

---

# **snakemake2019 Documentation**

***Release 0.0.1***

**Sateesh Peri**

**May 23, 2020**



---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Snakemake Intro Tutorial using Binder</b>	<b>5</b>
<b>3</b>	<b>Snakemake Tutorial on Cyverse Cloud</b>	<b>7</b>
3.1	Tutorial on Jetstream cloud . . . . .	7
<b>4</b>	<b>Working Locally</b>	<b>9</b>
<b>5</b>	<b>Awesome</b>	<b>11</b>
<b>6</b>	<b>Credits</b>	<b>13</b>
<b>7</b>	<b>Coming Soon</b>	<b>15</b>
<b>8</b>	<b>Introduction</b>	<b>17</b>
<b>9</b>	<b>Why Snakemake?</b>	<b>19</b>
9.1	Snakemake Advantages: . . . . .	20
<b>10</b>	<b>Why Containers?</b>	<b>21</b>
<b>11</b>	<b>Snakemake Basic Tutorial</b>	<b>23</b>
<b>12</b>	<b>Background</b>	<b>25</b>
<b>13</b>	<b>Getting started - your first Snakefile</b>	<b>27</b>
13.1	Updating the Snakefile to track inputs and outputs . . . . .	27
13.2	Forcibly re-running things . . . . .	28
<b>14</b>	<b>Multiple rules</b>	<b>31</b>
<b>15</b>	<b>A first refactoring: adding a better default rule</b>	<b>33</b>
<b>16</b>	<b>A second refactoring: doing a bit of templating</b>	<b>35</b>
<b>17</b>	<b>Refactoring 3: templating output files, too</b>	<b>37</b>
17.1	Adding some more files . . . . .	38
17.2	Rerunning snakemake . . . . .	38

<b>18 Building out the workflow</b>	<b>39</b>
18.1 Providing input files explicitly to the multiqc rule . . . . .	40
18.2 Refactoring this to make it slightly more concise – . . . . .	42
18.3 Digression: what files does snakemake check in order to decide about rerunning? . . . . .	42
18.4 Making a clean rule . . . . .	43
18.5 Digression: running things in parallel . . . . .	43
18.6 Doing more things in our workflow. . . . .	43
<b>19 More advanced snakemake</b>	<b>45</b>
19.1 Dry run . . . . .	45
19.2 Running things in parallel, revisited . . . . .	45
19.3 Specifying software required for a rule . . . . .	45
19.4 Outputting the entire workflow diagram . . . . .	46
19.5 Snakemake Report . . . . .	46
19.6 Adding in some Python... . . . .	47
<b>20 Final thoughts - writing your own snakefile</b>	<b>49</b>
20.1 Dealing with complexity . . . . .	49
<b>21 Thinking about workflows - a strong(er) argument</b>	<b>51</b>
<b>22 Go forth and Workflow!</b>	<b>53</b>
<b>23 Snakemake Tutorial on CyVerse cloud</b>	<b>55</b>
<b>24 Accessing The Atmosphere Cloud</b>	<b>57</b>
<b>25 Login &amp; Launch Instance</b>	<b>59</b>
<b>26 SSH Secure-Login</b>	<b>67</b>
<b>27 Instance Maintenance</b>	<b>69</b>
27.1 Atmosphere Dashboard . . . . .	69
27.2 Suspend Instance . . . . .	70
27.3 Delete Instance . . . . .	70
<b>28 Additional Features</b>	<b>73</b>
28.1 Did you know you can access a shell terminal and a web-desktop via your browser ??? . . . . .	73
<b>29 Advanced Topics</b>	<b>75</b>
29.1 Atmosphere Advanced Topics . . . . .	75
<b>30 Advanced Topics</b>	<b>77</b>
<b>31 Transferring Data to and from an Instance</b>	<b>79</b>
31.1 Transferring Data using iCommands . . . . .	79
<b>32 Creating custom Atmosphere Images</b>	<b>81</b>
<b>33 ssh-rsa-key for password-less login</b>	<b>83</b>
<b>34 Tmux &amp; Screen</b>	<b>87</b>
<b>35 SSH Remote host ID Changed Error</b>	<b>89</b>
<b>36 Workflow Management using Snakemake</b>	<b>91</b>
36.1 Learning Objectives . . . . .	91

36.2	Setup . . . . .	91
36.3	Introduction to Snakemake . . . . .	93
36.4	Starting with Snakemake . . . . .	94
36.5	Creating a pipeline with snakemake . . . . .	95
36.6	Using Snakemake to process multiple files . . . . .	97
36.7	Snakemake Additional Features . . . . .	98
36.8	Advanced Snakemake . . . . .	101
36.9	Resources . . . . .	101
<b>37</b>	<b>Singularity</b>	<b>103</b>
<b>38</b>	<b>Awesome List of Resources</b>	<b>105</b>
<b>39</b>	<b>Reproducibility Toolkit</b>	<b>107</b>
<b>40</b>	<b>Local Setup instructions</b>	<b>109</b>





The Snakemake workflow management system is a tool to create reproducible and scalable data analyses and can be seamlessly scaled to server, cluster, grid and cloud environments, without the need to modify the workflow definition.

Singularity is an open source-based container platform designed for scientific and high-performance computing (HPC) environments. Singularity supports Bring Your Own Environment (BYOE)—where entire Singularity environments can be transported between computational resources (e.g., users' PCs) with reproducibility.

- Basic **UNIX command-line** experience required. [Bootcamp](#)
- WINDOWS users can download UNIX-terminal interface **mobaxterm** [here](#)
- You don't need to be an expert at Python to use Snakemake, but it can sometimes be very useful.

No license; the below content is available under CC0 general license.

Author: [Sateesh Peri](#)





# CHAPTER 1

---

## Introduction

---

**What...Why???**



---

### Snakemake Intro Tutorial using Binder

---

#### Learning Objectives

- **Workflow Management** using **Snakemake**
- Move from separate scripts to connected analysis
- Understand snakemake syntax
- Understand the components of a Snakefile: rules, inputs, outputs, and actions
- Understand snakemake wildcards and pattern rules
- Understand how snakemake manages dependencies and outputs

#### Note:

- **No setup necessary**
- **Does not support ‘Singularity’ container execution**

Click on this **Binder** button to launch a Binder rendered via Jupyter Lab interface preinstalled with conda and Snakemake.

- **MyBinder** is a fantastic service that lets us run demonstrations and short workshops in the cloud!
- The binder is built using this ‘conda’ **environment.yml**.

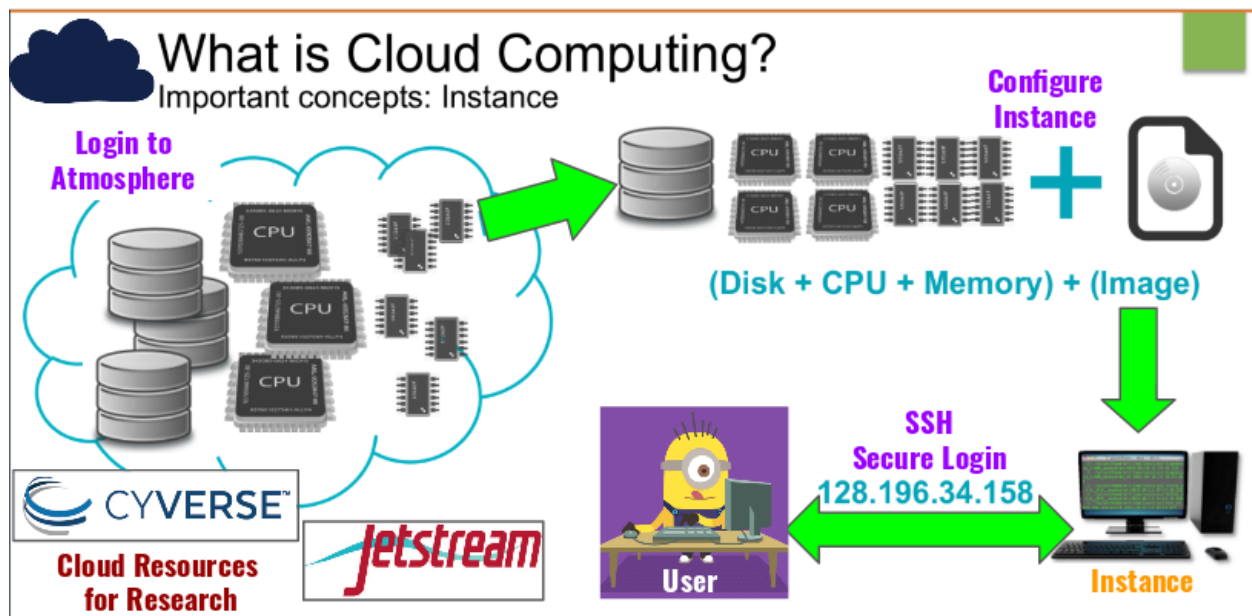
**[Click here for Introductory Snakemake Tutorial using Binder](#)**



## Snakemake Tutorial on Cyverse Cloud

Note:

- Requires access to [CyVerse Atmosphere](#)
- Supports 'Singularity' container execution



[Click here for Snakemake Tutorial using Singularity Containers in Atmosphere](#)

### 3.1 Tutorial on Jetstream cloud



## CHAPTER 4

---

### Working Locally

---

Click [here](#) for local setup instructions.

Requires *Administrator* privileges for Singularity installation.





## CHAPTER 5

---

### Awesome

---

- [Awesome list of Snakemake & Singularity resources](#)



## CHAPTER 6

---

### Credits

---

The content for this website has been compiled from tutorials put together by:

- [Titus Brown](#) - [Link to Tutorial](#)
- [Johannes Koster](#) - [Link to Tutorial](#)
- [CyVerse](#) - [Link to Container Camp](#)

Please report new issues in [GitHub](#)



## CHAPTER 7

---

Coming Soon

---

- Singularity modules in HPCs
- Building containers from scratch

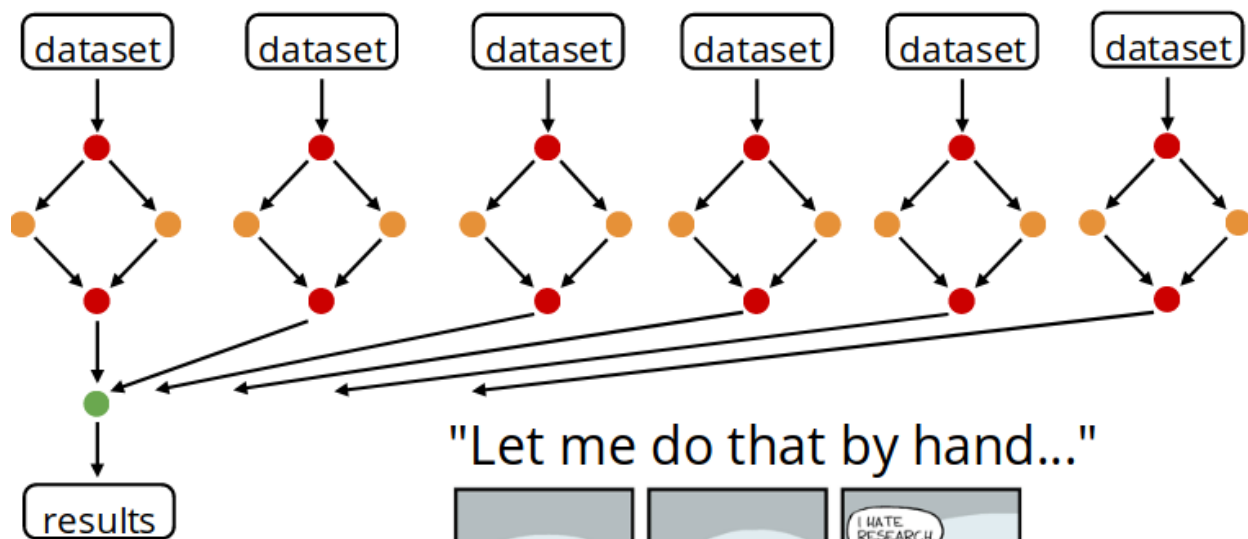


- Version Control
- Markdown
- Did someone say *Kubernetes* !!!



## CHAPTER 8

### Introduction



"Let me do that by hand..."



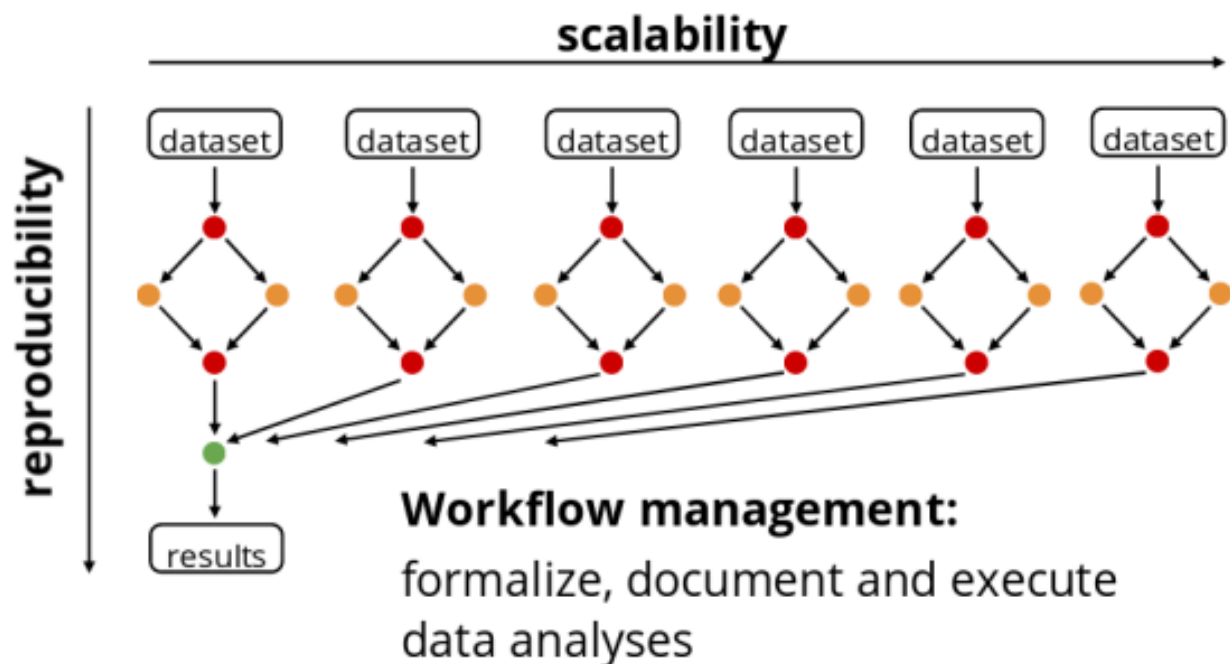




## Why Snakemake?

A common pattern in scientific computing involves the execution of many computational or data manipulation tasks. Those tasks are usually coupled, i.e., data produced by one task are consumed by one or more other tasks. Thus execution of such tasks often requires a non-trivial coordination (orchestration) to satisfy their data dependencies.

