



# 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

## 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

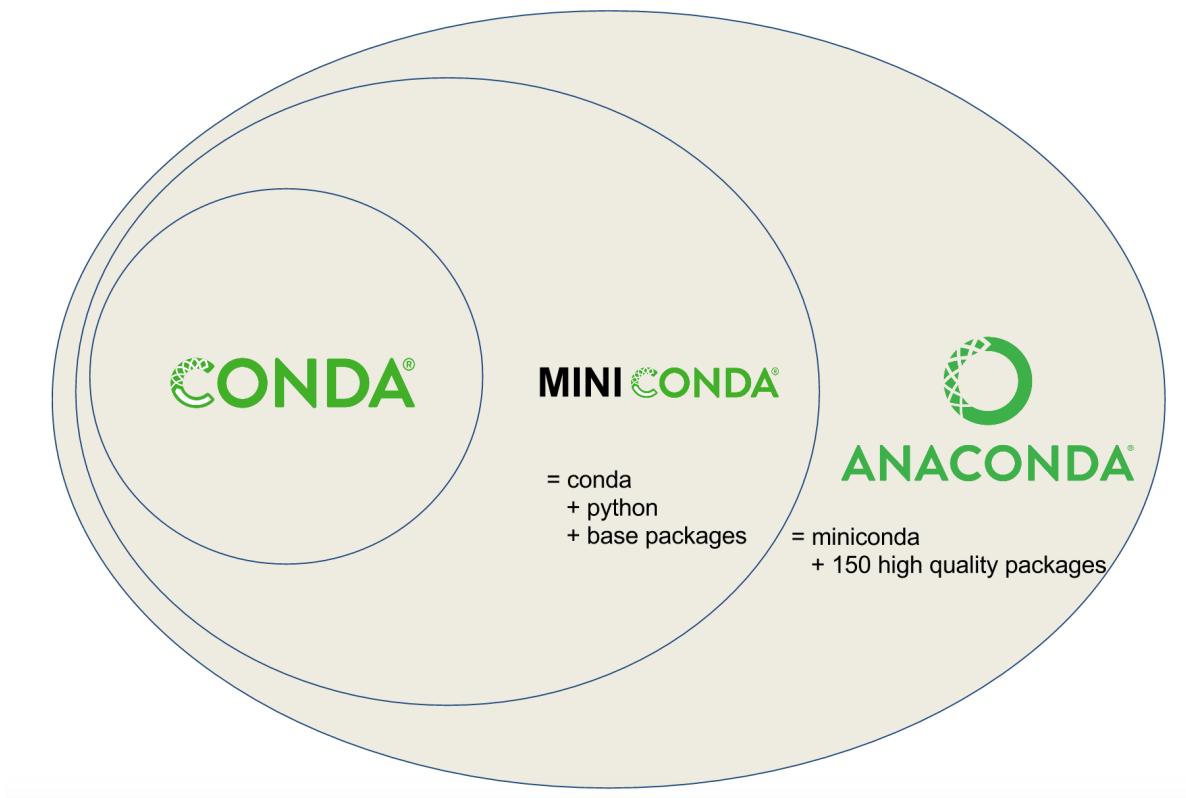
## Conda Key Features

- 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





# CONDA

## Channels

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

To add them:

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





# CONDA

## Quickstart

### Using Conda

- Install some packages within an isolated environment

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





# 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

The screenshot shows the Galaxy web-based platform interface. The left panel, titled "Tools", contains a sidebar with a search bar and a list of tool categories: Get Data, Send Data, Lift-Over, Collection Operations, Text Manipulation, Datamash, Convert Formats, Filter and Sort, Join, Subtract and Group, Fetch Alignments/Sequences, NGS: QC and manipulation, NGS: DeepTools, NGS: Mapping, NGS: RNA Analysis, NGS: SAMtools, NGS: BamTools, NGS: Picard, NGS: VCF Utilities, NGS: Peak Calling, NGS: Variant Analysis, NGS: RNA Structure, NGS: Du Novo, NGS: Gemini, NGS: Assembly, NGS: Chromosome Conformation, NGS: Mothur, Operate on Genomic Intervals, Statistics, Graph/Display Data, Phenotype Association, BEDTools, Genome Diversity, EMBOSS, Regional Variation, FASTA manipulation, Multiple Alignments, Metagenomic Analysis. The middle panel, titled "Main", features a red header with the Galaxy introduction text: "Galaxy is an open source, web-based platform for data intensive biomedical research. If you are new to Galaxy [start here](#) or consult our [help resources](#). You can install your own Galaxy by following the [tutorial](#) and choose from thousands of tools from the [Tool Shed](#)". Below this is a section titled "Running Your Own Understanding how Galaxy works" with the subtitle "An in-depth tutorial". It includes a "Tweets" feed from the @galaxyproject account, logos for Penn State, Johns Hopkins University, and the National Institute of Health, and logos for TACC and CyVerse. A note at the bottom states: "The Galaxy Team is a part of the Center for Comparative Genomics and Bioinformatics at Penn State, the Department of Biology and at Johns Hopkins University and the Computational Biology Program at Oregon Health & Science University." Another note indicates: "This instance of Galaxy is utilizing infrastructure generously provided by the CyVerse at the Texas Advanced Computing Center, with support from the National Science Foundation." The right panel, titled "History", shows a history dataset named "Galaxy introduction" which is currently empty. It includes a message: "This history is empty. You can load your own data or get data from an external source".

Home page divided into 3 panels



# Tools

The screenshot shows the Galaxy web interface. On the left, a sidebar lists various tools under categories like NGS: Peak Calling, NGS: Variant Analysis, NGS: Du Novo, NGS: Mothur, and Operate on Genomic Intervals. The 'Join' tool from the 'Operate on Genomic Intervals' section is highlighted with a red box. The main workspace shows the 'Join the intervals of two datasets side-by-side (Galaxy Version 1.0.0)' tool configuration. It has two dropdown menus: 'First dataset' set to '1: Exons' and 'Second dataset' set to '2: SNPs'. Below these is a text input for 'with min overlap' containing the value '1'. Under 'Return', it says 'Only records that are joined (INNER JOIN)'. A 'Execute' button is present. To the right, the 'History' panel shows two datasets: 'Galaxy 101' (9.06 MB) and '2: SNPs' (highlighted in green). Below the history is a 'TIP' section about interval format and a 'Syntax' section with detailed bullet points.

**Join the intervals of two datasets side-by-side (Galaxy Version 1.0.0)**

**Join**

First dataset: 1: Exons

Second dataset: 2: SNPs

with min overlap: 1

Return: Only records that are joined (INNER JOIN)

**TIP:** If your dataset does not appear in the pulldown menu, it means that it is not in interval format. Use "edit attributes" to set chromosome, start, end, and strand columns.

**Screencasts!**  
See Galaxy Interval Operation [Screencasts](#) (right click to open this link in another window).

**Syntax**

- Where overlap specifies the minimum overlap between intervals that allows them to be joined.
- Return only records that are joined returns only the records of the first dataset that join to a record in the second dataset. This is analogous to an INNER JOIN.
- Return all records of first dataset (fill null with ".") returns all intervals of the first dataset, and any intervals that do not join an interval from the second dataset are filled in with a period(.). This is analogous to a LEFT JOIN.
- Return all records of second dataset (fill null with ".") returns all intervals of the second dataset, and any intervals that do not join an interval from the first dataset are filled in with a period(.). Note that this may produce an invalid interval file, since a period(.) is not a valid chrom, start, end or strand.
- Return all records of both datasets (fill nulls with ".") returns all records from both datasets, and fills on either the right or left with periods. Note that this may produce an invalid interval file, since a period(.) is not a valid chrom, start, end or strand.

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



# Tool interface

NCBI BLAST+ blastp Search protein database with protein query sequence(s) (Galaxy Version 0.3.1)

Protein query sequence(s)  
No fasta or fasta.gz dataset available.  
(-query)

Subject database/sequences  
Locally installed BLAST database

Protein BLAST database  
Select/Unselect all

Type of BLAST  
blastp - Traditional BLASTP to compare a protein query to a protein database  
blastp-short - BLASTP optimized for queries shorter than 30 residues  
blastp-fast - Use longer words for seeding, faster but less accurate

See help text for default parameter values for each BLAST type. (-task)

Set expectation value cutoff  
0.001  
(-evalue)

Output format  
Tabular (extended 25 columns)  
(-outfmt)

