CyVerse Documentation

Release 0.1.0

CyVerse

Jun 05, 2018

CHAPTER 1

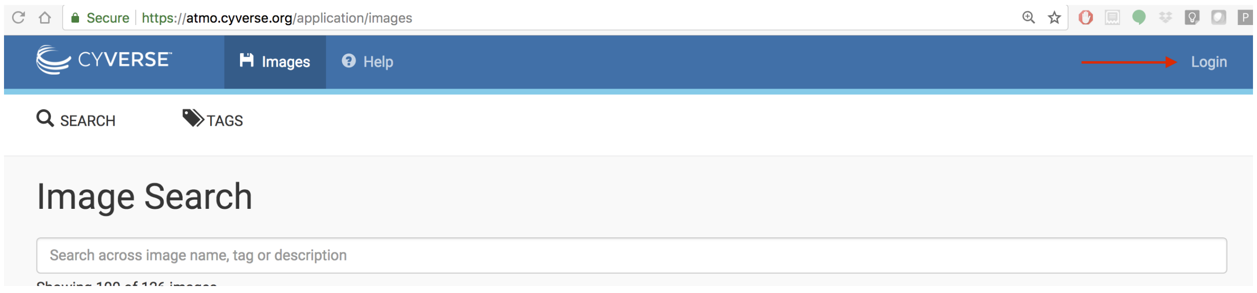
Logging onto Atmosphere Cloud

For the duration of this workshop, we will be accessing data and tools required to analyze our data on a remote computer using the **CyVerse's Atmosphere cloud**.

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, we will all be on the same operating system/computing environment.

Below, we've provided screenshots of the whole process. You can click on them to zoom in. The important areas to fill in are highlighted.

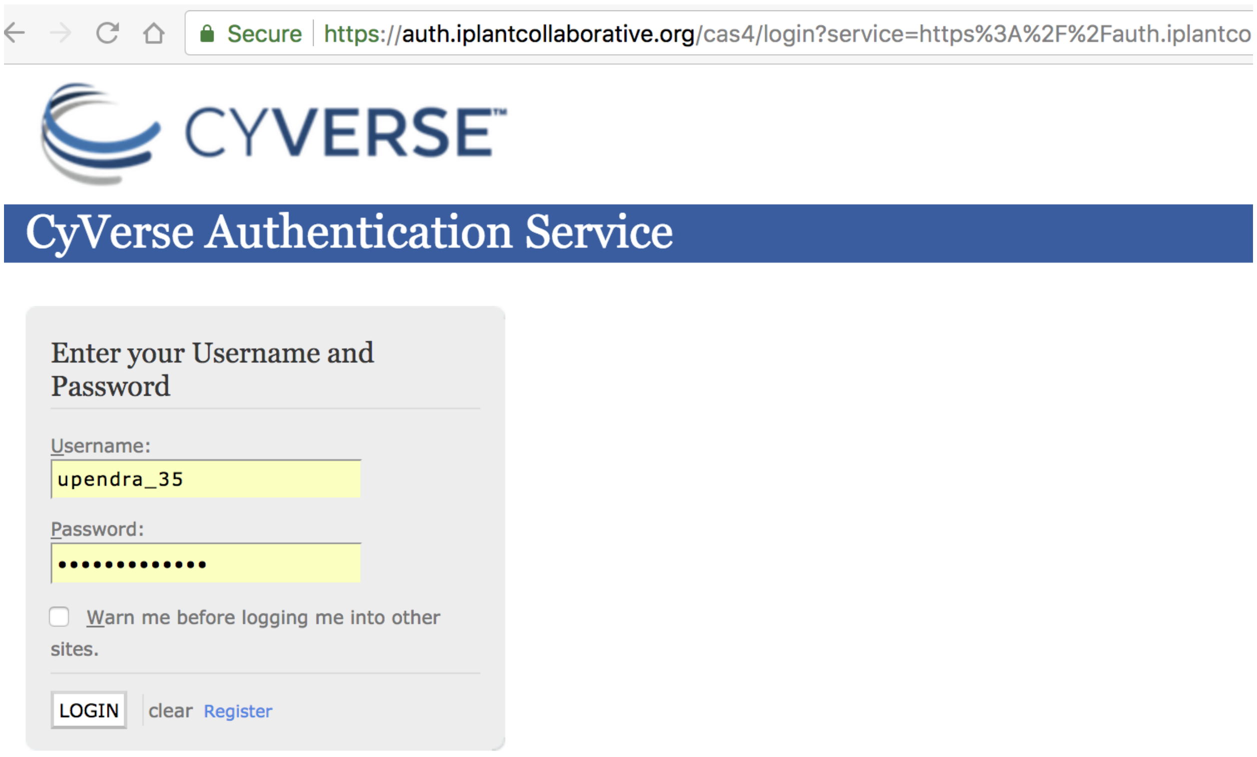
First, go to the Atmosphere application and then click *login*


Secure
https://atmo.cyverse.org/application/images
CYVERSE
Images
Help
Login
SEARCH
TAGS
Image Search
Search across image name, tag or description

Important: As descrbied in the pre-workshop setup, you need to have access to the CyVerse Atmosphere Cloud. If you are not able to log-in for some reason, please let us know and we will fix it immediately.

1. Fill in the username and password and click "LOGIN"

- Fill in the username, which is your CyVerse username, and then enter the password (which is your CyVerse password).