**Snakemake** is a workflow management system that .



Snakemake [slides](#)

- Snakemake allows you to create a set of rules, each one defining a “step” of your analysis. Rules can either use shell commands, plain Python code or external Python or R scripts to create output files from input files. [Original Paper](#)

The workflow is implied by dependencies between the rules that arise from one rule needing an output file of another as an input file.

For each rule you need to provide:

- **Input** : Data files, scripts, executables or any other files.
- **Expected output**. It's not required to list all possible outputs. Just those that you want to monitor or that are used by a subsequent step as inputs.
- A **command** to run to process the input and create the output.

```
rule myname:
    input: ['myinput1', 'myinput2']
    output: ['myoutput']
    shell: 'Some command to go from in to out'
```

## 9.1 Snakemake Advantages:

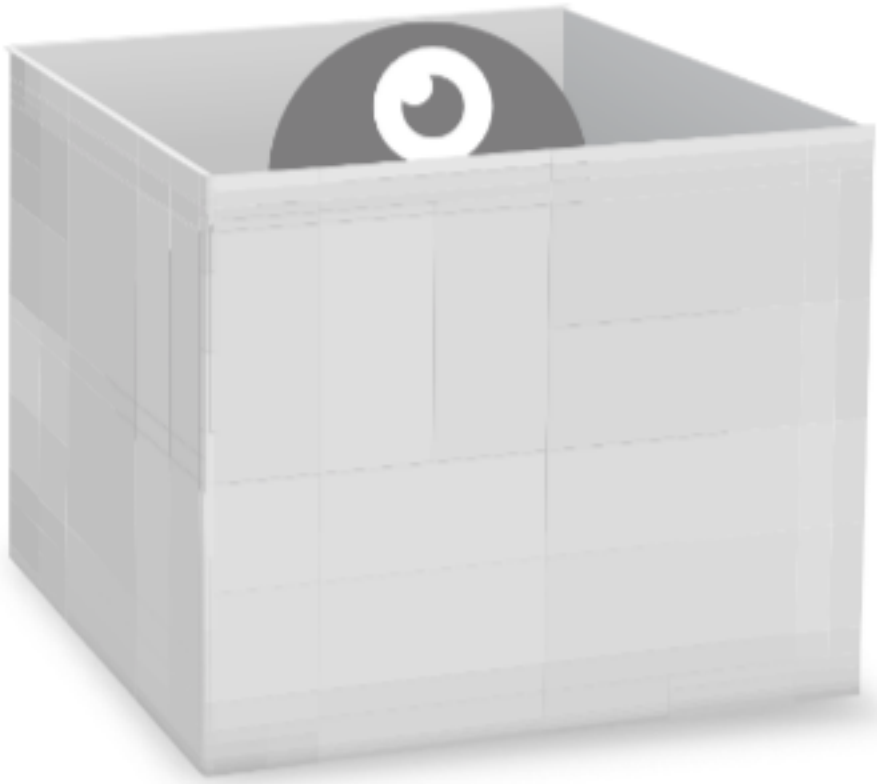
- **Memory**: stores the architecture of the work and allows you to rapidly perform a complete or partial analysis.
- **Portability**: if built in a user-sensitive way, anyone working on the same analysis can obtain every results.
- **Modularity**: as a rule is only a path between an input and an output, if one method or technique used in the analysis needs to be updated or modified, it can be replaced by a new rule with the same output. It allows a parallel comparison of different techniques in a very simple way.
- **Reproducibility**: if you're analysing a data sample collected by the same experiment in different years of exploitation and you plan to analyse data samples not even collected yet, Snakemake is a simple solution to perform the same analysis to different data sets.
- **Snakemake workflows can be easily executed on workstations, clusters, the grid, and in the cloud without modification**. The job scheduling can be constrained by arbitrary resources like e.g. available CPU cores, memory or GPUs.
- **Snakemake can automatically deploy required software dependencies of a workflow using Conda or Singularity**.
- Snakemake can use Amazon S3, Google Storage, Dropbox, FTP, WebDAV, SFTP and iRODS to access input or output files and further access input files via HTTP and HTTPS.

## CHAPTER 10

---

### Why Containers?

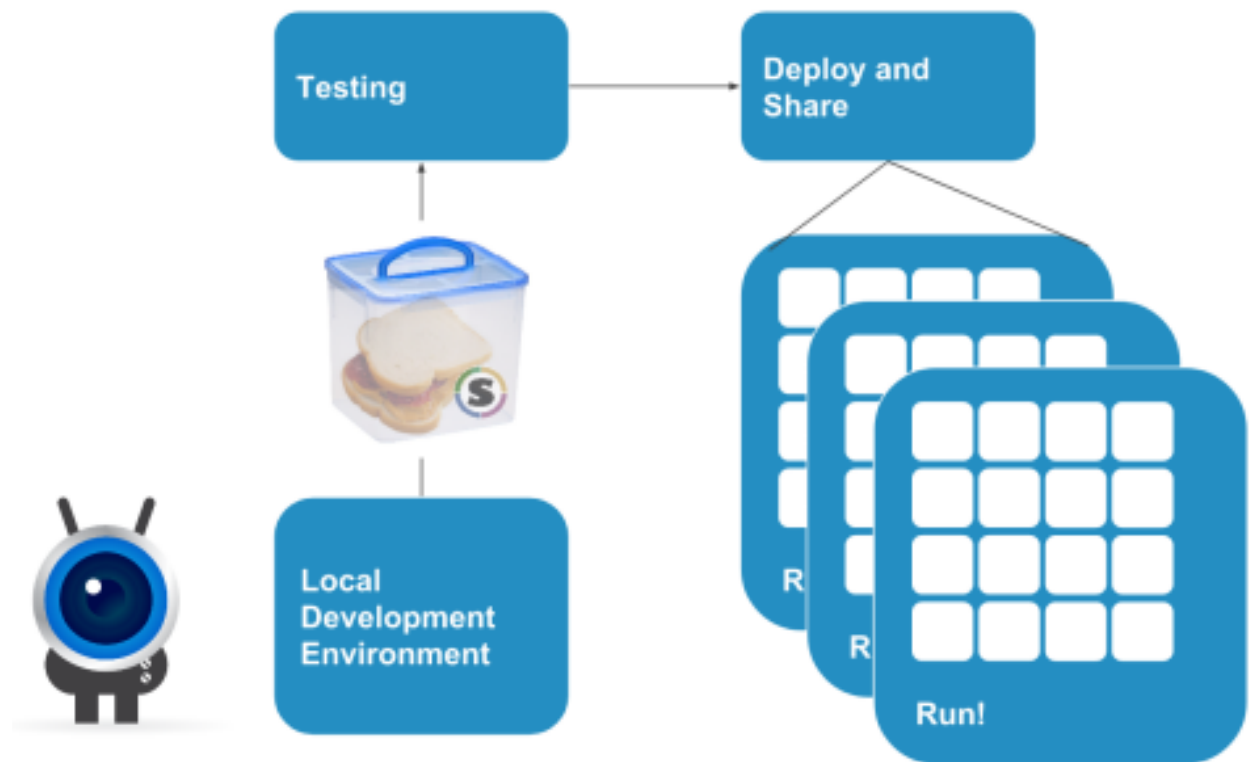
---



**A container image is an encapsulated, portable environment that is created to distribute a scientific analysis or a general function. Containers help with reproducibility of such content as they nicely package software and data dependencies, along with libraries that are needed.**

- Hubs hosts over 100,000 pre-built, ready-to-use containers
- examples of things you can do with containers:

- Package an analysis pipeline so that it runs on your laptop, in the cloud, and in a high performance computing (HPC) environment to produce the same result.
- Publish a paper and include a link to a container with all of the data and software that you used so that others can easily reproduce your results.
- Install and run an application that requires a complicated stack of dependencies with a few keystrokes.
- Create a pipeline or complex workflow where each individual program is meant to run on a different operating system.



- [Amazing slides on Singularity](#) by *Vanessa Sochat*

## CHAPTER 11

---

### Snakemake Basic Tutorial

---

Binder



## CHAPTER 12

---

### Background

---

The genome of a living organism encodes its hereditary information. It serves as a blueprint for proteins, which form living cells, carry information and drive chemical reactions. DNA sequencing, producing gigabytes of data from a single biological sample (e.g. a biopsy of some tissue). For technical reasons, DNA sequencing cuts the DNA of a sample into millions of small pieces, called reads. In order to recover the genome of the sample, one has to map these reads against a known reference genome (e.g., the human one obtained during the famous human genome project). This task is called read mapping.





# CHAPTER 13

---

## Getting started - your first Snakefile

---

- Create a new text Snakefile (File, New File, Text file) and write:

```
rule fastqc_a_file:
  shell:
    "fastqc data/0Hour_001_1.fq.gz"
```

- Then save it as a file named Snakefile.
- Now, run snakemake:

```
$ snakemake
```

- and you should see:

```
Building DAG of jobs...
Using shell: /bin/bash
Provided cores: 1
Rules claiming more threads will be scaled down.
...
Approx 95% complete for 0Hour_001_1.fq.gz
Analysis complete for 0Hour_001_1.fq.gz
[Wed Feb 27 13:09:51 2019]
Finished job 0.
1 of 1 steps (100%) done
Complete log: /home/jovyan/.snakemake/log/2019-02-27T130941.260352.snakemake.log
```

Upon execution there will be two new files, '0Hour\_001\_1\_fastqc.html' & '0Hour\_001\_1\_fastqc.zip'

Note: snakemake configuration file is by default called Snakefile

## 13.1 Updating the Snakefile to track inputs and outputs

At the moment this is basically just a shell script with extra syntax... what's the point?

Well, shell scripts - and this snakefile, too - will rerun the command every time you run the file, even if there's no reason to do so because the file hasn't changed.

**Digression:** This is particularly important for large or long workflows, where you're dealing with 10s to 100s of files that may take hours to days to process! It can be hard to figure out which files to rerun, but (spoiler alert) snakemake can really help you do this!

**It's hard to track this kind of thing in a shell script - I usually just comment out the lines I don't want run, or break my commands up into multiple shell scripts so they don't take so long - but with snakemake, you can annotate the rule with input and output files!**

- Change your snakefile to look like this:

```
rule fastqc_a_file:
    input:
        "data/0Hour_001_1.fq.gz"
    output:
        "data/0Hour_001_1_fastqc.html",
        "data/0Hour_001_1_fastqc.zip"
    shell:
        "fastqc data/0Hour_001_1.fq.gz"
```

here, we've annotated the rule with the required **input** file, as well as the expected **output** files.

Question: how do we know what the output files are?

- Now run:

```
$ snakemake
```

and you should see:

```
Building DAG of jobs...
Nothing to be done.
Complete log: /home/jovyan/.snakemake/log/2019-02-27T132031.813143.snakemake.log
```

### What happened??

snakemake looked at the file, saw that the output files existed, and figured out that it didn't need to do anything!

## 13.2 Forcibly re-running things

- You can tell snakemake to run the rule no matter what with `-f`:

```
$ snakemake -f
```

- You can also remove an output file and it will automatically re-run:

```
$ rm data/*.html
$ snakemake
```

**Note that you don't need to remove *all* the output files to rerun a command - just remove *one* of them.**

- You can *also* update the timestamp on an *input* file, and snakemake will figure out that the output file is older than the input file, and rerun things.

```
$ touch data/*.fq.gz  
$ snakemake
```

This will become important later :)



---

## Multiple rules

---

- Let's add a rule to run fastqc on a second file:

```
rule fastqc_a_file:
    input:
        "data/0Hour_001_1.fq.gz"
    output:
        "data/0Hour_001_1_fastqc.html",
        "data/0Hour_001_1_fastqc.zip"
    shell:
        "fastqc data/0Hour_001_1.fq.gz"

rule fastqc_a_file2:
    input:
        "data/6Hour_001_1.fq.gz"
    output:
        "data/6Hour_001_1_fastqc.html",
        "data/6Hour_001_1_fastqc.zip"
    shell:
        "fastqc data/6Hour_001_1.fq.gz"
```

Now, if you run this, the Right Thing won't happen: snakemake will do nothing. Why?

- Well, **snakemake only runs the *first* rule in a Snakefile, by default.** You can give a rule name on the command line, if you like, or you can tell snakemake what output file(s) you want. Let's do the latter:

```
$ snakemake fastqc_a_file2
```

and now you should see the second fastqc command run, with the appropriate output files!

**Note that snakemake only runs the second rule, because it looks at the output files and sees that the first file you wanted, 0Hour\_001\_1\_fastqc.html already exists!**

Points to note:

- this Snakefile is pretty long compared to the same shell script. . .
- specifying which file or rule you want to run explicitly is kind of annoying. . .



---

## A first refactoring: adding a better default rule

---

Let's start refactoring (cleaning up) this Snakefile.

- First, let's add a rule at the top:

```
rule all:
  input:
    "data/0Hour_001_1_fastqc.html",
    "data/6Hour_001_1_fastqc.html"

rule fastqc_a_file:
  input:
    "data/0Hour_001_1.fq.gz"
  output:
    "data/0Hour_001_1_fastqc.html",
    "data/0Hour_001_1_fastqc.zip"
  shell:
    "fastqc data/0Hour_001_1.fq.gz"

rule fastqc_a_file2:
  input:
    "data/6Hour_001_1.fq.gz"
  output:
    "data/6Hour_001_1_fastqc.html",
    "data/6Hour_001_1_fastqc.zip"
  shell:
    "fastqc data/6Hour_001_1.fq.gz"
```

this rule, by convention called **all**, is a default rule that produces all the output files. But it's a bit weird! It's all input, and no output!

This is a blank rule that gathers together all of the various files you want produced, and says “hey, snakemake, I depend on all of these files for my input - make them for me!” And then, once those files are all there, it ... does nothing.

*Yep, this is perfectly legal in snakemake, and it's one way to make your life easier.*

- Note that `snakemake -f` no longer works properly, because `-f` only forces rerunning a single rule. To rerun everything, you can run `touch data/*.fq.gz` to make all the output files stale; or `rm data/*.html` to remove some of the output files.



---

### A second refactoring: doing a bit of templating

---

There's a lot of repetition in each of these rules. Let's collapse it down a little bit by replacing the filename in the `fastqc` command with a magic variable, `{input}`.

```
rule all:
  input:
    "data/0Hour_001_1_fastqc.html",
    "data/6Hour_001_1_fastqc.html"

rule fastqc_a_file:
  input:
    "data/0Hour_001_1.fq.gz"
  output:
    "data/0Hour_001_1_fastqc.html",
    "data/0Hour_001_1_fastqc.zip"
  shell:
    "fastqc {input}"

rule fastqc_a_file2:
  input:
    "data/6Hour_001_1.fq.gz"
  output:
    "data/6Hour_001_1_fastqc.html",
    "data/6Hour_001_1_fastqc.zip"
  shell:
    "fastqc {input}"
```

This all works as before, but now the rule is a bit more generic and will work with any input file. Sort of.



---

Refactoring 3: templating output files, too

---

What do I mean, sort of?