Advanced Options  
Hide Advanced Options

Execute

- 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

The screenshot shows the Galaxy Tool Shed homepage. The top navigation bar includes links for 'Repositories', 'Groups', 'Help', and 'User'. The main content area is titled 'Repositories by Category' and features a search bar. A table lists various tool categories with their descriptions and counts:

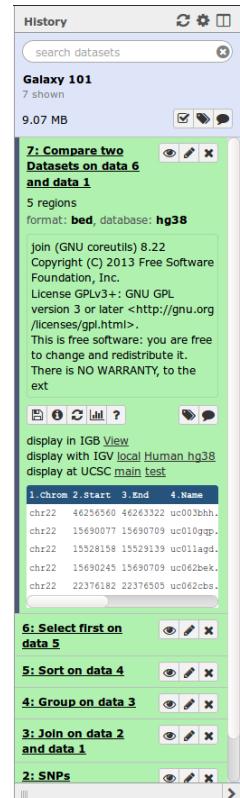
Name	Description	Repositories
Assembly	Tools for working with assemblies	128
ChIP-seq	Tools for analyzing and manipulating ChIP-seq data.	65
Combinatorial Selections	Tools for combinatorial selection	10
Computational chemistry	Tools for use in computational chemistry	76
Constructive Solid Geometry	Tools for constructing and analyzing 3-dimensional shapes and their properties	12
Convert Formats	Tools for converting data formats	114
	Tools for exporting data to various	~

- 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**

The screenshot shows the Galaxy web interface with four distinct histories displayed as tabs:

- Workflow extract error**: Contains datasets from a workflow step that failed.
- Unnamed history**: Contains a dataset named "127: Heatmap\_sim on collection 86: heatmap\_sim.svg".
- Training: 16S rRNA sequencing with mother**: Contains a dataset named "236: Krona pie chart on data 235: HTML".
- Unnamed history**: Contains a dataset named "41: samples".

Each history tab has a "Switch to" button above it. The interface also includes a search bar at the top and a sidebar on the left.