Secure https://auth.iplantcollaborative.org/cas4/login?service=https%3A%2F%2Fauth.iplantco

CyVerse Authentication Service

Enter your Username and Password

Username: upendra_35

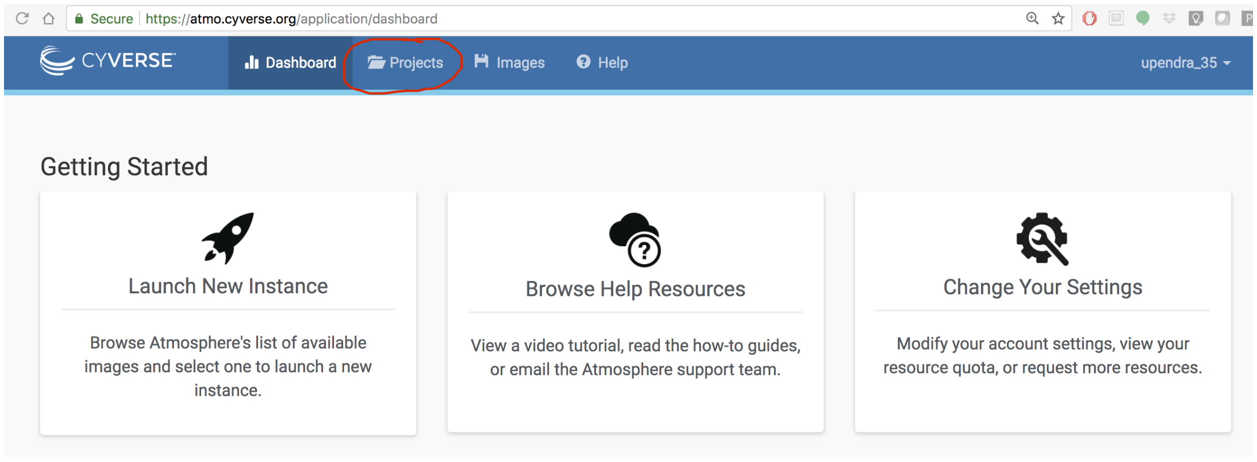
Password: •••••••••••••

☐ Warn me before logging me into other sites.

LOGIN clear Register

2. Select the "Projects" tab and then click the "CREATE NEW PROJECT" button

- This is something you only need to do once.
- A project is a workspace that lets you keep things together.
- Click on the "Projects" tab on the top and then click the "CREATE NEW PROJECT" button


Secure https://atmo.cyverse.org/application/dashboard

Getting Started

Launch New Instance

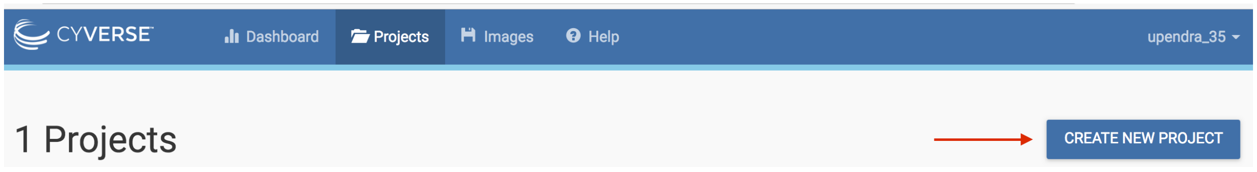
Browse Atmosphere's list of available images and select one to launch a new instance.

Browse Help Resources

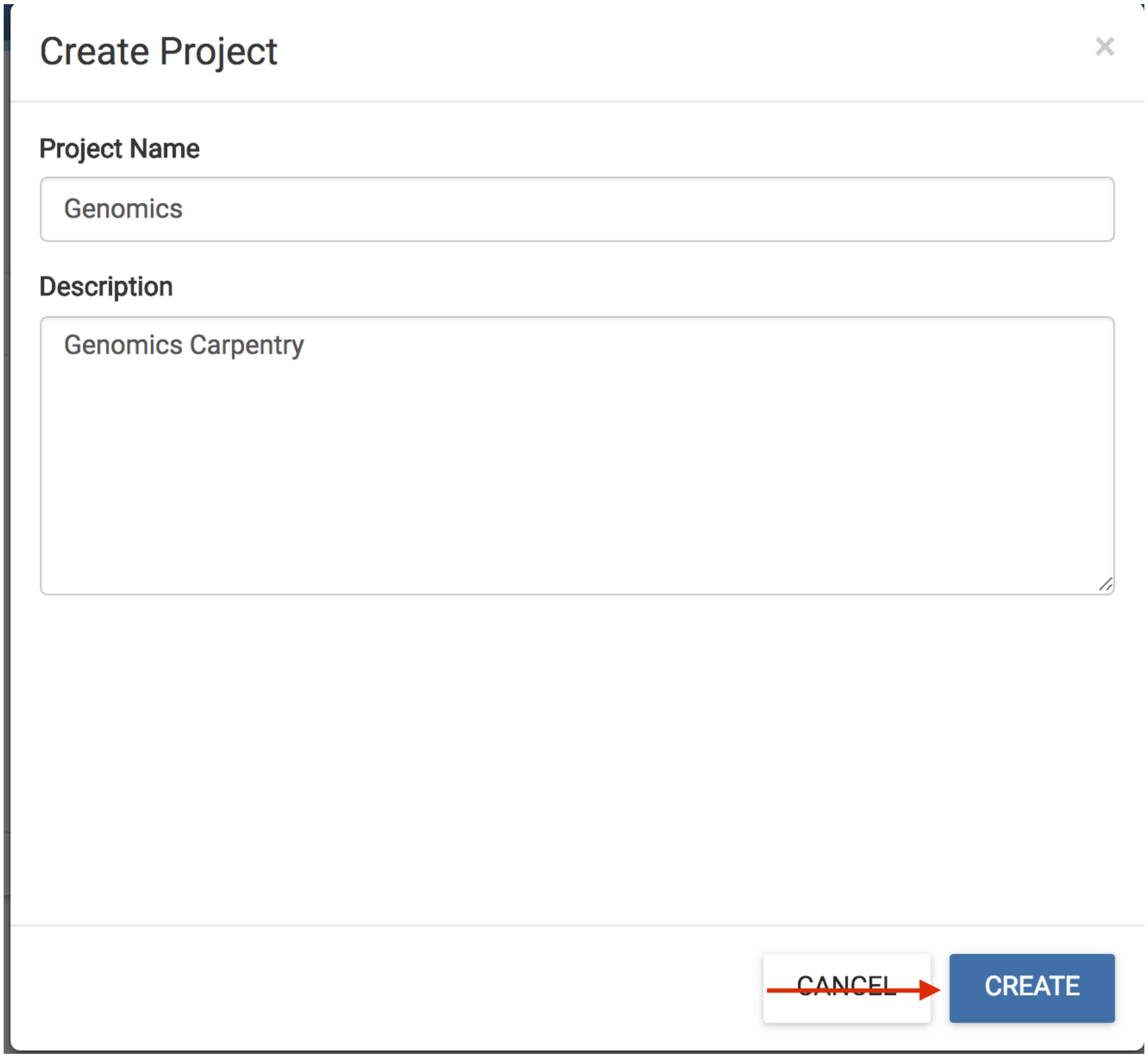
View a video tutorial, read the how-to guides, or email the Atmosphere support team.