Well, the output **filenames ALSO depend on the input file names in some way** - specifically, **fastqc replace part of the filename with `_fastqc.html` and `_fastqc.zip` to make its two output files.**

- Let's rewrite the rule using some **snakemake pattern matching**:

```
rule all:
  input:
    "data/0Hour_001_1_fastqc.html",
    "data/6Hour_001_1_fastqc.html"

rule fastqc_a_file:
  input:
    "{filename}.fq.gz"
  output:
    "{filename}_fastqc.html",
    "{filename}_fastqc.zip"
  shell:
    "fastqc {input}"

rule fastqc_a_file2:
  input:
    "{filename}.fq.gz"
  output:
    "{filename}_fastqc.html",
    "{filename}_fastqc.zip"
  shell:
    "fastqc {input}"
```

**What we've done here is tell snakemake that anytime we say we *want* a file that ends with `_fastqc.html`, it should look for a file that ends in `.fq.gz` and then run `fastqc` on it.**

- Try running this:

```
$ snakemake
```

Oh no! We get a `AmbiguousRuleException`! What's going on?

Well, if you look at the rule above, we've given snakemake two different rules to produce the same file(s)! `fastqc_a_file` and `fastqc_a_file2` are now identical rules! snakemake doesn't like that.

- Let's remove one, to get a trimmer, leaner, and above all *functional* snakefile:

```
rule all:
    input:
        "data/0Hour_001_1_fastqc.html",
        "data/6Hour_001_1_fastqc.html"

rule fastqc_a_file:
    input:
        "{filename}.fq.gz"
    output:
        "{filename}_fastqc.html",
        "{filename}_fastqc.zip"
    shell:
        "fastqc {input}"
```

and THAT should now work just fine!

## 17.1 Adding some more files

Now here's the fun bit – if you look in the data directory, you'll see that there are actually 8 files in there. Let's modify the snakefile to run fastqc on all of them!

- How should we do that? (Give it a try!)

```
rule all:
    input:
        "data/0Hour_001_1_fastqc.html",
        "data/6Hour_001_1_fastqc.html",
        "data/6Hour_001_2_fastqc.html"

rule fastqc_a_file:
    input:
        "{arglebarf}.fq.gz"
    output:
        "{arglebarf}_fastqc.html",
        "{arglebarf}_fastqc.zip"
    shell:
        "fastqc {input}"
```

## 17.2 Rerunning snakemake

Note you can just run snakemake whenever you want. It won't do anything unless something's changed.

```
$ snakemake
```

## CHAPTER 18

---

### Building out the workflow

---

- So, we've gotten fastqc sorted out. What's next?

**Let's add in a new rule - multiqc, to summarize our fastqc results.**

- multiqc takes a directory name under which there are one or more fastqc reports, and then summarizes them.
- Running it on the command line,

```
$ multiqc data
```

**you can see that it creates two outputs, multiqc\_report.html and the directory multiqc\_data/ which contains a bunch of files. Let's create a snakemake rule for this; add:**

```
rule run_multiqc:
    output:
        "multiqc_report.html",
        directory("multiqc_data")
    shell:
        "multiqc data/"
```

**to the bottom of the file. (Note, you need to tell snakemake if an output is a directory.)**

- Now run it:

```
$ snakemake run_multiqc
```

**This ...doesn't really do what we want, for a few reasons.**

1. **you have to specify the rule name or else snakemake doesn't run anything. How do we fix this??**
2. **multiqc\_report.html already exists, so snakemake doesn't run the rule. How do we actually test the rule??**
3. **the multiqc rule has no input dependencies. How do we specify them??**

- Let's fix the first two things first:
  - **add multiqc\_report.html to the inputs for the first all.**

– then remove `multiqc_report.html` and re-run snakemake.

- Your snakefile should look like:

```
rule all:
    input:
        "data/0Hour_001_1_fastqc.html",
        "data/6Hour_001_1_fastqc.html",
        "multiqc_report.html"

rule fastqc_a_file:
    input:
        "{filename}.fq.gz"
    output:
        "{filename}_fastqc.html",
        "{filename}_fastqc.zip"
    shell:
        "fastqc {input}"

rule run_multiqc:
    output:
        "multiqc_report.html",
        directory("multiqc_data")
    shell:
        "multiqc data/"
```

Yay, that seems to work!

Points to note:

- other than the first rule, rules can be in any order
- the rule name doesn't really matter, it's mostly for debugging. It just needs to be “boring” (text, underscores, etc. only)

## 18.1 Providing input files explicitly to the multiqc rule

The third problem, that multiqc doesn't have any input dependencies, is a bit harder to fix.

(Why do we want to fix this? Well, this is how snakemake tracks “out of date” files - if we don't specify input dependencies, then we may update one of the fastqc results that multiqc uses, but snakemake won't re-run multiqc on it, and our multiqc results will be out of date.)

Basically we need to tell snakemake *all* of the files that we want. On the face of it, that's easy – change the rule like so:

```
rule run_multiqc:
    input:
        "data/0Hour_001_1_fastqc.html",
        "data/6Hour_001_1_fastqc.html",
    output:
        "multiqc_report.html",
        directory("multiqc_data")
    shell:
        "multiqc data/"
```

This will work, but there are two reasons this is not great.

First, we're repeating ourselves. Those input files are in the `all` rule above, and also in this rule. If you are as forgetful as I am, this means that you run the risk of updating one rule and not the other, and getting out of sync.

Second, we're explicitly listing out the files that we want produced. That's not super great because it makes it annoying to add files.

We can fix the first issue by using "variables". The second issue requires a bit more advanced programming, and we'll leave it to the very end.

To use variables, let's make a Python list at the very top, containing all of our expected output files from `fastqc`:

```
fastqc_output = ["data/0Hour_001_1_fastqc.html", "data/6Hour_001_1_fastqc.html",
                 "data/0Hour_001_2_fastqc.html", "data/6Hour_001_2_fastqc.html",
                 "data/0Hour_002_1_fastqc.html", "data/6Hour_002_1_fastqc.html",
                 "data/0Hour_002_2_fastqc.html", "data/6Hour_002_2_fastqc.html"]
```

and modify the `all` and `multiqc` rules to contain this list. The final snakefile looks like this:

```
fastqc_output = ["data/0Hour_001_1_fastqc.html", "data/6Hour_001_1_fastqc.html",
                 "data/0Hour_001_2_fastqc.html", "data/6Hour_001_2_fastqc.html",
                 "data/0Hour_002_1_fastqc.html", "data/6Hour_002_1_fastqc.html",
                 "data/0Hour_002_2_fastqc.html", "data/6Hour_002_2_fastqc.html"]

rule all:
    input:
        fastqc_output,
        "multiqc_report.html"

rule fastqc_a_file:
    input:
        "{filename}.fq.gz"
    output:
        "{filename}_fastqc.html",
        "{filename}_fastqc.zip"
    shell:
        "fastqc {input}"

rule run_multiqc:
    input:
        fastqc_output
    output:
        "multiqc_report.html",
        directory("multiqc_data")
    shell:
        "multiqc data/"
```

**\*\*Points to note:**

- quoted strings are ... strings (filename, usually)
- you can also use Python to create, manipulate, etc. filenames or lists of them, and it will work fine!
- this includes loading in filenames from spreadsheets, etc. - more on that later.\*\*

## 18.2 Refactoring this to make it slightly more concise –

We've got one more redundancy in this file - the `fastqc_output` is listed in the `all` rule, but you don't need it there! Why?

Well, `multiqc_report.html` is already in the `all` rule, and the `multiqc` rule depends on `fastqc_output`, so `fastqc_output` already needs to be created to satisfy the `all` rule, so... specifying it in the `all` rule is redundant! And you can remove it!

(It's not a big deal and I usually leave it in. But I wanted to talk about dependencies!)

The Snakefile now looks like this:

```
fastqc_output = ["data/0Hour_001_1_fastqc.html", "data/6Hour_001_1_fastqc.html",
                 "data/0Hour_001_2_fastqc.html", "data/6Hour_001_2_fastqc.html",
                 "data/0Hour_002_1_fastqc.html", "data/6Hour_002_1_fastqc.html",
                 "data/0Hour_002_2_fastqc.html", "data/6Hour_002_2_fastqc.html"]

rule all:
    input:
        "multiqc_report.html"

rule fastqc_a_file:
    input:
        "{filename}.fq.gz"
    output:
        "{filename}_fastqc.html",
        "{filename}_fastqc.zip"
    shell:
        "fastqc {input}"

rule run_multiqc:
    input:
        fastqc_output
    output:
        "multiqc_report.html",
        directory("multiqc_data")
    shell:
        "multiqc data/"
```

and we can rerun it from scratch by doing:

```
$ rm data/*.html multiqc_report.html
$ snakemake
```

## 18.3 Digression: what files does snakemake check in order to decide about rerunning?

**snakemake will compare only at the very initial input files, and the specific output file(s) you are requesting, to decide if it needs to rerun the workflow.**

**In practical terms, this means that if you just delete the `data/*.html` files above but leave `multiqc_report.html` around, snakemake won't rerun anything. You have to delete both the intermediaries *and* the end output files (as we do in the previous section), *or* update the raw input files, in order to force rerunning.**



(This is a feature, not a bug - it helps deal with data-intensive pipelines where the intermediate files are really big.)

## 18.4 Making a `clean` rule

It's kind of annoying to have to delete things explicitly. Snakemake should take care of that for us. Let's add a new rule, `clean`, that forces rerunning of things –

```
rule clean:
    shell:
        "rm -f {fastqc_output} multiqc_report.html"
        "rmdir -fr multiqc_data/"
```

and now try re-running things:

```
$ snakemake -p clean
$ snakemake
```

**\*\*A few things to point out:**

- Here we see the use of variables inside a shell command, again - `{fastqc_output}` means “replace the thing in the curly quotes with the Python values in `fastqc_output`”.
- We're using `snakemake -p` to get a printout of the commands that are run.\*\*

What's particularly nice about the `clean` rule (the name is a convention, not a requirement) is that you only need to keep track of the expected output files in one or two places - the all rule, and the `clean` rule.

## 18.5 Digression: running things in parallel

So, we've put all this work into making this snakefile with its input rules and its output rules... and there are a lot of advantages to our current approach already! Let's list a few of them –

- we've completely automated our analysis!
- we can easily add new data files into `fastqc` and `multiqc`!
- we can rerun things easily, and (even better) by default only things that *need* to be run will be run.
- the snakefile is actually pretty reusable - we could drop this into a new project, and, with little effort, run all of these things on new data!

**but there's even more practical value in this, too – because we've given snakemake a recipe, rather than a script, we can now run these commands in parallel, too!**

Try:

```
$ snakemake clean
$ snakemake -j 4
```

this will run up to four things in parallel!

## 18.6 Doing more things in our workflow.

Let's add some trimming!

```
rule trim_reads:
    input:
        "{filename}_1.fq.gz",
        "{filename}_2.fq.gz"
    output:
        "{filename}_1.pe.qc.fq.gz",
        "{filename}_1.se.qc.fq.gz",
        "{filename}_2.pe.qc.fq.gz",
        "{filename}_2.se.qc.fq.gz"
    shell:
        "trimmomatic PE {input} {output} LEADING:2 TRAILING:2 \
        SLIDINGWINDOW:4:15 \
        MINLEN:25"
```

Points to make:

- I'm being really careful about input and output filenames to make sure that the wildcards work properly, and that we can fastqc things properly!
- the order of files in {input} and {output} matter - they're passed in that order to snakemake! (See the output of `snakemake -p`.)

Now add the appropriate files into the fastqc output list too – change it to:

```
fastqc_output = ["data/0Hour_001_1_fastqc.html", "data/6Hour_001_1_fastqc.html",
    "data/0Hour_001_2_fastqc.html", "data/6Hour_001_2_fastqc.html",
    "data/0Hour_002_1_fastqc.html", "data/6Hour_002_1_fastqc.html",
    "data/0Hour_002_2_fastqc.html", "data/6Hour_002_2_fastqc.html",
    "data/0Hour_001_1.pe.qc_fastqc.html", "data/0Hour_002_1.pe.qc_fastqc.html",
    "data/6Hour_001_1.pe.qc_fastqc.html", "data/6Hour_002_1.pe.qc_fastqc.html",
    "data/0Hour_001_2.pe.qc_fastqc.html", "data/0Hour_002_2.pe.qc_fastqc.html",
    "data/0Hour_001_2.pe.qc_fastqc.html", "data/0Hour_002_2.pe.qc_fastqc.html"]
```

and re-run everything:

```
$ snakemake clean
$ snakemake -j 4
```

and voila - you've got a pile of trimmed output files, and a nice multiqc report on pre- and post- quality!

**One other point to make:** by naming the files “right”, i.e. by making the trimmed files meet the input requirements for the fastqc rule, we could automatically take advantage of that rule to run fastqc on them. A lot of snakemake seems to boil down to file naming, for better or for worse... :)

### 19.1 Dry run

You can use `snakemake -n` to see what *would* be run, without actually running it! This is called a “dry run”. This is useful when you have really big compute.

```
$ snakemake clean
$ snakemake -n
```

### 19.2 Running things in parallel, revisited

Above, we saw that you can use `snakemake -j 4` to run four jobs in parallel. Here are some other

- all the output is sort of smushed together. . . if a rule fails, it may be hard to figure out what happened. You can always just rerun `snakemake`.
- still need to be careful about how much memory and processor you’re using!
- can be used on a cluster, to distribute jobs across multiple compute nodes. (This requires more work; also see, Specifying required software.)

### 19.3 Specifying software required for a rule

You can specify software on a per-rule basis! This is really helpful when you have incompatible software requirements for different rules, or want to run on a cluster, or just want to pin your `snakemake` workflow to a specific version.

For example, if you create a file `env_fastqc.yml` with the following content,

```
channels:
- bioconda
- defaults
- conda-forge
dependencies:
- fastqc==0.11.8
```

and then change the fastqc rule to look like this:

```
rule fastqc_a_file:
    input:
        "{filename}.fq.gz"
    output:
        "{filename}_fastqc.html",
        "{filename}_fastqc.zip"
    conda:
        "env_fastqc.yml"
    shell:
        "fastqc {input}"
```

you can now run snakemake like so,

```
$ snakemake --use-conda
```

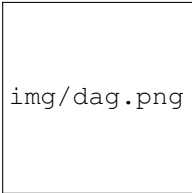
and for that rule, snakemake will install just that software, with the specified version.

This aids in reproducibility, in addition to the practical advantages of isolating software installs from each other.

## 19.4 Outputting the entire workflow diagram

You can visualize your workflow like so!

```
$ snakemake --dag | dot -Tpng > dag.png
```

img/dag.png

- The DAG png file should look something like this,

## 19.5 Snakemake Report

Snakemake can automatically generate detailed self-contained HTML reports that encompass runtime statistics, provenance information, workflow topology and results.

