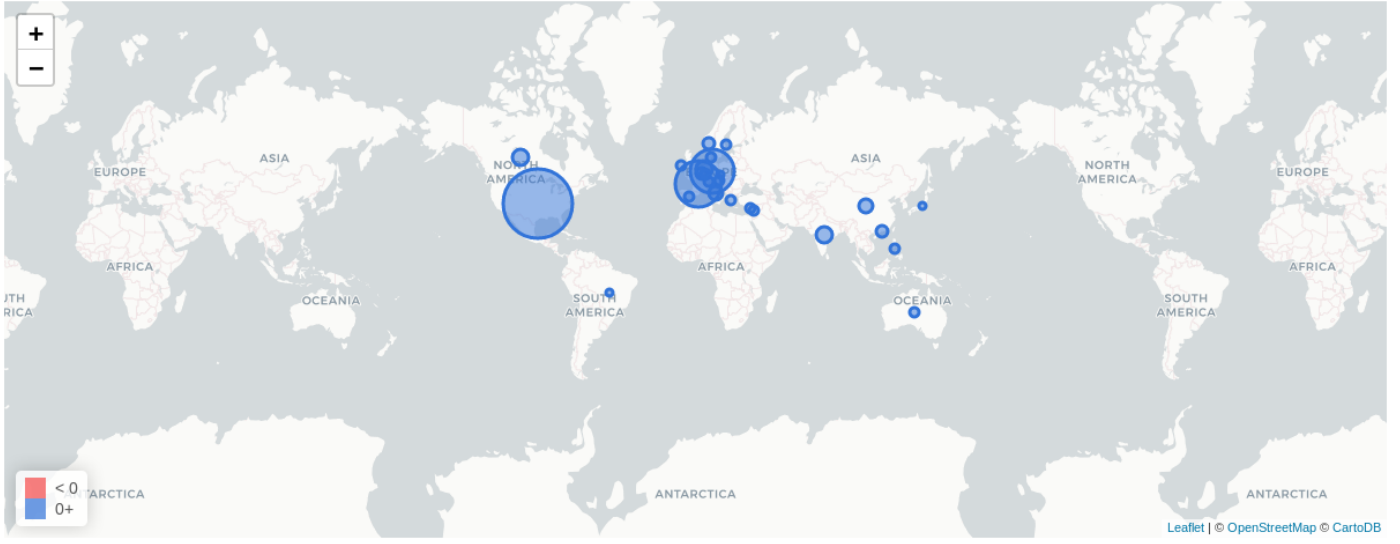
UseGalaxy Servers

UseGalaxy servers implement a common core set of tools and reference genomes, and are open to anyone to use. They also contain tools and genomes that are local to each server. Each is backed by significant computational resources and they are excellent places to get started with Galaxy, and to share and publish your results.

- UseGalaxy.org
- UseGalaxy.eu
- UseGalaxy.org.au
- UseGalaxy.be

Public Galaxy Servers

There are 126 Public Galaxy Servers



They are acccessible to (at least) any academic researcher in the world. Some require you to create/request an account, and they may have restrictive quotas, but often, to use these, just go the website and start running analyses. These are free to use.

Complete list and stats on status and number of tools:
https://stats.galaxyproject.eu/d/000000020/public-galaxy-servers?orgId=1

Academic Cloud Services

These are academic cloud providers where ready to run Galaxy instances are available. Many of these are restricted to users within a particular geographic domain, for example, a country, province, or consortium of countries. These are often free to eligible researchers.

Commercial Cloud Services

These commercial cloud providers have ready to run Galaxy instances are available. You need to pay for these, but you only need to pay for as long as you need the instance to be up.

Containers

These Galaxy instances are prepackaged using container technology, usually Docker. You run these locally by first installing the container technology (Docker is easy), and then launching the containerized Galaxy within that technology. These use your local resources.

Virtual Machines (VMs)

VMs are similar to containers but use a different technology. Containers take significant advantage of your computer's underlying operating system. Virtual Machines include an entire supporting operating system, and are significantly larger than containers. You run these locally by first installing a VM player like VirtualBox, and then downloading and launching the VM within that player. These use local resources.

Run your own Galaxy locally

- Galaxy is open source software and can be installed on local compute infrastructure, from lab servers to institutional compute clusters
- Installing Galaxy locally is relatively easy, but
  - the initial install does not include reference genomes and only has a few tools
  - installing tools and genomes, setting up authentication, and connecting to institutional compute resources all takes work
- Main advantages personnaly as an admin:
  - you install the tools you want (including your own).
  - the storage can be greatly increased compared to public ones.
  - goes faster (?)
  - have access to your biologist histories
  - also to the logs to understand when something goes wrong.

Useful links

- All galaxy training material (for biologist and developers)
- online documentation for tools wrapper
- List of possible formats
- Documentation about how to write conda recipes from scratch
    - Hands-on to build a conda recipe

Key points

- Galaxy gives access to command line tools to biologists.
- Wrapping a tool makes it available to all local galaxy platform and potentially to public ones.
- Conda and Bioconda are Galaxy best practices for connecting Galaxy tools to underlying applications and libraries.
- Planemo is your friend if you want to add a new tool in galaxy

Thank you!

This material is the result of a collaborative work. Thanks to the Galaxy Training Network and all the contributors!

Anthony Bretaudeau, Andrea Bagnacani, Bérénice Batut, Björn Grüning, Helena Rasche, Hervé Ménager, John Chilton, Gildas Le Corguillé, Lucille Delisle, Nicola Soranzo, Anne Pajon, Saskia Hiltemann, Dave Clements