Change Your Settings

Modify your account settings, view your resource quota, or request more resources.

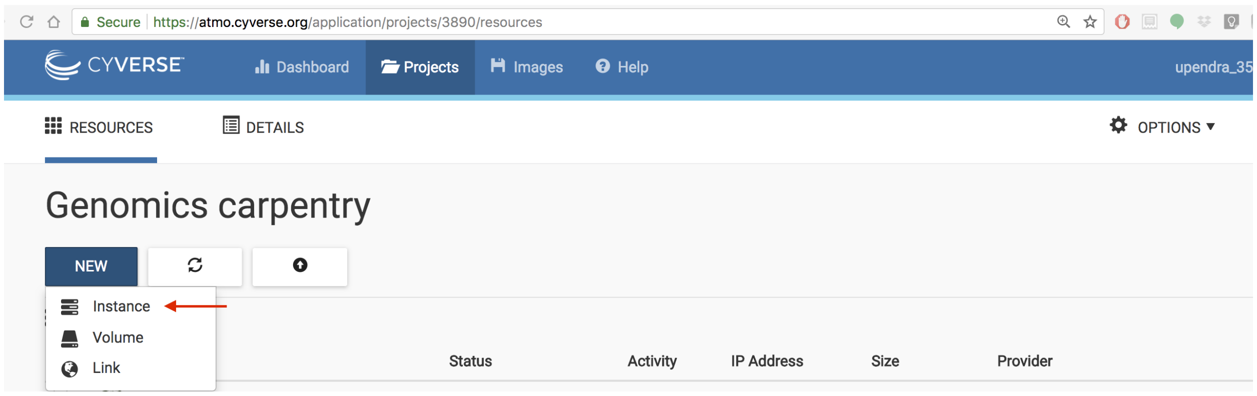


- Enter the name "Genomics" into the Project Name and put something simple like "Genomics" into the description. Then click "CREATE".