- To create the report simply run

```
$ snakemake --report report.html
```

- Sample report here(xyz)

## 19.6 Adding in some Python...

- You can add in some Python to load in the input files, like so:

```
import glob, sys
fastqc_input = glob.glob('data/?Hour_00?_?.fq.gz')

fastqc_output = []
for filename in fastqc_input:
    new_filename = filename.split('.')[0] + '_fastqc.html'
    fastqc_output.append(new_filename)

for filename in fastqc_input:
    new_filename = filename.split('.')[0] + '.pe.qc_fastqc.html'
    fastqc_output.append(new_filename)

print('from these input files', fastqc_input, file=sys.stderr)
print('I constructed these output filenames', fastqc_output, file=sys.stderr)

rule all:
    input:
        "multiqc_report.html"

rule clean:
    shell:
        "rm -f {fastqc_output} multiqc_report.html"

rule fastqc_a_file:
    input:
        "{arglebarf}.fq.gz"
    output:
        "{arglebarf}_fastqc.html",
        "{arglebarf}_fastqc.zip"
    conda:
        "env_fastqc.yml"
    shell:
        "fastqc {input}"

rule run_multiqc:
    input:
        fastqc_output
    output:
        "multiqc_report.html",
        directory("multiqc_data")
    shell:
        "multiqc data/"

rule trim_reads:
    input:
        "{filename}_1.fq.gz",
        "{filename}_2.fq.gz"
    output:
        "{filename}_1.pe.qc.fq.gz",
        "{filename}_1.se.qc.fq.gz",
        "{filename}_2.pe.qc.fq.gz",
        "{filename}_2.se.qc.fq.gz"
```

(continues on next page)

(continued from previous page)

```
shell:
"trimmomatic PE {input} {output} LEADING:2 TRAILING:2 \
  SLIDINGWINDOW:4:15 \
  MINLEN:25"
```

Take a look at `glob_wildcards` as well.

---

### Final thoughts - writing your own snakefile

---

**Just like scripting, or writing an R script, writing a snakefile is a kind of programming. So you'll have to do a lot of debugging :) :(.**

- A few thoughts for you on how to do this:
- start small, grow your snakefile!
- DO copy and paste from this tutorial and others you find online!
- it rarely hurts to just re-run snakemake!

#### 20.1 Dealing with complexity

- Workflows can get really complicated; [here](#), for example, is one for our most recent paper. But it's all just using the building blocks that I showed you above!
- If you want to see some good examples of how to build nice, clean, simple looking workflows, check out [this RNAseq example](#).





---

### Thinking about workflows - a strong(er) argument

---

what do (snakemake) workflows do for me?

- declarative vs procedural specification
  - allows analysis of workflow graph, parallelization
  - allows declaration of specific software needed, for later tracking
  - can compare workflows more easily, in theory (=> reproducibility)
  - can rerun failed steps
- tracks dependencies on files
  - allows rerunning just the bits that need to rerun
- reusability, in theory
- different conda environments for each step
  - allows pinning of software versions
  - allows use of potentially incompatible software

**The main reason to use snakemake is that it lets be sure that my workflow completed properly. snakemake tracks which commands fails, and will stop the workflow in its tracks! This is not something that you usually do in shell scripts.**



## CHAPTER 22

---

Go forth and Workflow!

---



---

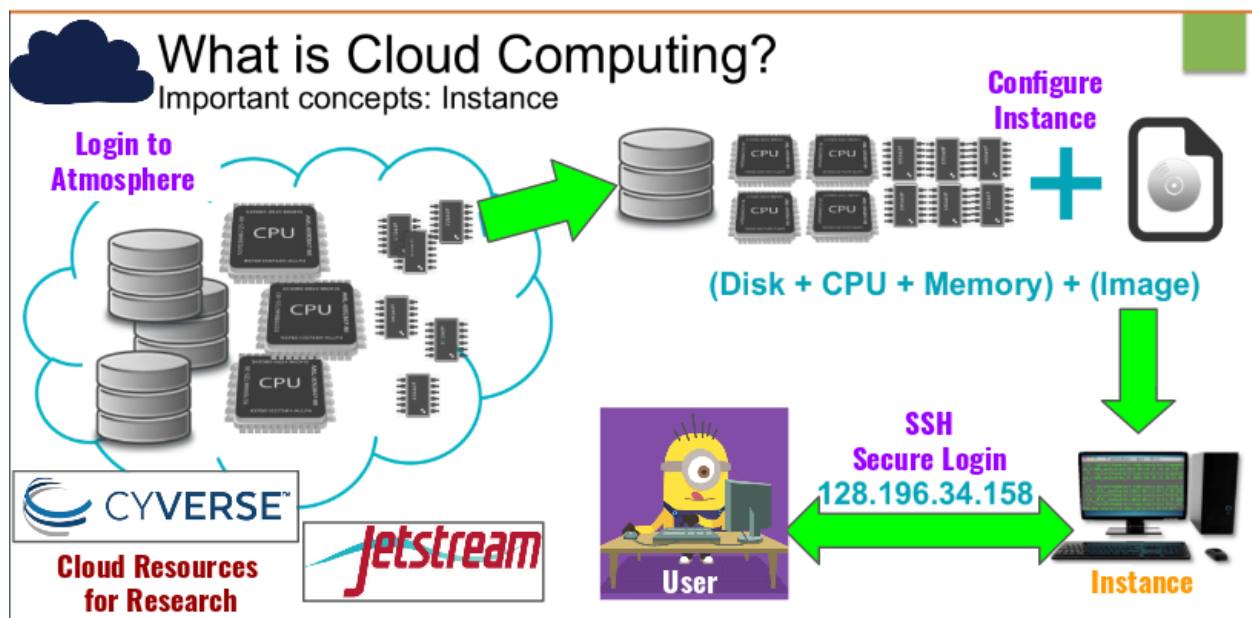
Snakemake Tutorial on CyVerse cloud

---

Learning Objective

- **Cloud Computing**
- Launch and work on remote instances (virtual machines) with pre-built images
- 

To run Snakemake jobs via [Singularity](#) / [Docker](#) containers in the cloud, we recommend [Atmosphere](#), CyVerse's cloud-computing platform which provides easy-to-use web-access to cloud resources and is designed to accommodate computationally and data-intensive tasks.



1. Register for a free account on CyVerse [here](#)
2. Follow this [Guide to Launching Atmosphere Instances](#)

### 3. [Click here for Snakemake Tutorial on Atmosphere](#)

The Snakemake2019 image supports: 1) Miniconda3 2) bio-conda 3) Snakemake 4) Singularity 5) Docker

#### Atmosphere / Jetstream Key Features:

- Access virtual machine images pre-configured with an operating system and software to help you do scientific computations in domain-specific tasks
- Find and use tools with the intuitive self-service portal
- Easily manage virtual machines
- Publish your own software suites, create your own work environments, and run the software for community use
- Access your data in the Data Store, including high-performance computing (HPC) and grid computing environments
- Integrate with existing infrastructure components using API services
- Easily generate and manage statistical reporting of user resources for total CPU hours and memory usages, total instances and applications launched by user, cloud monitoring, and on-demand intelligence resource allocation

**CyVerse** is a cyber-infrastructure initiative funded by the National Science Foundation's Directorate for Biological Sciences to address the growing needs for highly configurable and customized computational infrastructure to support research efforts in data sciences.

- [CyVerse Allocations Wiki](#)

**Jetstream** is an [NSF/XSEDE] resource designed to promote and provide configurable cyberinfrastructure in the form of cloud computing to both novice and experienced users. Jetstream features a web-based user interface based on the popular Atmosphere cloud computing environment developed by CyVerse.

#### 1. [Register for an account on XSEDE User Portal](#)

- [Jetstream Allocations Wiki](#)

---

### Accessing The Atmosphere Cloud

---



**Atmosphere, CyVerse’s cloud-computing platform allows you to launch your own isolated virtual machine (VM) image and software, using compute resources such as CyVerse-provided software suites, and pre-configured, frequently used analysis routines, relevant algorithms, and datasets.**

To request access to Atmosphere, login to the [CyVerse User Portal](#). In the [Services](#) menu under ‘MY SERVICES’ you should see Atmosphere listed as an option you can launch. If not, look under the [Available](#) menu, and click the ‘REQUEST ACCESS’ link. You will receive an email requesting additional information.

- This is the first and last place in these lessons where it will matter if you are using PC, Mac, or Linux. After we connect to our virtual machines built using the same image; we will all be on the same operating system/computing environment.
- CyVerse documentation on how to use Atmosphere resources can be found [here](#)

**NOTE: WINDOWS users** will need to install a UNIX-ready terminal to SSH-login (if you haven’t already). We recommend - [mobaxterm home edition](#)



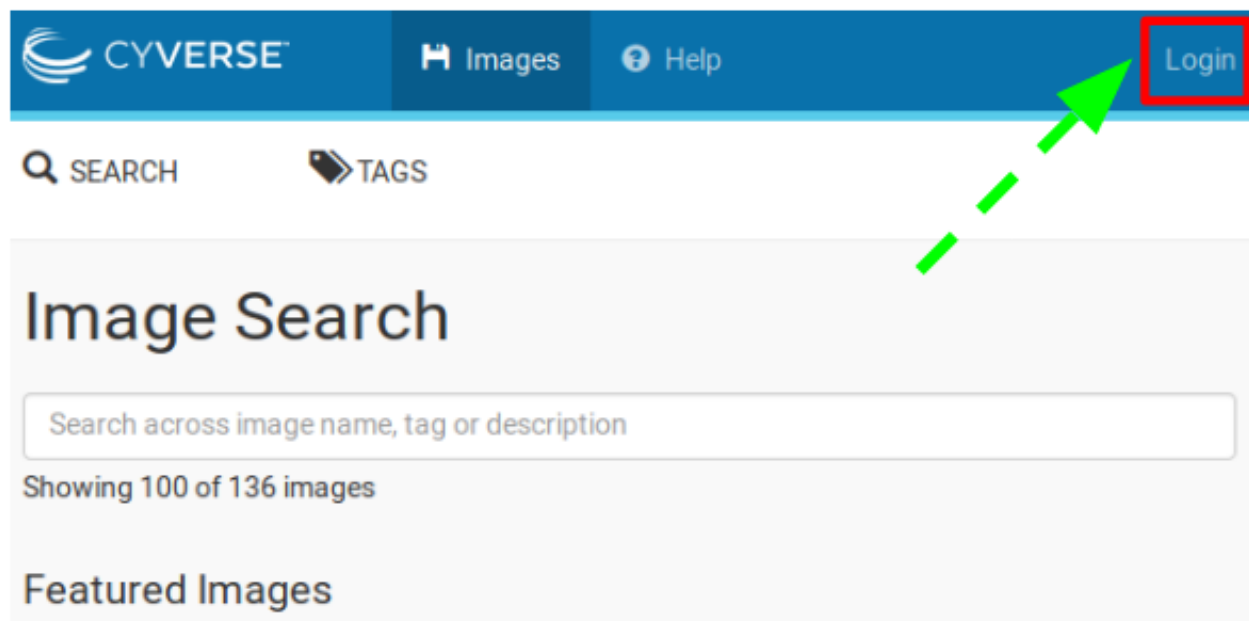


---

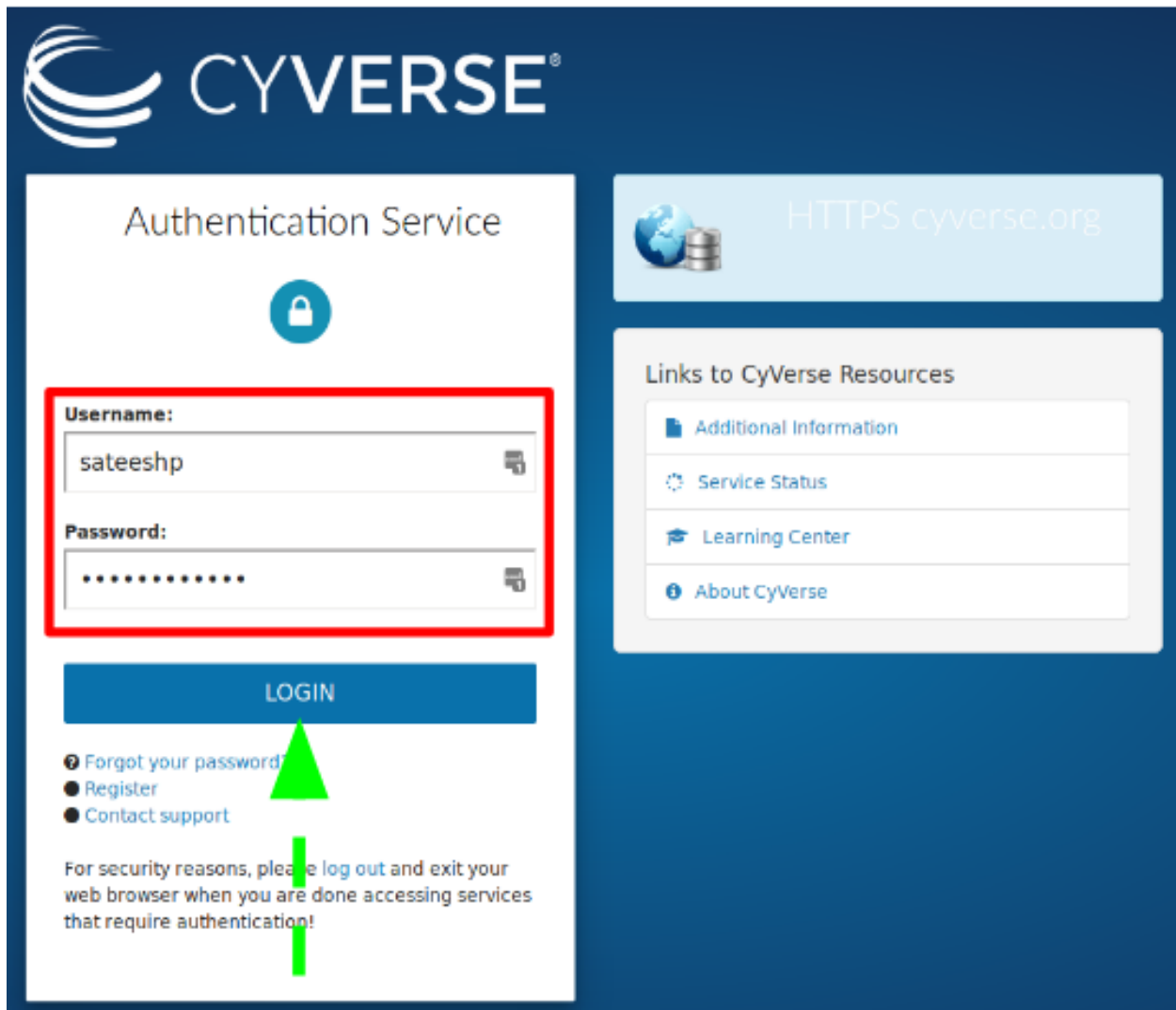
### Login & Launch Instance

---

- Login to [Atmosphere](#) by clicking the “login” button towards the right-upper corner of the screen.



- Fill in your CyVerse username and password and click “LOGIN”



The image shows the CyVerse Authentication Service login page. The page has a dark blue header with the CyVerse logo. Below the header, there's a white box containing the login form. The form has two input fields: "Username:" with the value "sateeshp" and "Password:" with masked characters. A red rectangle highlights these fields. Below the fields is a blue "LOGIN" button. To the right of the login form, there's a light blue box with the URL "HTTPS cyverse.org" and a globe icon. Below that, there's a white box titled "Links to CyVerse Resources" with four links: "Additional Information", "Service Status", "Learning Center", and "About CyVerse". At the bottom of the login form, there are links for "Forgot your password?", "Register", and "Contact support". A green arrow points from the "LOGIN" button down to the "Projects" tab in the next screenshot.