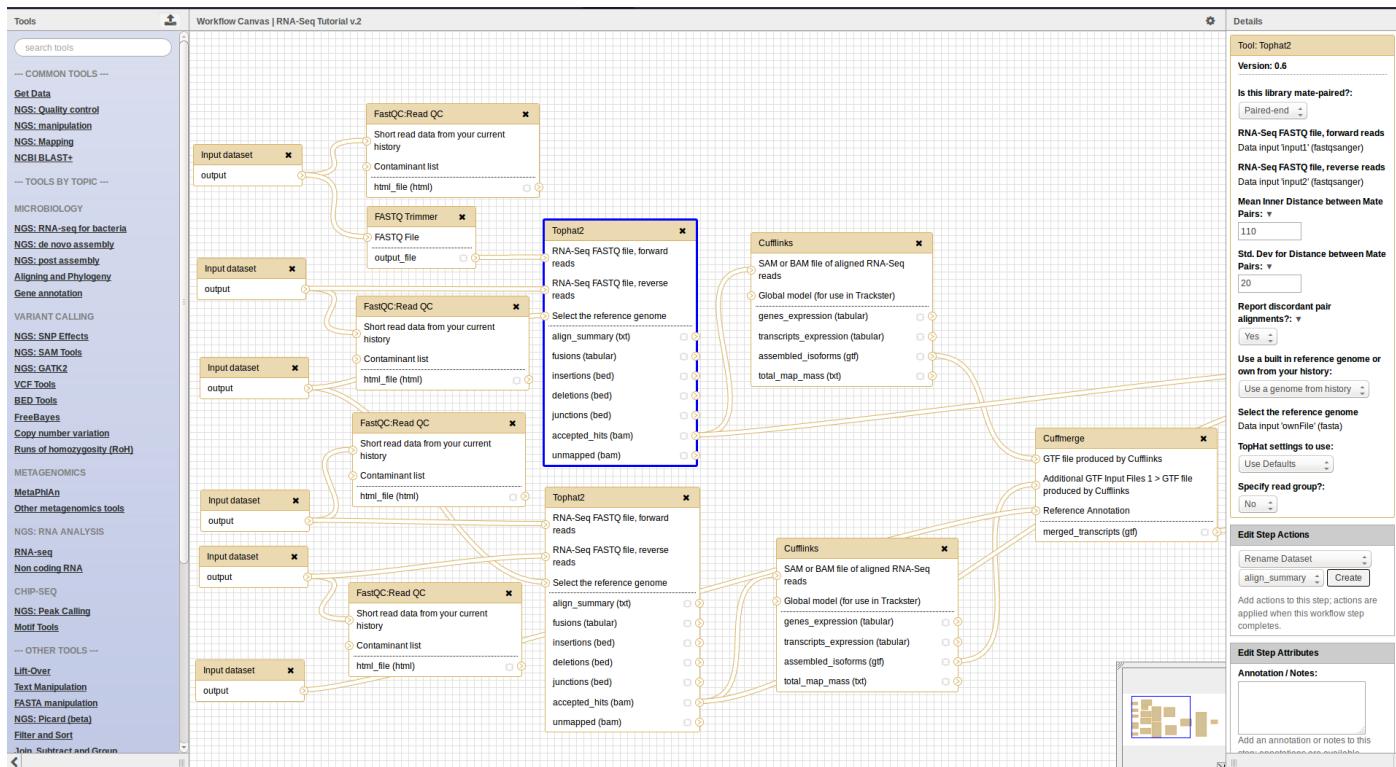


# Workflows





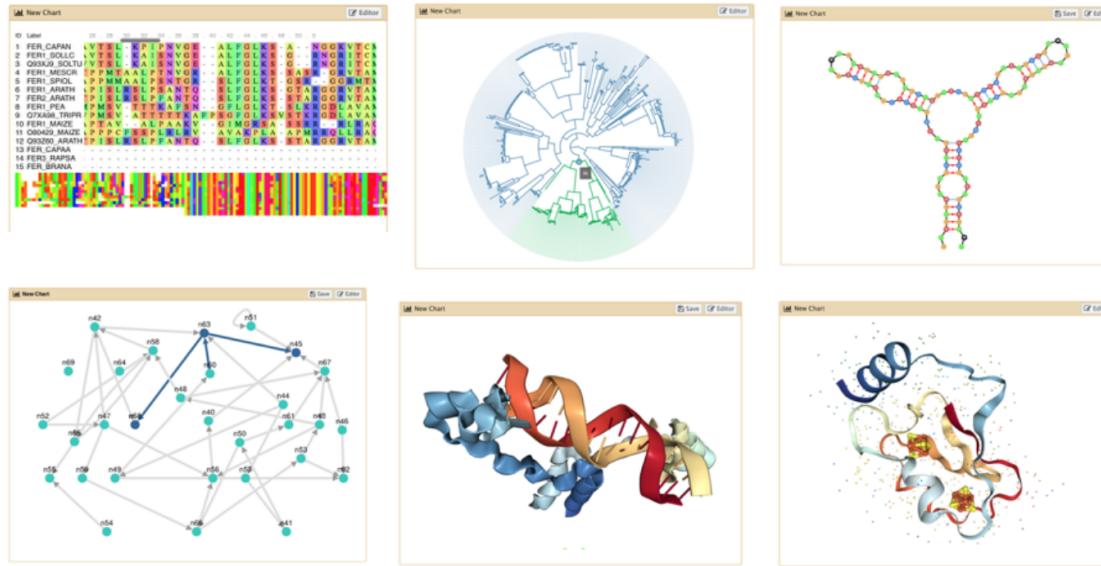
# Workflow Editor



- **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](#)

The screenshot shows the Galaxy Help forum homepage. At the top, there is a navigation bar with links for "Sign Up", "Log In", and a menu icon. Below the navigation is a search bar with placeholder text "search topics, posts, users, or categories". Underneath the search bar are four buttons: "all categories", "all tags", "Latest" (which is highlighted in red), "Top", and "Categories". The main content area displays two forum posts:

Topic	Category	Users	Replies	Views	Activity
<a href="#">Troubleshooting resources for errors or unexpected results</a> Start by reviewing the troubleshooting FAQ. Common reasons and solutions for tool errors are explained. Most job errors can be resolved by correcting your input data's format/content. Others indicate a tool setting/param... <a href="#">read more</a>	usegalaxy.org support		1	85	7d
<a href="#">Welcome to Galaxy Community Help</a> For assistance with a specific Galaxy server please post into appropriate category.			1	75	15d

- 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

The screenshot shows the Galaxy web interface with the following components:

- Left Sidebar:** Lists available tools under categories like COMMON TOOLS, SEQUENCE PREPARATION TOOLS, and STATISTICS AND VISUALIZATION TOOLS.
- Middle Panel (Tool Configuration):** For the "GraPhlAn to produce graphical output of an input tree (Galaxy Version 0.9.7)" tool.
  - Input tree:** Set to "107: Generation, personalization and annotation of tree on data 106 and data 105: Tree in PhyloXML".
  - Output format:** Set to "PNG".
  - Dpi of the output image (Optional):** Empty input field.
  - Size of the output image (in inches):** Set to "7".
  - Distance between the most external graphical elements (Optional):** Empty input field.
  - Execute:** A blue button with a checkmark icon.
- Center Panel (Tool Help):** A black box containing the word "Wrapper".
- Right Panel (History):** Shows a list of datasets with green headers:
  - 108: GraPhlAn on data 1 (07: Image)
  - 107: Generation, personalization and annotation of tree on data 106 and data 105: Tree in PhyloXML
  - 96: graphlan\_formatted\_mapping\_results.txt
  - 95: Plot grouped barplot on data 94: PDF barplot
  - 94: Join two Datasets on data 81 and data 93
  - 93: Normalize a dataset

History  
with  
results as  
datasets





# Galaxy tool / wrapper

GraPhlAn to produce graphical output of an input tree (Galaxy Version 0.9.7) ▾ Options

**Input tree**  
   107: Generation, personalization and annotation of tree on data 106 and data 105: Tree in PhyloXML ▾  
The tree must be in PhyloXML, Newick or text format.

**Output format**  
 PNG ▾  
(--format)

**Dpi of the output image (Optional)**

For non vectorial formats (--dpi)

**Size of the output image (in inches)**  
 7  
(--size)

**Distance between the most external graphical element and the border of the image (Optional)**  
  
(--pad)

Execute

## What it does

GraPhlAn is a software tool for producing high-quality circular representations of taxonomic and phylogenetic trees. GraPhlAn focuses on concise, integrative, informative, and publication-ready representations of phylogenetically- and taxonomically-driven investigation.

For more information, check the [user manual](#).