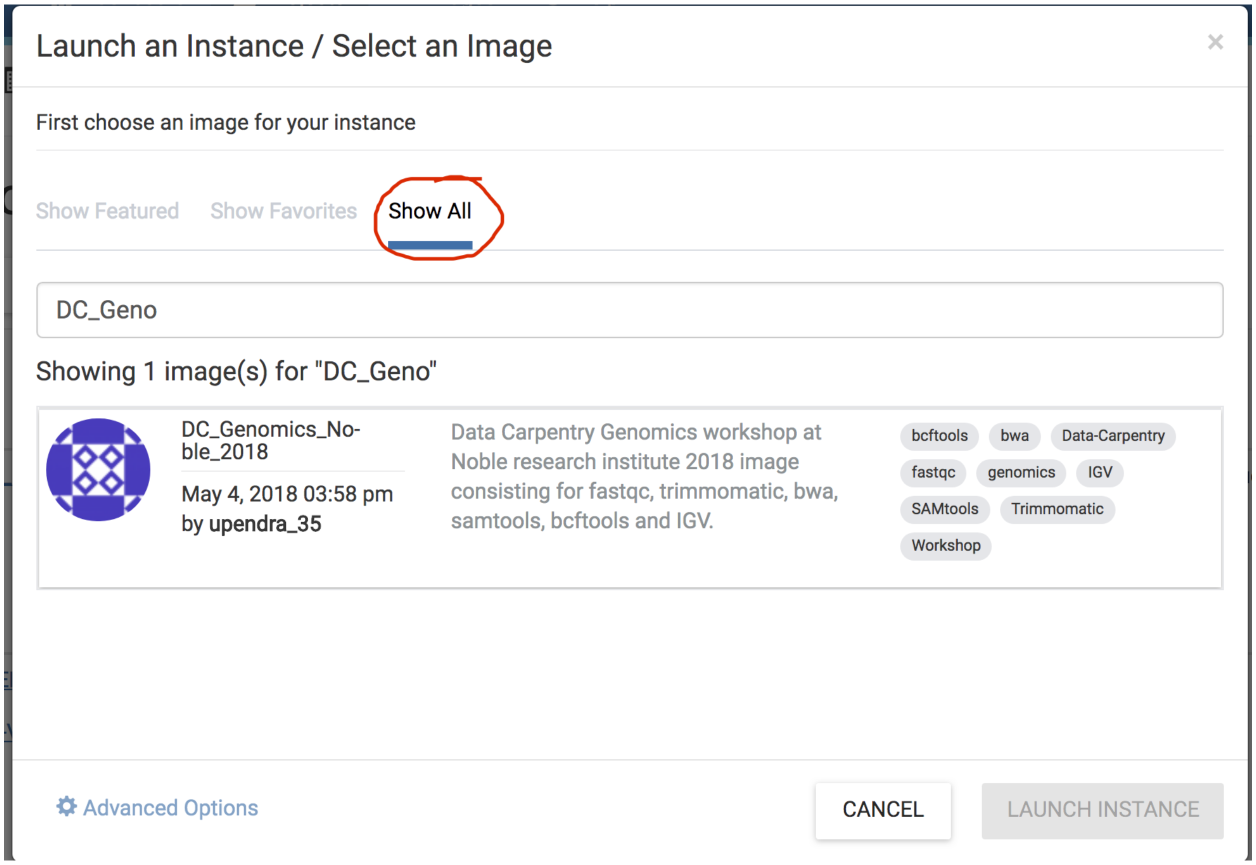


3. Select the newly created project

- Click on your newly created project.
- Click "NEW" and then "Instance" from the dropdown menu to start up a new virtual machine.



- Naviagate to "Show All tab and Search for "DC_Genomics"; click the "DC_Genomics_Noble_2018" image.



Launch an Instance / Select an Image

First choose an image for your instance

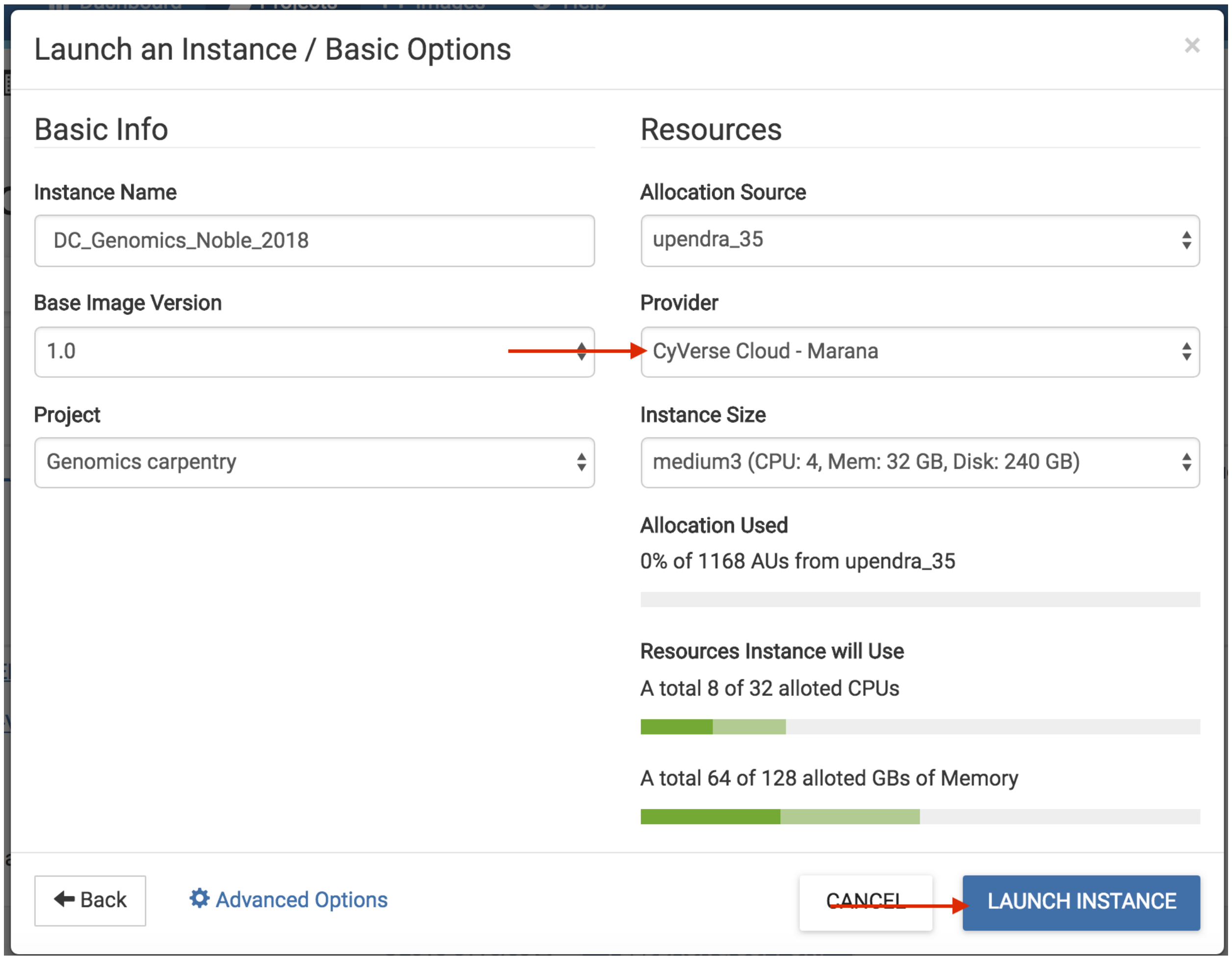
DC_Geno

Showing 1 image(s) for "DC_Geno"