CYVERSE®

Authentication Service

Username:

sateeshp

Password:

.....

LOGIN

Forgot your password?

Register

Contact support

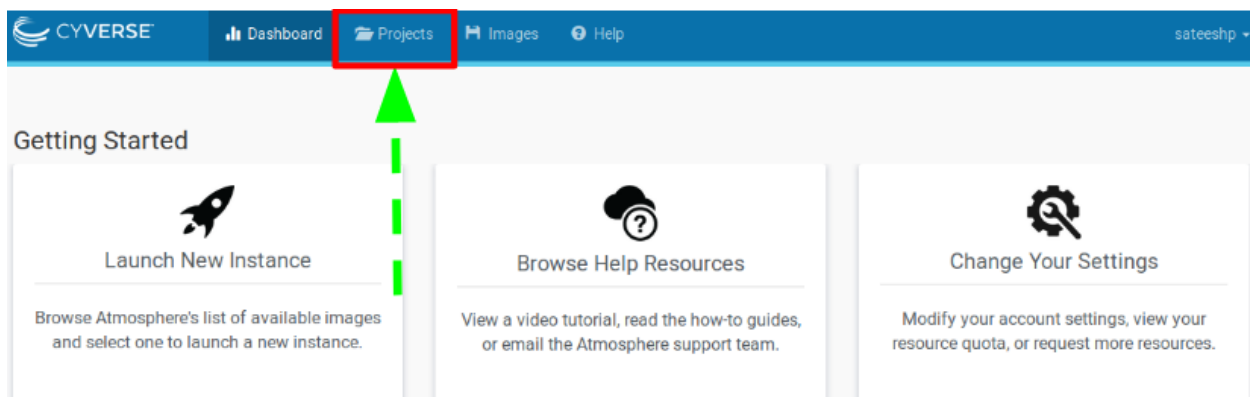
For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

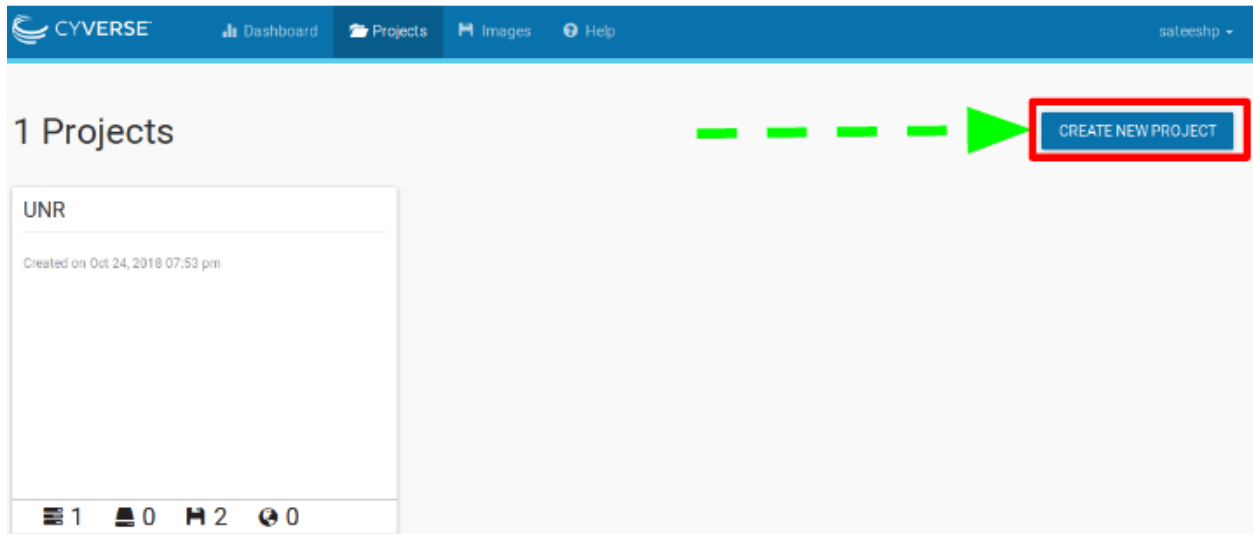
HTTPS cyverse.org

Links to CyVerse Resources

- Additional Information
- Service Status
- Learning Center
- About CyVerse

- Select the “Projects” tab and then click the “CREATE NEW PROJECT” button



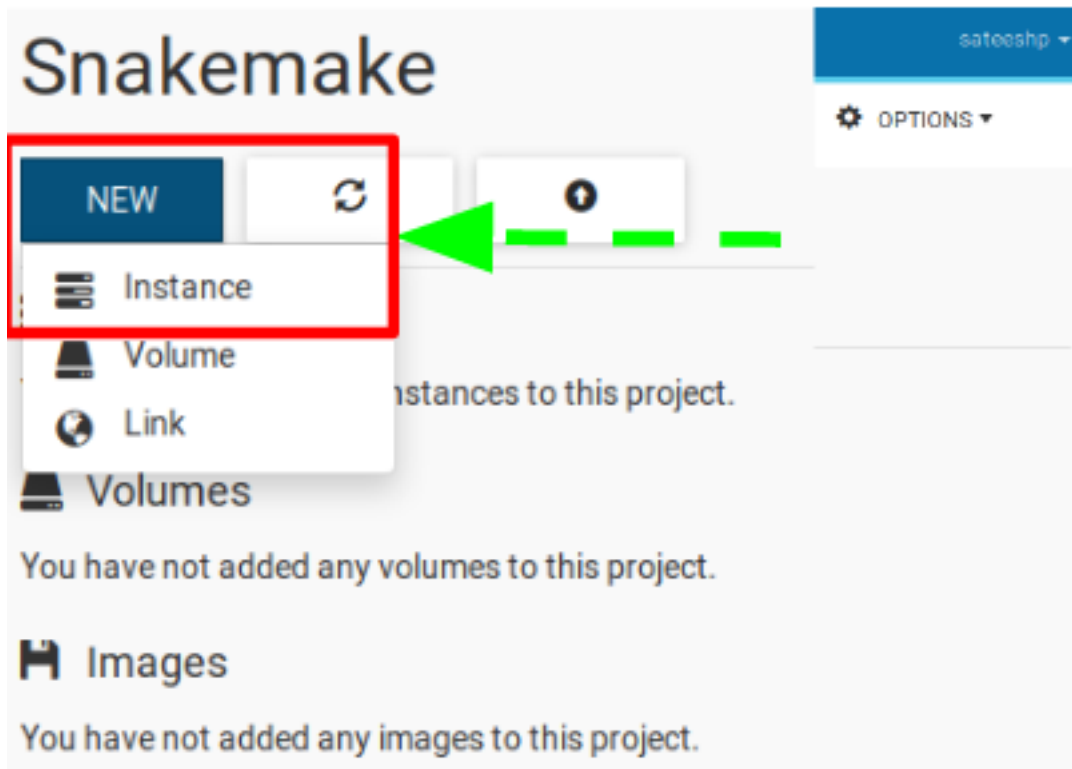


- Give your Project folder a name (Description is optional). Then click “CREATE”.

The 'Create Project' dialog box is shown with a red border. It contains two input fields: 'Project Name' with the text 'Snakemake' and 'Description' with the text 'Snakemake & Singularity Tutorial'. A green dashed arrow points upwards from the bottom towards the description field. At the bottom right of the dialog are 'CANCEL' and 'CREATE' buttons.

- Click on your newly created project and then Click on “NEW” and then “Instance” from the drop-down

menu to start up a new virtual machine.



- To select an image click on “Show All” tab and Search for “Snakemake2019” and choose the “Snake-make2019” image created by ‘sateeshp’.

Launch an Instance / Select an Image ×

First choose an image for your instance

Show Featured

Show Favorites

**Show All**

Snakemake2019|

Showing 1 image(s) for "Snakemake2019"

	<b>Snakemake2019</b>	Software: - Miniconda 4.6.8 - Rstudio	bioconda	docker	miniconda
	Mar 18, 2019 04:11 am by <b>sateeshp</b>	1.1.453 - Singularity 2.6.1 - Docker 18.09.2 Snakemake 5.4.0 Tools for RNAseq analysis: - fastqc 0.11.8 - multi...	singularity	snakemake	

[⚙️ Advanced Options](#)

CANCEL

LAUNCH INSTANCE

**\*\*You will be presented with options to choose and configure your virtual machine here:**

```
+ Instance Name: e.x., "Snake-Tutorial" or you can leave it default which is the
  ↳ image name.

+   Base Image Version: "1.0"

+   Project: select project folder to host the instance

+   Allocation Source: should be your cyverse account

+   Provider: "CyVerse Cloud - Marana"

+ Instance size: We recommend "tiny1" (CPU: 1, Mem: 4GB, Disk: 30GB) for this
  ↳ tutorial; Though depending on your allocations, choose most suitable one. [Click
  ↳ here] (https://wiki.cyverse.org/wiki/display/atmman/
  ↳ Requesting+More+Atmosphere+Resources) to read more about allocations.**
```

Launch an Instance / Basic Options

Basic Info

Instance Name

Snakemake2019

Base Image Version

1.0

Project

Snakemake

Resources

Allocation Source

sateeshp

Provider

CyVerse Cloud - Marana

Instance Size

tiny1 (CPU: 1, Mem: 4 GB, Disk: 30 GB)

tiny1 (CPU: 1, Mem: 4 GB, Disk: 30 GB)

tiny2 (CPU: 1, Mem: 8 GB, Disk: 60 GB)

small1 (CPU: 2, Mem: 8 GB, Disk: 30 GB)

small2 (CPU: 2, Mem: 16 GB, Disk: 120 GB)

medium1 (CPU: 4, Mem: 8 GB, Disk: 80 GB)

medium2 (CPU: 4, Mem: 16 GB, Disk: 160 GB)

medium3 (CPU: 4, Mem: 32 GB, Disk: 240 GB)

large1 (CPU: 8, Mem: 16 GB, Disk: 160 GB)

large2 (CPU: 8, Mem: 48 GB, Disk: 320 GB)

large3 (CPU: 8, Mem: 64 GB, Disk: 480 GB)

xlarge1 (CPU: 16, Mem: 32 GB, Disk: 400 GB)

xlarge2 (CPU: 16, Mem: 64 GB, Disk: 800 GB)

xlarge3 (CPU: 16, Mem: 128 GB, Disk: 1200 GB)

← Back

⚙️ Advanced Options

- **Launch instance and wait for the build to be deployed (~ 5-10 minutes).**

Note: During the build process: scheduling-->building-->spawning-->deploying-->Networking-->N/A; Be patient! Don't reload!. Once the virtual machine is ready, the "Activity" column will show "N/A" and the "Status" column will turn green and "Active".



Activity	IP Address
Spawning	N/A

Activity	IP Address
Deploying	128.196.142.10

Activity	IP Address
N/A	128.196.142.65

- Navigate back to ‘Projects’ and click on your new instance’s name to see more information related to the instance you just created! and Copy the IP address of your instance created.

Great! We have now built our very own remote virtual machine with all the software pre-installed. Next we will use SSH-Secure-Login to access these remote instances from our laptop’s and start Snakemaking !!!.





## CHAPTER 26

---

### SSH Secure-Login

---

- MACOS & LINUX users can open a Terminal window now.
- Windows users start a new session in [mobaxterm](#)
  - Start a new session; Fill in your “remote host” the IP address of your virtual machine; select “specify username” and enter your cyverse username; Click OK.
- Establish a secure-login to the instance by typing the following:

```
$ ssh your_cyverseusername@ip_address
```

- This should log you into CyVerse and you should see a screen like this; Enter ‘yes’ and then enter your CyVerse password.

Your cursor will not move or indicate you are typing as you enter your password. If you make a mistake, hit enter and you will be prompted again.

```
sateeshp@sp:~$ ssh sateeshp@128.196.142.16
The authenticity of host '128.196.142.16 (128.196.142.16)' can't be established.
ECDSA key fingerprint is SHA256:eLr2uBeArJZfK+eZ/CpZYd7Cyv6DlnPpR1qW9dz6JYo.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '128.196.142.16' (ECDSA) to the list of known hosts.
sateeshp@128.196.142.16's password:
Welcome to
Athenosphere
Last login: Sat Mar 16 13:06:48 2019 from 71.93.90.181
sateeshp@vm142-29:~$
```

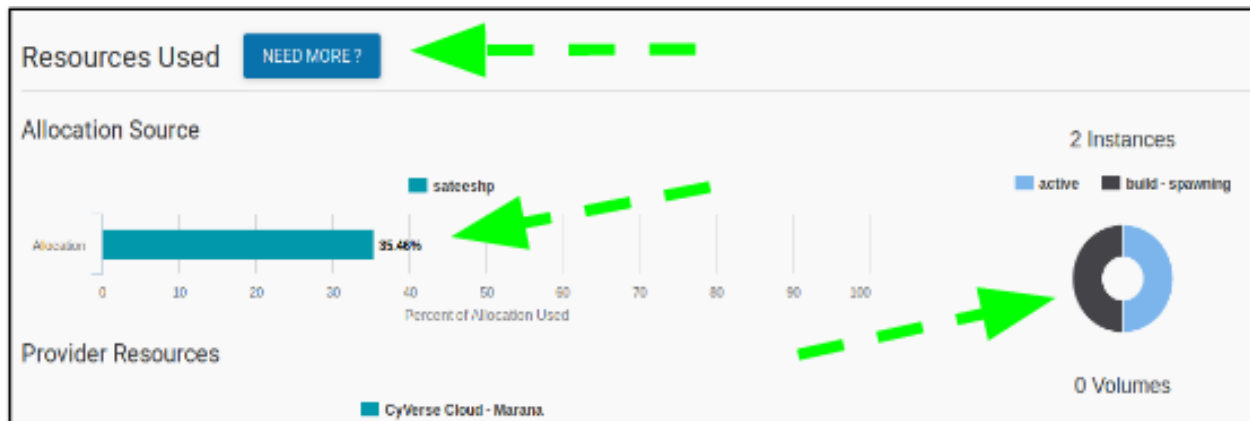
**Success !!! We have established connections with our instances. Proceed to the Tutorial section.**