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



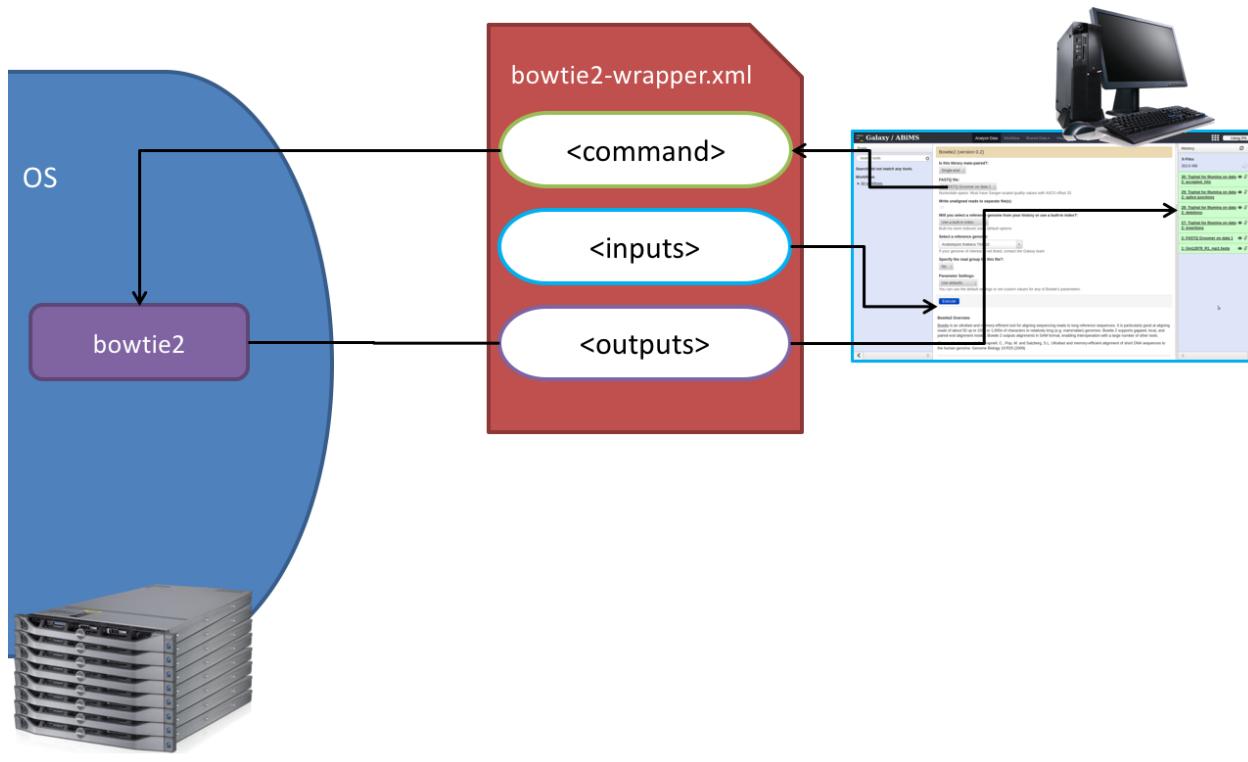
# Wrapper

- 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





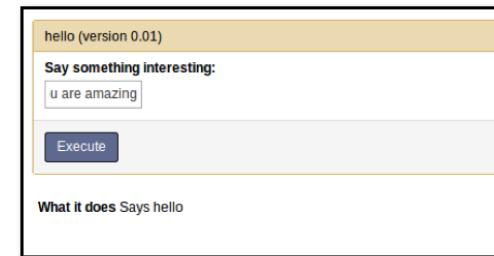
## Wrapper





# 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**
    ]]></help>
    <Job Command-Line:>
        /bin/echo 'Hello World You are amazing' > '/usr/local/galaxy/database/files/000/dataset_9.dat'
    </Job Command-Line:>
</tool>
```

The diagram illustrates the data flow within the tool configuration. A blue-bordered box labeled 'param name="mystring"' points to a green-bordered box containing the command substitution '\$mystring'. Another blue-bordered box labeled 'data format="tabular" name="output1"' points to a green-bordered box containing the output path '\$output1'.





# 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>
```

The diagram illustrates the execution flow of the tool. It starts with a blue-bordered box containing the XML code. A black-bordered box labeled 'mystring' points to the '\$mystring' placeholder in the command line. Another black-bordered box labeled 'output1' points to the '\$output1' placeholder. A third black-bordered box labeled 'Hello World! You are amazing' points to the final output data. Arrows indicate the flow from the input parameter to the command substitution and finally to the output data.





# 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?

GraPhlAn to produce graphical output of an input tree (Galaxy Version 0.9.7) ▼ Options

**Input tree**  
   107: Generation, personalization and annotation of tree on data 106 and data 105: Tree in PhyloXML ▼  
The tree must be in PhyloXML, Newick or text format.

**Output format**  
 ▼  
(--format)

**Dpi of the output image (Optional)**  
  
For non vectorial formats (--dpi)

**Size of the output image (in inches)**  
  
(--size)

**Distance between the most external graphical element and the border of the image (Optional)**  
  
(--pad)

**Execute**

### What it does

GraPhlAn is a software tool for producing high-quality circular representations of taxonomic and phylogenetic trees. GraPhlAn focuses on concise, integrative, informative, and publication-ready representations of phylogenetically- and taxonomically-driven investigation.

For more information, check the [user manual](#).





# 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

## Input tree

107: Generation, personalization and annotation of tree on data 106 and data 105: Tree in PhyloXML ▾

The tree must be in PhyloXML, Newick or text format.

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

[List of possible formats](#)





# inputs > param > integer | float | text | boolean

## E-value threshold

(-e)

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

## Label for x axis

(--xlab)