DC_Genomics_Noble_2018
May 4, 2018 03:58 pm
by upendra_35

Data Carpentry Genomics workshop at Noble research institute 2018 image consisting for fastqc, trimmomatic, bwa, samtools, bcftools and IGV.

- Name your virtual machine something simple such as "workshop" or leave it as default which is the image name and select the following"
  - Provider: "CyVerse Cloud - Marana" or "DC_Genomics_Noble"
  - Instance size: "medium3 (CPU: 4, Mem: 32GB, Disk: 240GB)"
  - Leave rest of the fields as default.



Launch an Instance / Basic Options

Basic Info

Instance Name: DC_Genomics_Noble_2018

Base Image Version: 1.0

Project: Genomics carpentry

Resources

Allocation Source: upendra_35

Provider: CyVerse Cloud - Marana

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

Allocation Used

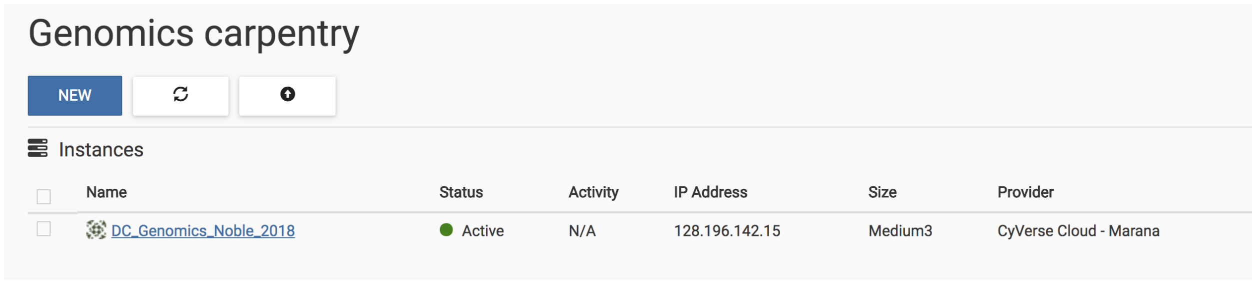
0% of 1168 AUs from upendra_35

Resources Instance will Use

A total 8 of 32 alloted CPUs

A total 64 of 128 alloted GBs of Memory

- Wait for it to become active
- It will now be booting up! This will take 2-5 minutes. Just wait! Don't reload or do anything.One the virtual machine is ready, the "Status" column will turn green and described as "Active" (see below)

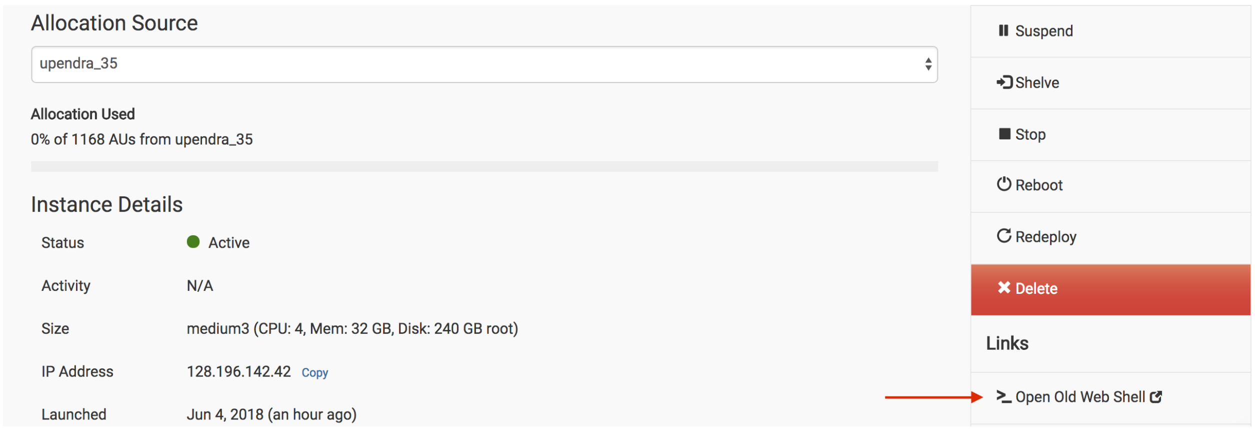


Genomics carpentry

Instances

Name	Status	Activity	IP Address	Size	Provider
DC_Genomics_Noble_2018	● Active	N/A	128.196.142.15	Medium3	CyVerse Cloud - Marana

- Click on your new instance's name to get more information!
- Now, you can either click "Open Web Shell", *or*, if you know how to use ssh, you can ssh in with your CyVerse username on the IP address of the machine