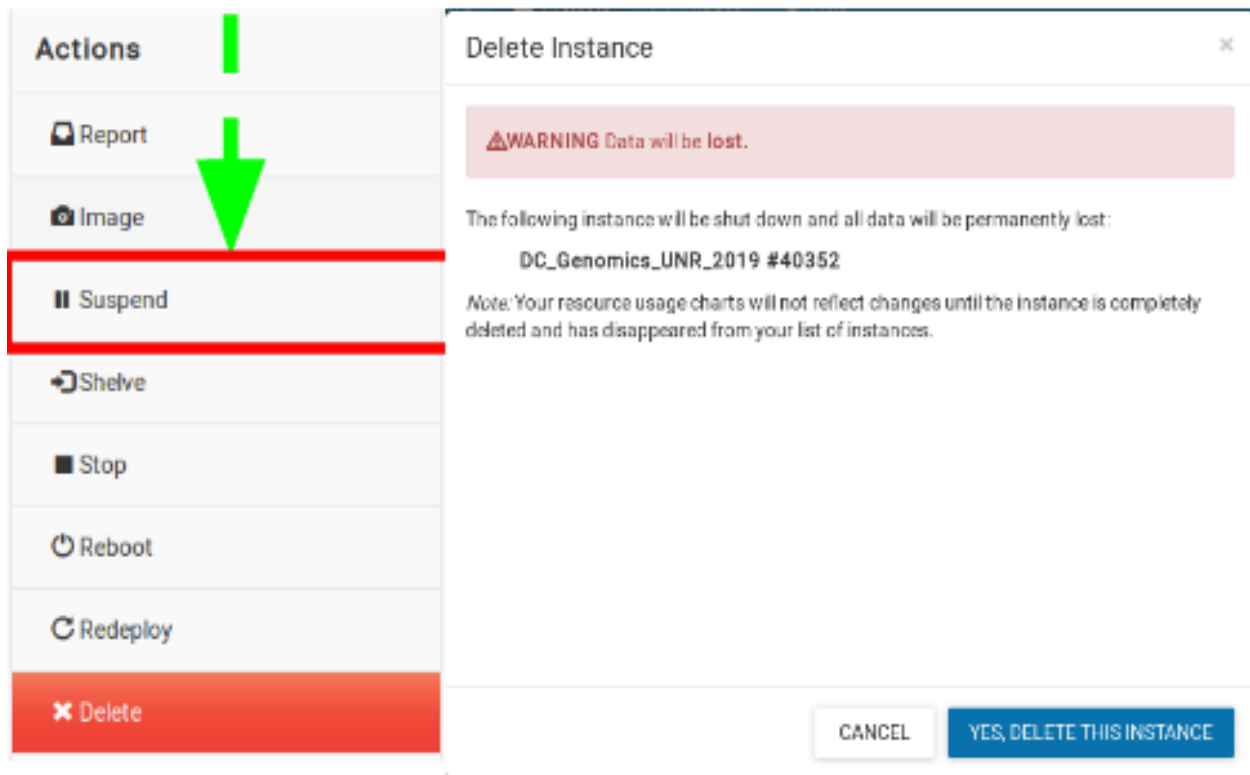
## Instance Maintenance

To end your current session on an Instance and close SSH connection, type 'exit'

### 27.1 Atmosphere Dashboard



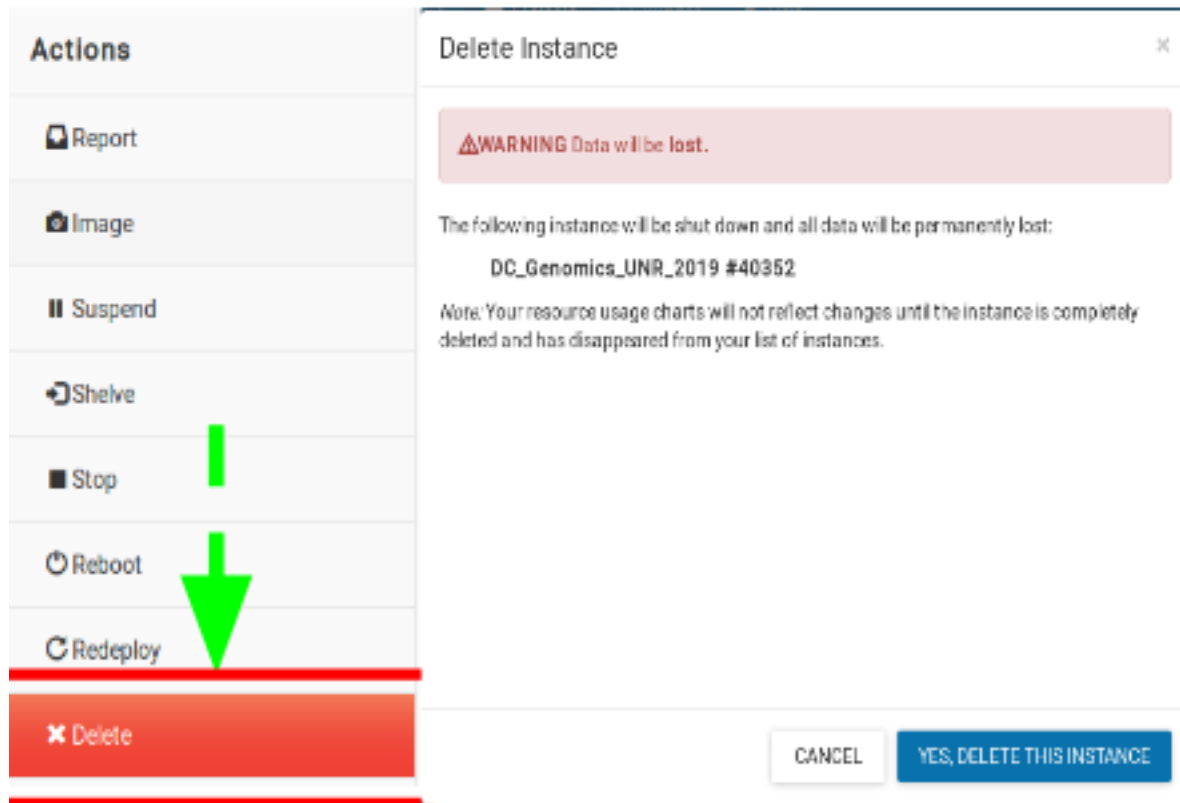
## 27.2 Suspend Instance



**Note:** It is advisable to delete the machine if you are not planning to use it in future to save valuable resources. However if you want to use it in future, you can suspend it. Notice: IP address changes

## 27.3 Delete Instance

- To completely remove your instance, you can select the “Delete” button from the instance details page.
- This will open up a dialogue window. Select the “Yes, delete this instance” button.
- It may take Atmosphere a few minutes to process your request. The instance should disappear from the project when it has been successfully deleted.

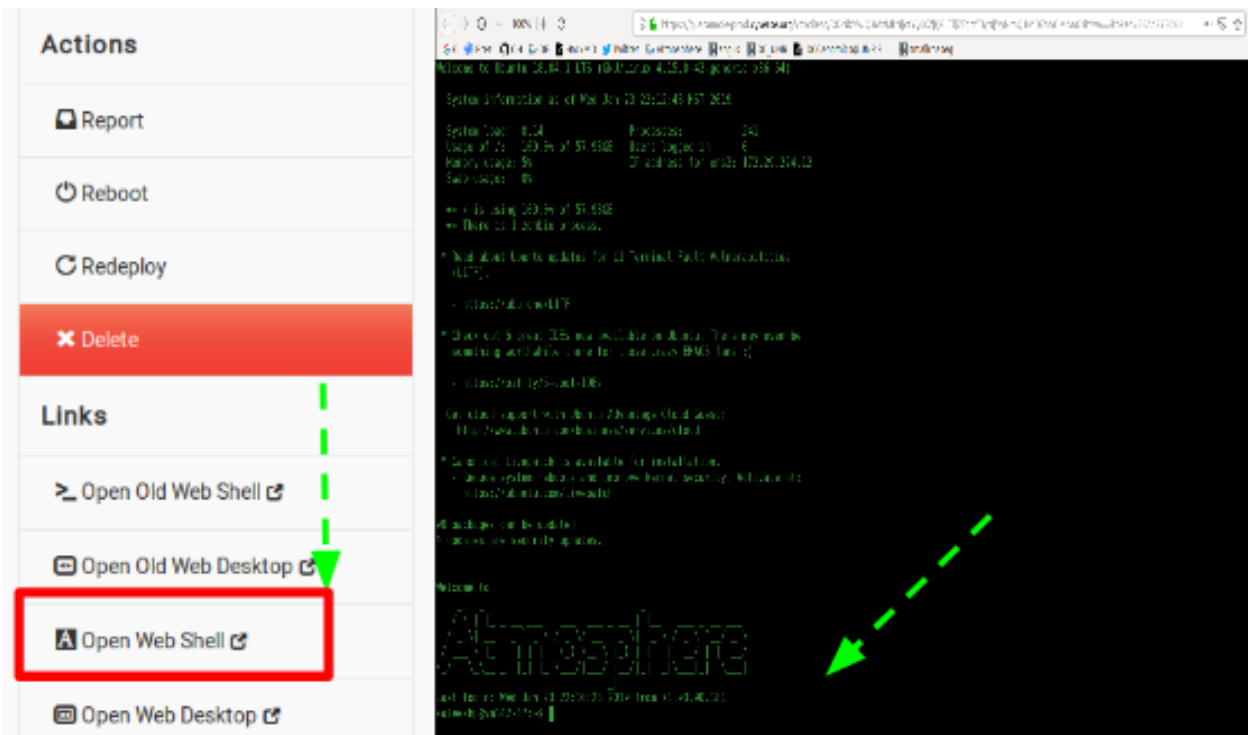




## 28.1 Did you know you can access a shell terminal and a web-desktop via your browser ???

### 28.1.1 Web Shell

- Access Command-Line Directly from your browser



## 28.1.2 Web Desktop

- Access your instance through a web desktop version from your browser





#### 29.1 Atmosphere Advanced Topics

- How do I transfer Data to and from my remote instances on Atmosphere?
- How can I create custom Atmosphere Images for my personal use?
- How can I add 'ssh-rsa-key' to Atmosphere for password-less login?



## CHAPTER 30

---

### Advanced Topics

---



---

## Transferring Data to and from an Instance

---

An Atmosphere instance only contains data that were included in the original image that was used to make that instance. You can transfer data to/from the CyVerse Data Store, an Atmosphere Volume, a sever, or your local machine.

### 31.1 Transferring Data using iCommands

- **iCommands Guide:**

- **iCommands Download:**

iCommands is installed on every Atmosphere instance by default

- Open terminal, start the iCommands configuration using the command `iinit` command and enter configure.

Note: This configuration is a one-time step on your first use with this instance.

```
$ iinit
# As prompted, enter the following values:
$ **Host: data.cyverse.org**
$ **Port: 1247**
$ **User: your_cyverse_username**
$ **Zone: iplant**
$ **Password: your_cyverse_password**
```

If you make a mistake in your configuration you can edit `~/.irods/irods_environment.json` on your instance.

- Verify that your iCommands installation works and is properly configured using the ‘ils’ command to list the contents of your Data Store home directory

```
$ ils
```

- To download a file from the Data Store to your instance, use “iget”

```
$ iget -option data_store_file
```

‘-r’ - recursive transfer of directories and their contents

‘-P’ - display the progress of the upload

‘-f’ - force the upload and overwrite

- To upload file from your instance to the Data Store use “iput”

```
$ iput -option file_on_instance location_on_data_store
```

- Additional Frequently used iCommands

```
$ ipwd print current directory
$ imkdir create directory
$ icd # change directory
```

iCommands has a variety of options, to see progress of transfers, operate recursively, and more. See additional [iCommands documentation](#) on the CyVerse wiki.

---

### Creating custom Atmosphere Images

---

- [Snakemake2019 v1.0 Image](#)
- [Jetstream](#)
- [Ubuntu 18.04 NO-GUI Atmosphere Image](#)

```
cd /opt

sudo su

ezd

ezs

apt-get install debootstrap

curl -O -L https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh #install in /opt/miniconda3

echo export PATH=$PATH:/opt/miniconda3/bin >> ~/.bashrc
source ~/.bashrc

conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda

conda install -y fastqc multiqc trimmomatic trinity time salmon

apt-get install tree sl

apt-get update && sudo apt-get -y install gdebi-core r-base

wget https://download2.rstudio.org/rstudio-server-1.1.463-amd64.deb
gdebi -n rstudio-server-1.1.463-amd64.deb
```

(continues on next page)

(continued from previous page)

```
RSTUDIO_LATEST=$(wget --no-check-certificate -qO- https://s3.amazonaws.com/rstudio-  
↪server/current.ver)  
  
`echo http://$(hostname):8787/`
```



---

### ssh-rsa-key for password-less login

---

- Cryptographic keys are a convenient and secure way to authenticate without having to use passwords. They consist of a pair of files called the public and private keys: the public part can be shared with whoever you'd like to authenticate with (in our case, CyVerse Atmosphere!), and the private part is kept "secret" on your machine. Things that are encrypted with the public key can be decrypted with the private key, but it is computationally intractable (ie, it would take on the order of thousands of years) to determine a private key from a public key. You can read more about it [here](#)
- Create the RSA Key Pair on your laptop/computer:

```
$ ssh-keygen -t rsa
```

- Store the Keys and Passphrase. Once you have entered the Gen Key command, you will get a few more questions:

```
$ Enter file in which to save the key (/home/demo/.ssh/id_rsa):
```

- You can press enter here, saving the file to the user home (in this case, my example user is called demo).

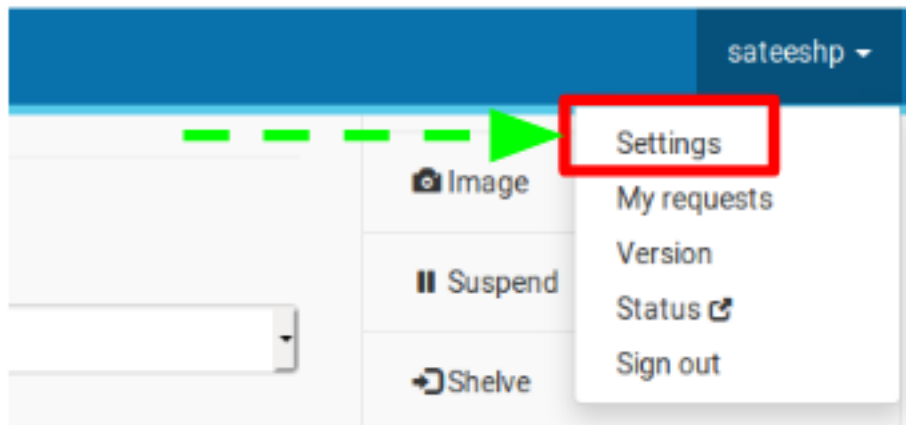
```
$ Enter passphrase (empty for no passphrase):
```

- It's up to you whether you want to use a passphrase though we recommend NOT to!.
- The public key is now located in /home/demo/.ssh/id\_rsa.pub. The private key (identification) is now located in /home/demo/.ssh/id\_rsa
- Copy the Public Key
- Once the key pair is generated, it's time to place the public key on the server that we want to use.