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

## Generate statistics file

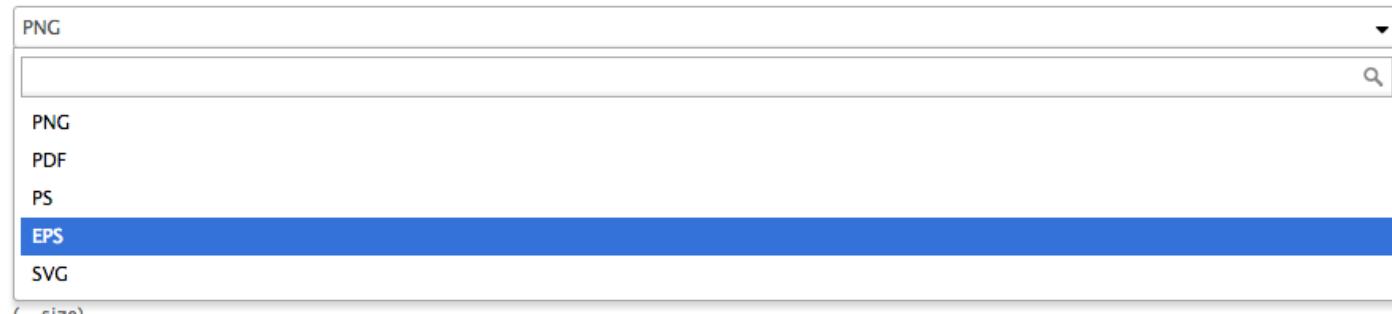
Generates statistics for the rRNA content of reads, as well as rRNA subunit distribution. (--log)

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



# inputs > param > select

## Output format



A screenshot of a dropdown menu titled "Output format". The menu contains five items: "PNG", "PDF", "PS", "EPS", and "SVG". The "EPS" option is highlighted with a blue background, indicating it is selected. A search icon is visible in the top right corner of the dropdown.

```
<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?

The screenshot shows a software interface with a 'History' tab at the top. Below it is a search bar labeled 'search datasets'. A section titled 'Unnamed history' shows three entries:

- 3: Export to GraPhlAn on data 1: Annotation**  
112 lines  
format: txt, database: ?  
Content:

```
clade_separation      0.5
branch_bracket_depth  0.8
branch_bracket_width  0.2
annotation_legend_font_size 10
class_legend_font_size 10
class_legend_marker_size 1.5
```
- 2: Export to GraPhlAn on data 1: Tree**  
94 lines  
format: txt, database: ?  
Content:

```
k_Archaea
k_Archaea.p_Euryarchaeota
k_Archaea.p_Euryarchaeota.c_Methano
k_Archaea.p_Euryarchaeota.c_Methano
k_Archaea.p_Euryarchaeota.c_Methano
k_Archaea.p_Euryarchaeota.c_Methano
```
- 1: 1 taxonomy.tabular**  
Content:





# 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

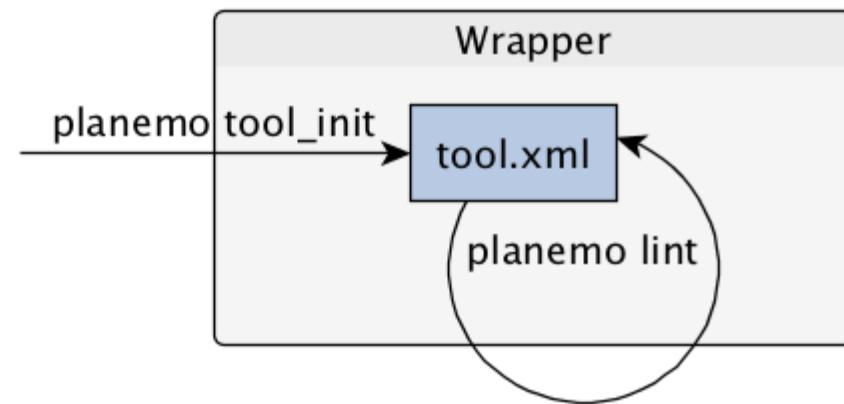
```
$ planemo serve
```

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.fasta' \
    --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.





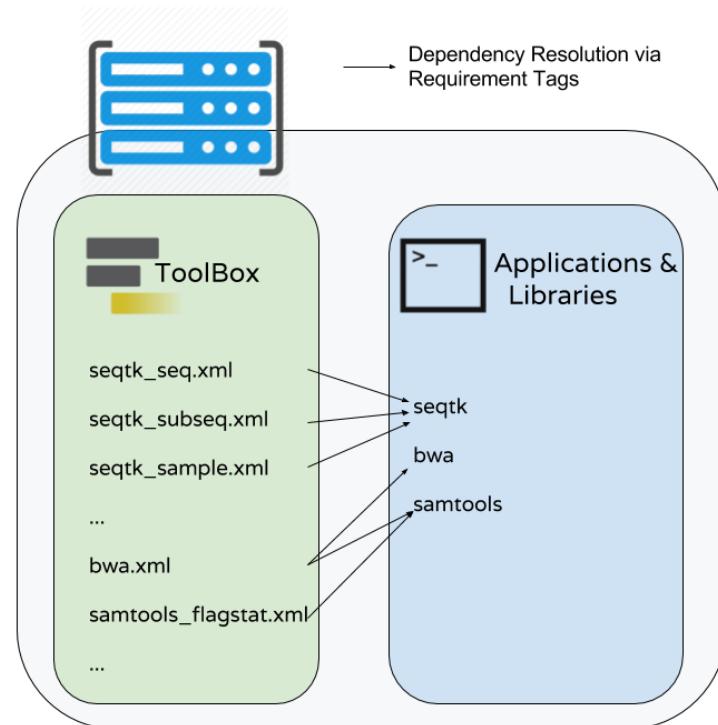
## Example Tool (2 / 2)

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

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/ldelelis/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/ldelelis/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/ldelelis/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.toolsdeps] Using dependency seqtk version 1.2 of type conda
2019-10-14 12:24:41,525 DEBUG [galaxy.toolsdeps] 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.toolsdeps] Using dependency bcftools version 1.5 of type conda
2019-10-14 12:24:41,572 DEBUG [galaxy.toolsdeps] 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/ldelelie/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.toolsdeps] 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/ldeleine/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>

The screenshot shows the Anaconda Cloud sign-up page. At the top, there's a navigation bar with links for Gallery, About, Pricing, Anaconda, Help, Download Anaconda, and Sign In. A search bar contains the text "seqtk". On the left, the Anaconda Cloud logo is displayed with the text "Where packages, notebooks, and environments are shared." Below this, there are three bullet points: "Powerful collaboration and package management for open source and private projects.", "Public projects and notebooks are always free.", and "Private plans start at \$7/month.". On the right, there are "Sign Up" and "Sign In" buttons. A registration form follows, with fields for "Pick a username", "Your email", and "Create a password" (with a note: "Use at least one lowercase letter, one numeral, and seven characters."), and a "Confirm password" field. A checkbox for accepting the Terms & Conditions is present, followed by a large green "Sign up!" button. A small note at the bottom states: "By clicking "Sign up!" you agree to our privacy policy and terms of service. We will send you account related emails occasionally."

## Why you'll love Anaconda Cloud

Making it easy to share packages, notebooks, and environments to be more collaborative.





# Using Anaconda Search - <https://anaconda.org>

ANACONDA CLOUD

Gallery About Anaconda Help Download Anaconda Sign In

You must login to search private packages X

seqtk Search

**Filters**

Type: All ▾ Access: All ▾ Platform: All ▾

▼ Favorites	▼ Downloads	▼ Package (owner / package)	Platforms
3	56710	<span style="color: green;">●</span> bioconda / seqtk 1.3 Seqtk is a fast and lightweight tool for processing sequences in the FASTA or FASTQ format <small>conda</small>	linux-64 osx-64
0	1112	<span style="color: green;">●</span> BioBuilds / seqtk 1.2.94 Toolkit for processing sequences in FASTA/Q formats <small>conda</small>	linux-64 linux-ppc64le osx-64
0	786	<span style="color: green;">●</span> bioconda / fusioncatcher-seqtk 1.2 This is modified Seqtk version required for FusionCatcher. <small>conda</small>	linux-64 osx-64
0	55	<span style="color: green;">●</span> faircloth-lab / seqtk 1.0.82 <small>conda</small>	osx-64
0	49	<span style="color: green;">●</span> HCC / seqtk 1.2 This is modified Seqtk version required for FusionCatcher. <small>conda</small>	linux-64 osx-64
0	10	<span style="color: green;">●</span> alienzj / seqtk 1.3 Seqtk is a fast and lightweight tool for processing sequences in the FASTA or FASTQ format <small>conda</small>	linux-64

« Previous showing 1 - 6 of 6 Next »



 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.





# CONDA

## 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?

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

<sup>1</sup> If you qualify for the service.

<sup>2</sup> Although these technologies can also be deployed on clouds.

<sup>3</sup> Depends on the size of the system you are running it on.

<sup>4</sup> With these technologies you can save the server and share the entire platform with them.

<sup>5</sup> 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 accessible 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 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 personally 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