Allocation Source

upendra_35

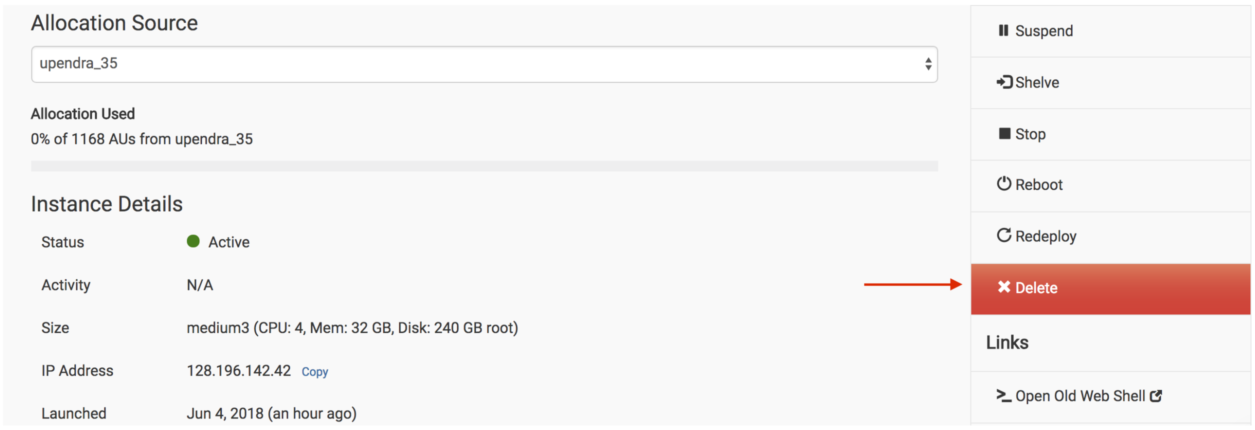
Allocation Used
0% of 1168 AUs from upendra_35

Instance Details

Status	● Active
Activity	N/A
Size	medium3 (CPU: 4, Mem: 32 GB, Disk: 240 GB root)
IP Address	128.196.142.42 Copy
Launched	Jun 4, 2018 (an hour ago)

4. **Deleting your 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.



Allocation Source

upendra_35

Allocation Used
0% of 1168 AUs from upendra_35

Instance Details

Status	● Active
Activity	N/A
Size	medium3 (CPU: 4, Mem: 32 GB, Disk: 240 GB root)
IP Address	128.196.142.42 Copy
Launched	Jun 4, 2018 (an hour ago)

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.

CHAPTER 2

Introducing the Shell

2.1 What is a shell and why should I care?

A shell is a computer program that presents a command line interface which allows you to control your computer using commands entered with a keyboard instead of controlling graphical user interfaces (GUIs) with a mouse/keyboard combination.

There are many reasons to learn about the shell.

- Many bioinformatics tools can only be used through a command line interface, or have extra capabilities in the command line version that are not available in the GUI. This is true, for example, of BLAST, which offers many advanced functions only accessible to users who know how to use a shell.
- The shell makes your work less boring. In bioinformatics you often need to do the same set of tasks with a large number of files. Learning the shell will allow you to automate those repetitive tasks and leave you free to do more exciting things.
- The shell makes your work less error-prone. When humans do the same thing a hundred different times (or even ten times), they're likely to make a mistake. Your computer can do the same thing a thousand times with no mistakes.
- The shell makes your work more reproducible. When you carry out your work in the command-line (rather than a GUI), your computer keeps a record of every step that you've carried out, which you can use to re-do your work when you need to. It also gives you a way to communicate unambiguously what you've done, so that others can check your work or apply your process to new data.
- Many bioinformatic tasks require large amounts of computing power and can't realistically be run on your own machine. These tasks are best performed using remote computers or cloud computing, which can only be accessed through a shell.

In this lesson you will learn how to use the command line interface to move around in your file system.

2.2 How to access the shell?

On a Mac or Linux machine, you can access a shell through a program called Terminal, which is already available on your computer. If you're using Windows, you'll need to download a separate program to access the shell.

We will spend most of our time learning about the basics of the shell by manipulating some experimental data. Some of the data we're going to be working with is quite large, and we're also going to be using several bioinformatics packages in later lessons to work with this data.

You can log-in to the remote server using the instructions here.

CHAPTER 3

The UNIX filesystem

3.1 Learning objectives

- Navigate around the Unix file system
- Explore the content of files
- Learn how to manipulate files

3.2 Navigating your file system

The part of the operating system responsible for managing files and directories is called the file system. It organizes our data into files, which hold information, and directories (also called "folders"), which hold files or other directories.

Several commands are frequently used to create, inspect, rename, and delete files and directories.

3.2.1 Preparation Magic

Tip: If you type the command: PS1='$ ' into your shell, followed by pressing the Enter key, your window should look like our example in this lesson. This isn't necessary to follow along (in fact, your prompt may have other helpful information you want to know about). This is up to you!

$

The dollar sign is a prompt, which shows us that the shell is waiting for input; your shell may use a different character as a prompt and may add information before the prompt. When typing commands, either from these lessons or from other sources, do not type the prompt, only the commands that follow it.

Let's find out where we are by running a command called `pwd` (which stands for "print working directory"). At any moment, our current working directory is our current default directory, i.e., the directory that the computer assumes we want to run commands in unless we explicitly specify something else. Here, the computer's response is `/home/upendra_35`, which is the top level directory within our cloud system:

```
$ pwd
/home/upendra_35
```

Let's look at how our file system is organized.