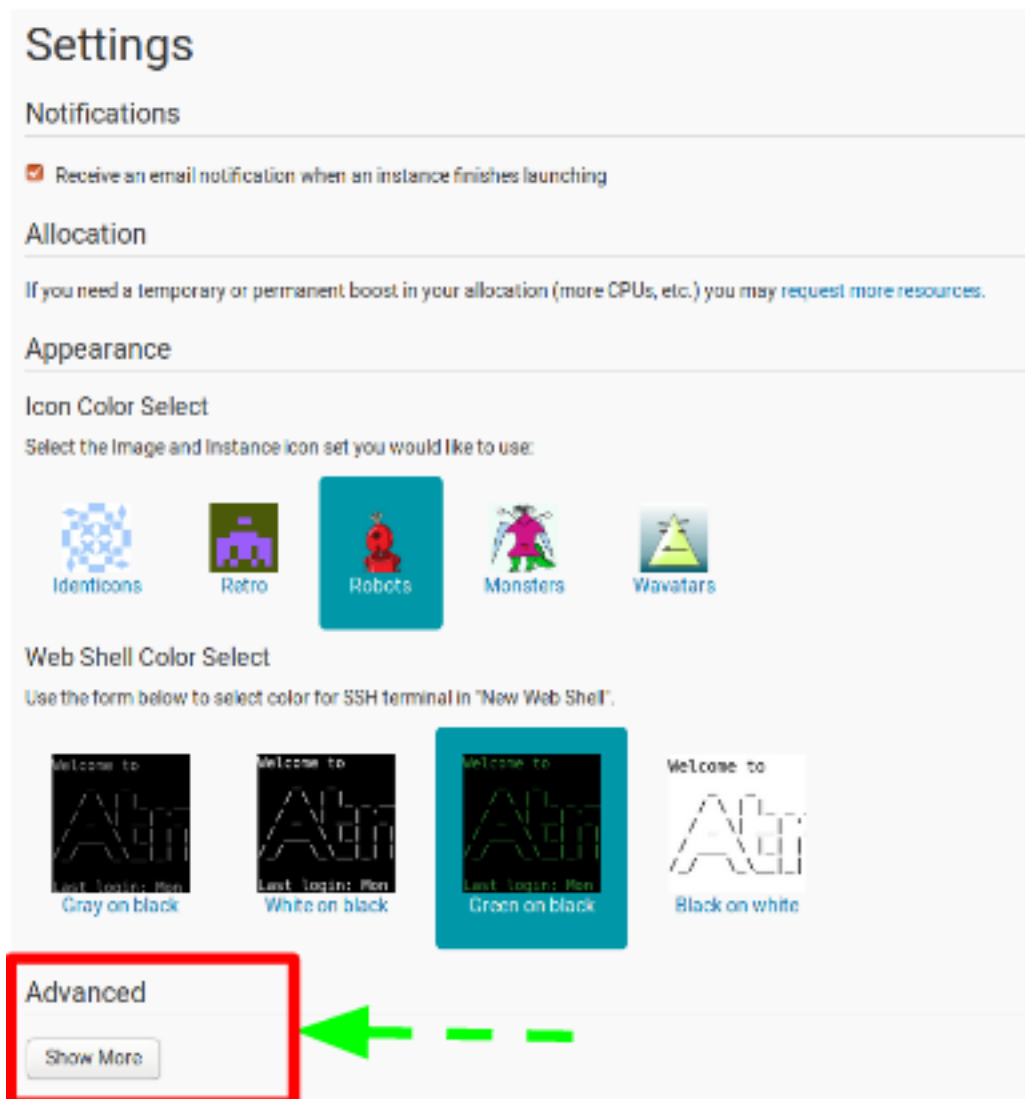
```
$ cat ~/.ssh/id_rsa.pub
```

- Copy all of the text that constitutes your public key and we will place it in our Atmosphere accounts.
- You are now ready to deposit your generated public key onto your Atmosphere account, to gain secure login each time you build and use an instance on Atmosphere.

- Click on your username on the Atmosphere page and navigate to ‘Settings’ page



- Scroll down to the advanced section and click on ‘Show More’



- In the ‘SSH Configuration’ section, click on the ‘+’ sign

Advanced

### Deployment Scripts (formerly Boot Scripts)

Use the table below to create and/or edit existing deployment scripts. These scripts can be selected when you launch an instance.

[Click here to learn more.](#)

Script Title	Execution Strategy Type	Deployment Type	Input Type	Actions
+				

### SSH Configuration

Use the table below to list SSH keys that you would like to be present when you launch an instance.

[Click here to learn more.](#)

Name	Public Key	Actions
+		

### Personal Access Tokens

Personal Access Tokens are API Tokens that can be used instead of passwords for authentication. Other applications can use Personal Access Tokens to access Atmosphere services under your account.

Name	Issued	Actions
+		

[Show Less](#)

- Paste your public key generated earlier and give this key a name

Add a public SSH key

Key Name

sateesh\_laptop

Public Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCb5A5YtIbojGzERq8oBg2H3+yC/BOJ0/9rde
4BCCtovjkd9Q
/LQZn4M8e4CqxX77+Z0G4f0VDaT1+MSV4ZRrtANkPftxcBCIKsFG2ABXgfNlbbOv6UD
Ux6sw
```

Cancel Confirm

- You can now securely login to all the instances you launch on Atmosphere without having to type your password each-time you ssh login.

```

sateeshp@sp:~$ ssh sateeshp@128.196.142.22
The authenticity of host '128.196.142.22 (128.196.142.22)' can't be
established.
ECDSA key fingerprint is SHA256:wSs8Q7JTjmHFE7atvp2xxWQeTNn8oMgSmAsY
iHWHgds.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '128.196.142.22' (ECDSA) to the list of k
nown hosts.
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-46-generic x86_64)

   System information as of Sun Mar 17 15:07:31 MST 2019

   System load:  0.07               Processes:            172
   Usage of /:   46.3% of 57.98GB   Users logged in:     0
   Memory usage: 28%               IP address for ens3: 172.29.224.8
   Swap usage:   0%

Welcome to
_Altoughspine_
Last login: Sat Mar 16 13:06:48 2019 from 71.93.90.181
sateeshp@vm142-29:~$

```

## CHAPTER 34

---

### Tmux & Screen

---

Have you ever faced the situation where you perform a long-running task on a remote machine and suddenly your connection drops, the SSH session is terminated and your work is lost. Well it has happened to all of us at some point, hasn't it? Luckily, there are utilities like [Tmux](#) & [Screen](#) that allow us to the resume our sessions.



---

### SSH Remote host ID Changed Error

---

- When we suspend an instance and re-start it, the IP address might change. But, when two instances get the same address this login-error might pop-up

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:6vrlEYqlC1DJnWEoE9k2gfbaXUhsEM2iF9jKuhfNs2o.
Please contact your system administrator.
Add correct host key in /home/sateeshp/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/sateeshp/.ssh/known_hosts:9
  remove with:
    ssh-keygen -f "/home/sateeshp/.ssh/known_hosts" -R "128.196.142.26"
ECDSA host key for 128.196.142.26 has changed and you have requested strict checking.
Host key verification failed.
```

- Executing the `ssh-keygen -f` command as above will delete the conflicting key the `known_hosts` file.





---

## Workflow Management using Snakemake

---

### 36.1 Learning Objectives

- Identify cases where workflow managers are helpful for automation
- Understand the components of a Snakefile: rules, inputs, outputs, and actions.
- Write and run a Snakefile
- Learn about using isolated conda environments in rules
- Learn to use docker/singularity containers in Snakemake rules

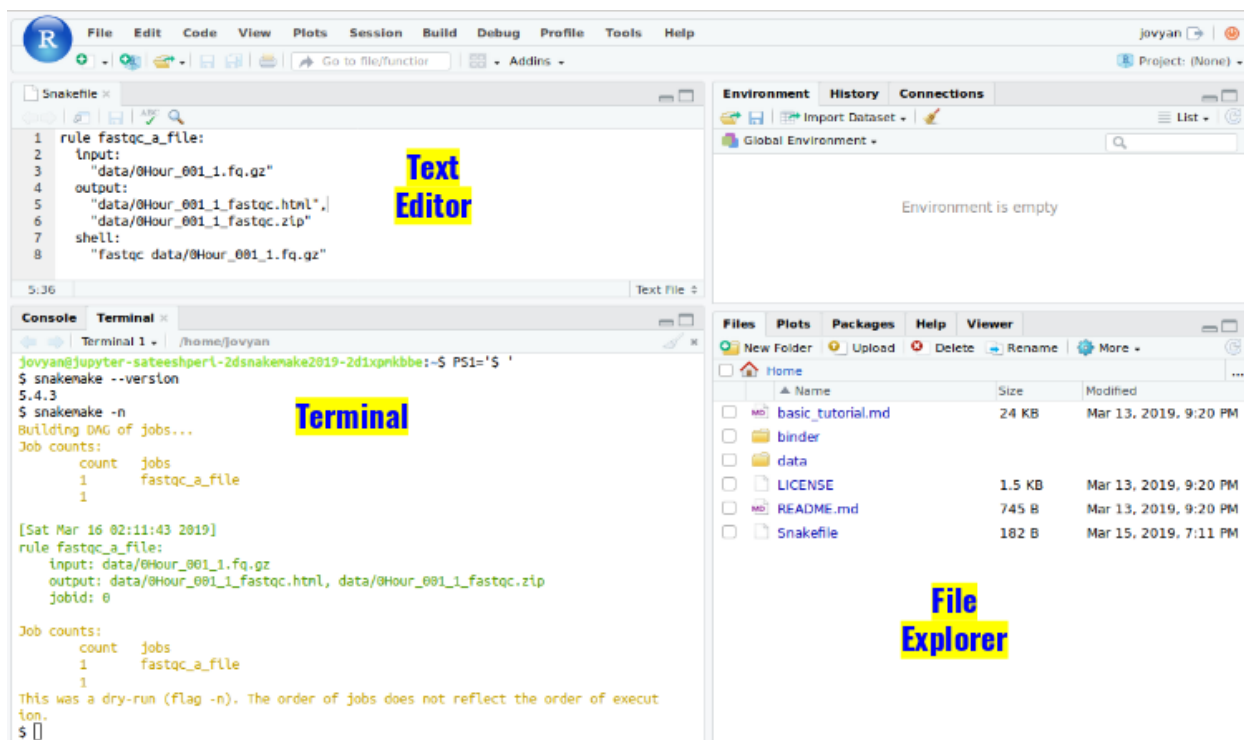
See here for CyVerse [Guide to Launching Atmosphere Instances](#)

### 36.2 Setup

1. Login to CyVerse [Atmosphere](#) and launch a medium ‘m1’ instance with the ‘Snakemake2019’ v1.0 base image
2. Open Rstudio
  - Type the following in your terminal to display a link to Rstudio web-server for your instance’s \$(hostname)

```
echo http://$(hostname -i):8787/
```

- click on the link generated to open Rstudio in your browser and login with your CyVerse credentials.



### 1. Activate Conda

```
echo export PATH=$PATH:/opt/miniconda3/bin >> ~/.bashrc
```

### 1. Then, run the following command (or start a new terminal session) in order to activate the conda environment:

```
source ~/.bashrc
```

### 1. Try running the following UNIX command ‘which’, which returns the pathnames of the files (or links) which would be executed in the current environment:

```
which snakemake
```

it should show the absolute path of snakemake as ‘/opt/miniconda3/bin/snakemake’

### 1. Check if singularity is available in your \$PATH and print version:

```
which singularity
```

It should show the absolute path of singularity ‘/usr/local/bin/singularity’

- We will be executing the same workflow **fastqc**→**multitqc**→**trimmomatic** as in **Basic Tutorial** but, with tools being executed in singularity containers based on either Docker or Singularity builds

### 1. Download data

```
mkdir data
cd data/
curl -L https://osf.io/5daup/download -o ERR458493.fastq.gz
curl -L https://osf.io/8rvh5/download -o ERR458494.fastq.gz
curl -L https://osf.io/2wvn3/download -o ERR458495.fastq.gz
curl -L https://osf.io/xju4a/download -o ERR458500.fastq.gz
```

(continues on next page)

(continued from previous page)

```
curl -L https://osf.io/nmqe6/download -o ERR458501.fastq.gz
curl -L https://osf.io/qfsze/download -o ERR458502.fastq.gz
```

### 36.3 Introduction to Snakemake

The Snakemake workflow management system is a tool to create reproducible and scalable data analyses. It orchestrates and keeps track of all the different steps of workflows that have been run so you don't have to! It has a lot of wonderful features that can be invoked for different applications, making it very flexible while maintaining human interpretability.

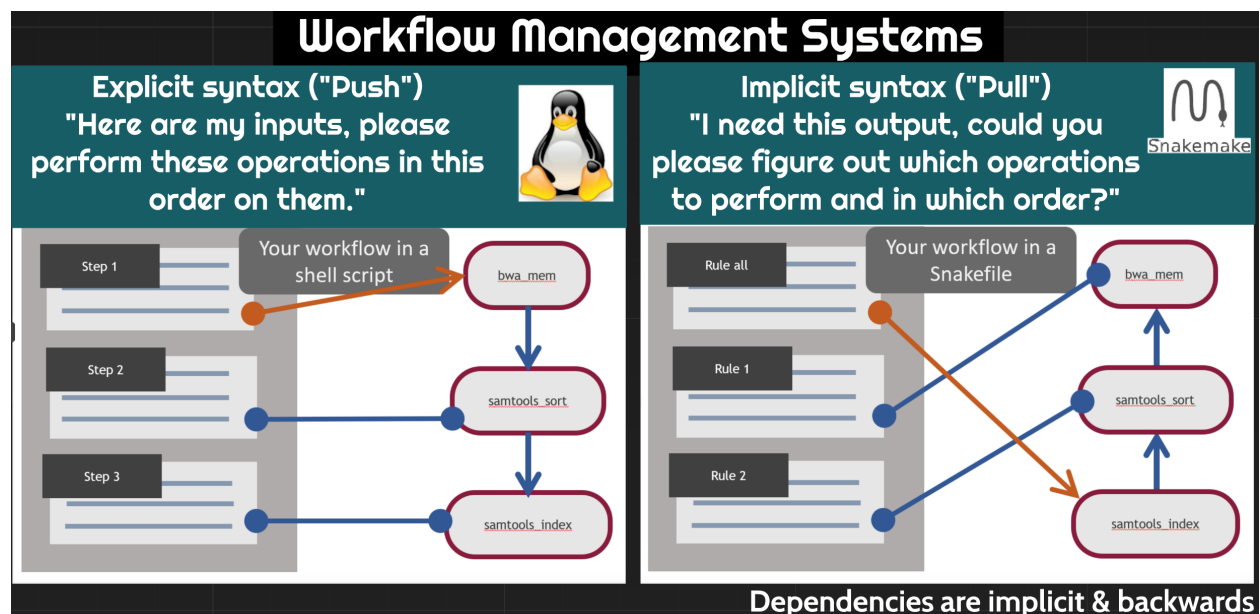
There are many different tools that researchers use to automate computational workflows. We selected snakemake for the following reasons:

- It's free, open-source, and conda-installable
- Snakemake works cross-platform (Windows, MacOS, Linux) and is compatible with all HPC schedulers. It works on laptops, the cloud, and clusters without modification to the main workflow (as long as you have enough compute resources!).
- Snakemake is written using Python, but supports bash and R code as well.
- Anything that you can do in Python, you can do with Snakemake (since you can pretty much execute arbitrary Python code anywhere).