Let's learn a few commands by investigating the folders in the `dc_sample_data` directory. But since we don't have directory yet, let's download that data from the datastore

```
$ cd /scratch
```

`cd` stands for 'change directory'. We are not changing any directory but actually navigate to that particular directory

Change the permissions of the directory so that you can download data into it.

```
$ sudo chown -hR ${USER} /scratch/
```

`sudo` is root access.

Now download the data into that directory

```
$ wget https://de.cyverse.org/dl/d/CA8E86CA-D089-43AC-93ED-79A257C5815D/osfstorage-
↪archive.zip
```

Warning: It will take few minutes to download the data, depending on the internet connection speed. So please be patient

Once the data is downloaded, you need to uncompress it using `unzip` command

```
$ unzip osfstorage-archive.zip
```

Note: It will take few minutes to uncompress the data, depending on the internet connection speed. So please be patient

Now again we `cd` into `dc_sample_data` folder so that we can investiage it using some basic Linux commands

```
$ cd dc_sample_data
```

Let's see what is inside the folder. Type:

```
$ ls
```

You will see:

```
sra_metadata  untrimmed_fastq
```

`ls` stands for 'list' and it lists the contents of a directory.

There are two items listed. What are they? We can use a command line "modifier" with `ls` to get more information; this modifier is called an argument (more below).

```
$ ls -F
```

```
sra_metadata/  untrimmed_fastq/
```

Anything with a "/" after it is a directory. Things with a "*" after them are programs. If there are no decorations after the name, it's a file.

You can also use the command `ls -l` to see whether items in a directory are files or directories.

```
$ ls -l
```

```
drwxr-xr-x 2 upendra_35 iplant-everyone 4096 Jun  4 17:59 sra_metadata
drwxr-xr-x 2 upendra_35 iplant-everyone 4096 Jun  4 17:59 untrimmed_fastq
```

`ls -l` gives a lot more information too.

Let's go into the `untrimmed_fastq` directory and see what is in there.

```
$ cd untrimmed_fastq

$ ls -F
```

```
SRR097977.fastq  SRR098026.fastq
```

There are two items in this directory with no trailing slashes, so they are files.

3.2.2 Arguments

Most commands take additional arguments that control their exact behavior. For example, `-F` and `-l` are arguments to `ls`. The `ls` command, like many commands, takes a lot of arguments. Another useful one is `-a`, which shows everything, including hidden files.

How do we know what arguments are available for particular commands? Most commonly used shell commands have a manual. You can access the manual using the `man` command. Try entering:

```
$ man ls
```

This will open the manual page for `ls`. Use the `space key` to go forward and `b` to go backwards. When you are done reading, just hit `q` to quit.

Commands that are run from the shell can get extremely complicated. To see an example, open up the manual page for the `find` command. No one can possibly learn all of these arguments, of course. So you will probably find yourself referring to the manual page frequently.

If the manual page within the terminal is hard to read and traverse, the manual exists online, use your web searching powers to get it! In addition to the arguments, you can also find good usage examples online; Google is your friend.

3.3 The Unix directory file structure (a.k.a. where am I?)

As you've already just seen, you can move around in different directories or folders at the command line. You are probably accustomed to navigating around the normal way using a GUI (GUI = Graphical User Interface, pronounced *gooey*), but you will find that it's not too difficult once you get the hang of it.

3.3.1 Moving around the file system

Let's practice moving around a bit. Previously, we moved to the `untrimmed_fastq` directory.

To get there, we first changed directories from the folder of our username to `/scratch` to `dc_sample_data` then we changed directories to `untrimmed_fastq`

Let's draw out how that went.

Now let's draw some of the other files and folders we could have clicked on.

This is called a hierarchical file system structure, like an upside down tree with root (/) at the base that looks like this:

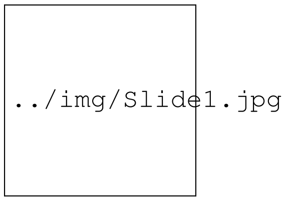


Fig. 1: Unix

That root (/) is often also called the 'top' level.

When you are working at your computer or log in to a remote computer, you are on one of the branches of that tree, your home directory

Now let's go do that same navigation at the command line.

Type:

```
$ cd
```

This puts you in your home directory. No matter where you are in the directory system, `cd` will always bring you back to your home directory.

Now using `cd` and `ls`, go in to the `untrimmed_fastq` directory and list its contents.

Let's also check to see where we are. Sometimes when we're wandering around in the file system, it's easy to lose track of where we are and get lost.

If you want to know what directory you're currently in, type:

```
$ pwd
```

This stands for 'print working directory'. The directory you're currently working in.

What if we want to move back up and out of the `untrimmed_fastq` directory? Can we just type `cd dc_sample_data`? Try it and see what happens.

To go 'back up a level' we need to use `..`

Type

```
$ cd ..
```

Note: `..` denotes parent directory, and you can use it anywhere in the system to go back to the parent directory.

Now do `ls` and `pwd`.

Exercise

Now we're going to try a hunt. Find a hidden directory in `dc_sample_data` and list its contents. What is the name of the text file in the hidden directory?

Hint: hidden files and folders in unix start with `.`, for example `.my_hidden_directory \* \* \*`

3.3.2 Full vs. Relative Paths

The `cd` command takes an argument which is the directory name. Directories can be specified using either a *relative path* or a *full path*. As we know, the directories on the computer are arranged into a hierarchy. The full path tells you where a directory is in that hierarchy.