Like other workflow management systems, Snakemake allows you to:

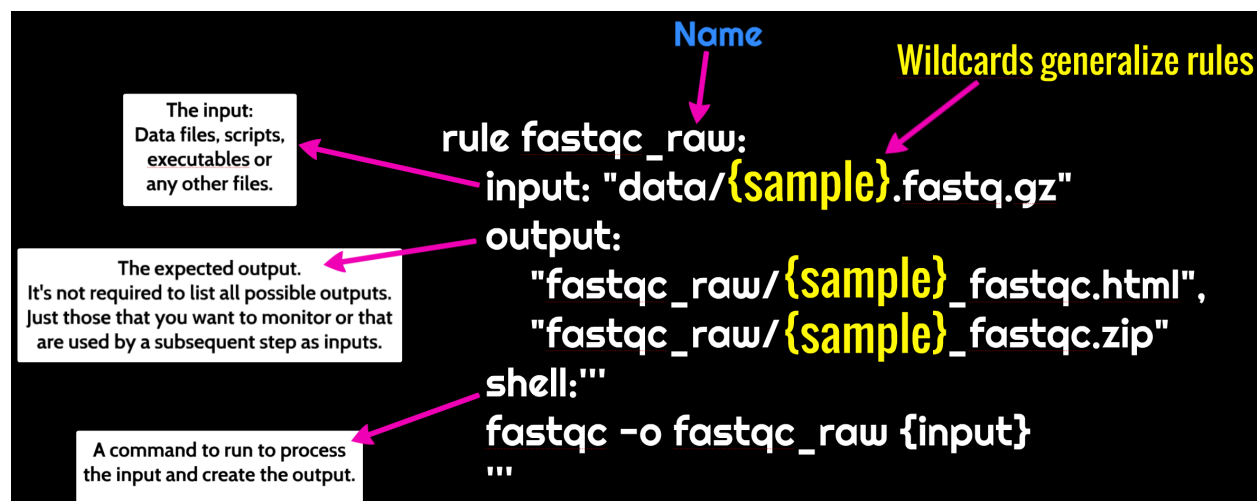
- Keep a record of how your scripts are used and what their input dependencies are
- Run multiple steps in sequence, parallelizing where possible
- Automatically detect if something changes and then reprocess data if needed

Our goal is to automate the first two steps (FastQC MultiQC) of our example workflow using snakemake!



## 36.4 Starting with Snakemake

Snakemake workflows are built around **rules**. The diagram below shows the anatomy of a snakemake rule:



Let's make a rule to run `fastqc` on one of our samples below. We'll put this rule in a file called `Snakefile`.

```
# This rule will run fastqc on the specified input file.
rule fastqc_raw:
    input: "data/ERR458493.fastq.gz"
    output:
        "fastqc_raw/ERR458493_fastqc.html",
        "fastqc_raw/ERR458493_fastqc.zip"
    shell:'''
    fastqc -o fastqc_raw {input}
    '''
```

Let's try and run our `Snakefile`! Return to the command line and run `snakemake`.

```
snakemake
```

You should see output that starts like this:

```
Building DAG of jobs...
Using shell: /bin/bash
Provided cores: 1
Rules claiming more threads will be scaled down.
Job counts:
   count      jobs
    1      fastqc_raw
    1

[Tue Jul  2 19:10:26 2019]
rule fastqc_raw:
  input: data/ERR458493.fastq.gz
  output: fastqc_raw/ERR458493_fastqc.html, fastqc_raw/ERR458493_fastqc.zip
  jobid: 0
```

Let's check that the output file is there:

```
ls fastqc_raw/*fastqc*
```

Yay! Snakemake ran the thing!

We can also use better organization. Let's **specify a different output folder** for our fastqc results

```
# This rule will run fastqc on the specified input file
# (replace the prior fastqc_raw rule with this new rule)
rule fastqc_raw:
    input: "data/ERR458493.fastq.gz"
    output:
        "fastqc_raw/ERR458493_fastqc.html",
        "fastqc_raw/ERR458493_fastqc.zip"
    shell:'''
    fastqc -o fastqc_raw {input}
    '''
```

If we look in our directory, we should now see a fastqc\_raw directory, even though we didn't create it:

```
ls
```

Snakemake created this directory for us. We can look inside it to see if it really ran our command:

```
ls fastqc_raw
```

## 36.5 Creating a pipeline with snakemake

We told snakemake to do something, and it did it. Let's add another rule to our Snakefile telling snakemake to do something else. This time, we'll run multiqc.

```
# Run fastqc on the specified input file
rule fastqc_raw:
    input: "data/ERR458493.fastq.gz"
    output:
        "fastqc_raw/ERR458493_fastqc.html",
        "fastqc_raw/ERR458493_fastqc.zip"
    shell:'''
    fastqc -o fastqc_raw {input}
    '''

# Run multiqc on the results of the fastqc_raw rule
rule multiqc_raw:
    input: "fastqc_raw/ERR458493_fastqc.zip"
    output: "fastqc_raw/multiqc_report.html"
    shell:'''
    multiqc -o fastqc_raw fastqc_raw
    '''
```

We see output like this:

```
Building DAG of jobs...
Nothing to be done.
Complete log: /Users/tr/2019_angus/.snakemake/log/2019-07-02T191640.002744.snakemake.
↪ log
```

However, when we look at the output directory `fastqc_raw`, we see that our `multiqc` file does not exist! Bad Snakemake! Bad!!

Snakemake looks for a rule `all` in a file as the final file it needs to produce in a workflow. Once this file is defined, it will go back through all other rules looking for which ordered sequence of rules will produce all of the files necessary to get the final file(s) specified in rule `all`. For this point in our workflow, this is our `fastqc` sample directory.. Let's add a rule `all`.

```
rule all:
    input:
        "fastqc_raw/multiqc_report.html"

rule fastqc_raw:
    input: "data/ERR458493.fastq.gz"
    output:
        "fastqc_raw/ERR458493_fastqc.html",
        "fastqc_raw/ERR458493_fastqc.zip"
    shell:'''
    fastqc -o fastqc_raw {input}
    '''

rule multiqc_raw:
    input: "fastqc_raw/ERR458493_fastqc.html"
    output: "fastqc_raw/multiqc_report.html"
    shell:'''
    multiqc -o fastqc_raw fastqc_raw
    '''
```

And it worked! Now we see output like this:

```
Building DAG of jobs...
Using shell: /bin/bash
Provided cores: 1
Rules claiming more threads will be scaled down.
Job counts:
   count      jobs
   1      all
   1      multiqc_raw
   2
```

Snakemake now has two processes it's keeping track of.



## 36.6 Using Snakemake to process multiple files

So far we've been using snakemake to process one sample. However, we have 6! Snakemake can be flexibly extended to more samples using wildcards.

We already saw wildcards previously.

When we specified the output file path with `{input}`, `{input}` was a wildcard. The wildcard is equivalent to the value we specified for `{input}`.

```

rule fastqc_raw:
    input: "data/ERR458493.fastq.gz"
    output:
        "fastqc_raw/ERR458493_fastqc.html",
        "fastqc_raw/ERR458493_fastqc.zip"
    shell:'''
    fastqc -o fastqc_raw {input}
    '''
  
```

We can create our own wildcard too. This is really handy for running our workflow on all of our samples.

```

# Create a list of strings containing all of our sample names
SAMPLES=['ERR458493', 'ERR458494', 'ERR458495', 'ERR458500', 'ERR458501',
'ERR458502']

rule all:
    input:
        "fastqc_raw/multiqc_report.html"

rule fastqc_raw:
    input: "data/{sample}.fastq.gz"
  
```

(continues on next page)

(continued from previous page)

```

output:
    "fastqc_raw/{sample}_fastqc.html",
    "fastqc_raw/{sample}_fastqc.zip"
shell:'''
fastqc -o fastqc_raw {input}
'''

rule multiqc_raw:
    input: expand("fastqc_raw/{sample}_fastqc.html", sample = SAMPLES)
    output: "fastqc_raw/multiqc_report.html"
    shell:'''
multiqc -o fastqc_raw fastqc_raw
'''

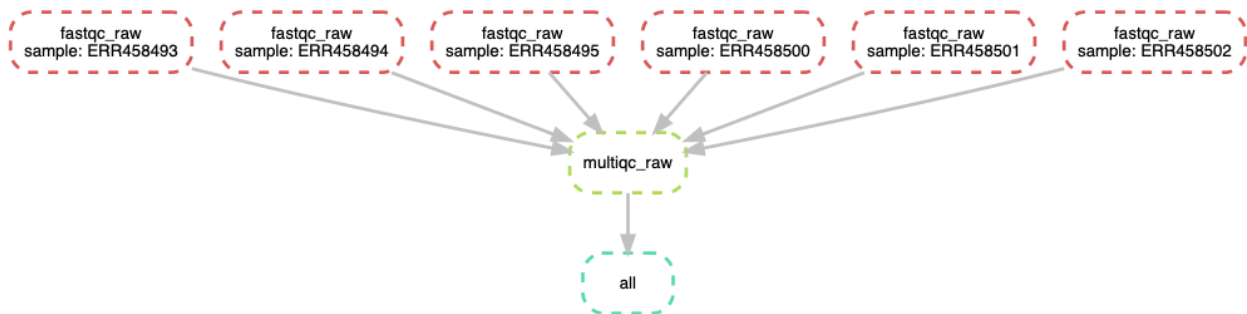
```

We can run this again at the terminal.

```
snakemake
```

And we have now run these rules for each of our samples!

Note that we added new syntax here as well. We define a variable at the top of the snakefile call `SAMPLES`. Snakemake solves the values for the wildcard `{sample}` the last time that see that wildcard. However, we need to `expand` the wildcard using the `expand` function, and tell snakemake in which variable to look for the values.



### 36.6.1 Helpful guidelines

- Indentation is important, use two or four spaces for each indentation.
- Define your target (final output) files in rule `all`
- Use unique unique extensions or directories for each rule to avoid wildcard collisions

## 36.7 Snakemake Additional Features

### 36.7.1 dry-run

```
snakemake -n
```



### 36.7.2 print shell commands

```
snakemake -p
```

### 36.7.3 print reason for execution

```
snakemake -r
```

### 36.7.4 execute the workflow with n cores

```
snakemake --cores n
```

### 36.7.5 run the workflow on a SLURM cluster

```
snakemake --cluster-config cluster.yml --cluster \
    "sbatch -A {cluster.account} -t {cluster.time}"
```

### 36.7.6 Visualize entire workflow diagram

```
snakemake --dag | dot -Tpng > dag.png
```

The DAG png file should look something as shown above.

### 36.7.7 Snakemake Report

Snakemake can automatically generate detailed self-contained HTML reports that encompass runtime statistics, provenance information, workflow topology and results.

To create the report, run

```
snakemake --report report.html
```

### 36.7.8 Using Singularity/Docker containers in Snakemake

- It is possible to define, for each rule, a docker or singularity container to use, e.g.,

```
# Create a list of strings containing all of our sample names
SAMPLES=['ERR458493', 'ERR458494', 'ERR458495', 'ERR458500', 'ERR458501',
'ERR458502']

rule all:
    input:
        "fastqc_raw/multiqc_report.html"

rule fastqc_raw:
    input: "data/{sample}.fastq.gz"
```

(continues on next page)

(continued from previous page)

```

output:
    "fastqc_raw/{sample}_fastqc.html",
    "fastqc_raw/{sample}_fastqc.zip"
singularity:
    "docker://sateeshperi/fastqc"
shell:'''
fastqc -o fastqc_raw {input}
'''

rule multiqc_raw:
    input: expand("fastqc_raw/{sample}_fastqc.html", sample = SAMPLES)
    output: "fastqc_raw/multiqc_report.html"
    singularity:
        "docker://sateeshperi/multiqc"
    shell:'''
multiqc -o fastqc_raw fastqc_raw
'''

```

Save the file as Snakefile and execute Snakemake in your terminal by:

```
snakemake --use-singularity
```

it will execute the job within a singularity container that is spawned from the given image. Allowed image urls entail everything supported by singularity (e.g., shub:// and docker://).

- Dockerfiles used in this workflow
  - fastqc
  - multiqc
  - trimmomatic

### 36.7.9 Specifying software required for a rule

**You can specify software on a per-rule basis! This is really helpful when you have incompatible software requirements for different rules, or want to run on a cluster, or want to make your workflow reproducible.**

For example, if you create a file `env_fastqc.yml` with the following content:

```

channels:
    - conda-forge
    - bioconda
    - defaults
dependencies:
    - fastqc==0.11.8

```

and then change the fastqc rule to look like this:

```

rule fastqc_raw:
    input: "data/{sample}.fastq.gz"
    output:
        "fastqc_raw/{sample}_fastqc.html",
        "fastqc_raw/{sample}_fastqc.zip"
    conda:
        "env_fastqc.yml"
    shell:'''

```

(continues on next page)

(continued from previous page)

```
fastqc -o fastqc_raw {input}
'''
```

you can now run snakemake like so,

```
snakemake --use-conda
```

and for that rule, snakemake will install the specified software and dependencies in its own environment, with the specified version.

This aids in reproducibility, in addition to the practical advantages of isolating software installs from each other.

## 36.8 Advanced Snakemake

- You can access an **iRODS** server to retrieve data from and upload data to it. Read more [here](#)

## 36.9 Resources

- [Snakemake Documentation](#)
- Here are some great [Snakemake Workflows](#). Check out the RNAseq-STAR-DESeq2 workflow [here](#).
- [snakemake paper](#)
- [Snakemake Carpentry Lesson](#)

**Note:** It is advisable to delete your instance if you are not planning to use it in future to save valuable resources. However if you want to use it in future, you can suspend it. See [Instance Maintenance](#) for more info

---

**Snakemake2019 v1.0** [Atmosphere Image Specifications](#)



### Learning Objective

- **Container Management** using [Singularity](#)
- Make your project self-sustainable and distributable
- Understand what containers are
- Understand what Dockers are
- Understand why Singularity
- How to pull & run singularity containers
- [Link to Container Camp](#)
- [Amazing slides on Singularity](#) by *Vanessa Sochat*



## CHAPTER 38

---

### Awesome List of Resources

---





## CHAPTER 39

---

### Reproducibility Toolkit

---

- [Snakemake Carpentry Lesson](#)
- [NIH-NPC Singularity Tutorial](#)
- [Snakemake Wrapper Repository](#)
- [curated list of awesome pipeline toolkits](#)



---

## Local Setup instructions

---

### Learning Objective

- **Environment Management** using **conda & bioconda**
- Set up and manage the project environment
- Understand conda environments
- We recommend installing **conda**, an open source package management system and environment management system that runs on Windows, macOS and Linux.
- Download and install conda:

```
$ curl -O -L https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh  
$ bash Miniconda3-latest-Linux-x86_64.sh
```

Say “yes” to everything the installer asks, and accept default locations by pressing enter when it says “Miniconda3 will now be installed into this location”. (If the prompt looks like this “>>>”, then you are still within the installation process.)

- When the installation is complete and the regular prompt returns, run the following command (or start a new terminal session) in order to activate the conda environment:

```
$ source ~/.bashrc
```

- Next, enable various “channels” for software install, including bioconda:

```
$ conda config --add channels defaults  
$ conda config --add channels conda-forge  
$ conda config --add channels bioconda
```

- Next, download conda ‘environment.yml’ file, designed for this tutorial [here](#)
- Install tools in a new conda environment ‘smake’ using downloaded environment file

```
$ conda env create --file environment.yml -n smake
```

- To activate the environment for tutorial

```
$ conda activate smake
```

- To deactivate environment after use

```
$ conda deactivate
```

- Singularity installation needs *sudo* permissions and instructions can be found [here](#)