Navigate to the scratch directory (`cd /scratch`). Now, enter the `pwd` command and you should see:

```
/scratch
```

which is the full path for your home directory. This tells you that you are in a directory called `scratch`, which sits inside the very top directory called `root` which is usually referred to as the *root directory*.

Now enter the following command:

```
$ cd dc_sample_data/.hidden
```

This jumps to `.hidden`. This command takes us to the `hidden` directory. But instead of specifying the full path (`/scratch/dc_sample_data/.hidden`), we specified a *relative path*. In other words, we specified the path **relative to our current directory**.

A full path always starts with a `/`, a relative path does not.

A relative path is like getting directions from someone on the street. They tell you to "go right at the Stop sign, and then turn left on Main Street". That works great if you're standing there together, but not so well if you're trying to tell someone how to get there from another country. A full path is like GPS coordinates. It tells you exactly where something is no matter where you are right now.

You can usually use either a full path or a relative path depending on what is most convenient. If we are in the home directory, it is more convenient to just enter the relative path since it involves less typing.

Over time, it will become easier for you to keep a mental note of the structure of the directories that you are using and how to quickly navigate amongst them.

3.3.3 Examining the contents of other directories

By default, the `ls` commands lists the contents of the working directory (i.e. the directory you are in). You can always find the directory you are in using the `pwd` command. However, you can also give `ls` the names of other directories to view.

Navigate to the `scratch` directory if you are not already there.

Type:

```
$ cd /scratch
```

Then enter the command:

```
$ ls dc_sample_data
```

This will list the contents of the `dc_sample_data` directory without you having to navigate there.

The `cd` command works in a similar way. Try entering:

```
$ cd dc_sample_data/untrimmed_fastq

$ pwd
```

You will jump directly to `untrimmed_fastq` without having to step through the intermediate directory.

List the `SRR097977.fastq` file from your scratch directory without changing directories ****

3.4 Saving time with shortcuts

There are several shortcuts which you should know about, but today we are going to talk about only a few. As you continue to work with the shell and on the terminal a lot more, you will come across and hopefully adapt many other shortcuts.

Dealing with the home directory is very common. So, in the shell the tilde character, ~, is a shortcut for your home directory. Navigate to the `dc_sample_data/sra_metadata/` directory:

```
$ cd /scratch

$ cd dc_sample_data/sra_metadata/
```

Then enter the command:

```
$ ls ~
```

This prints the contents of your home directory, without you having to type the full path.

Another shortcut is the "..", which we encountered earlier:

```
$ ls ..
```

The shortcut `..` always refers to the directory above your current directory. So, it prints the contents of the `/scratch/dc_sample_data`. You can chain these together, so:

```
$ ls ../../
```

prints the contents of `/scratch` which is your home directory.

Finally, the special directory `.` always refers to your current directory.

```
$ ls .
```

So, `ls`, `ls .`, and `ls ././././.` all do the same thing, they print the contents of the current directory. This may seem like a useless shortcut right now, but it is needed to specify a destination, e.g. `cp ../data/counts.txt .` or `mv ~/james-scripts/parse-fasta.sh .`.

To summarize, while you are in your home directory, the commands `ls ~`, `ls ~/.`, and `ls /home/dcuser` all do exactly the same thing. These shortcuts are not necessary, but they are really convenient!

3.4.1 Tab Completion

Tab completion is an important shortcut to know; it improves efficiency navigating the file system and helps avoid typos.

To practice with tab completion, let's first navigate to `scratch` directory.

```
$ cd /scratch
```

Typing out directory names can waste a lot of time. When you start typing out the name of a directory, then hit the tab key, the shell will try to fill in the rest of the directory name. For example, type `cd` to get back to your home directy, then enter:

```
$ cd /scratch/dc_<>
```

The shell will fill in the rest of the directory name for `dc_sample_data`. Now go to `dc_sample_data/untrimmed_fastq`

```
$ cd un<tab>
$ ls SR<tab><tab>
```

When you hit the first tab, nothing happens. The reason is that there are multiple directories in the home directory which start with `SR`. Thus, the shell does not know which one to fill in. When you hit tab again, the shell will list the possible choices.

3.4.2 Wild cards

Navigate to the `/scratch/dc_sample_data/untrimmed_fastq` directory. This directory contains FASTQ files from our RNA-Seq experiment.

The `*` character is a shortcut for "everything". Thus, if you enter `ls *`, you will see all of the contents of a given directory. Now try this command:

```
$ ls *fastq
```

This lists every file that ends with a `fastq`. This command:

```
$ ls SRR*
```

lists every file in that starts with the characters `SRR`.

```
$ ls *977.fastq
```

Lists only the file that ends with '977.fastq'

So how does this actually work? Well. . . when the shell (bash) sees a word that contains the `*` character, it automatically looks for filenames that match the given pattern.

Exercise

1. Change directories to your home directory, and list the contents of `/scratch/dc_sample_data/sra_metadata/` without changing directories again.
2. List the contents of the /bin directory. Do you see anything familiar in there? How can you tell these are programs rather than plain files?
3. Do each of the following using a single `ls` command without navigating to a different directory.
   (a) List all of the files in `/bin` that start with the letter 'c
   (b) List all of the files in `/bin` that contain the letter 'a'
   (c) List all of the files in `/bin` that end with the letter 'o'

BONUS: List all of the files in '/bin' that contain the letter 'a' or 'c'

3.5 Command History

You can easily access previous commands. Hit the up arrow. Hit it again. You can step backwards through your command history. The down arrow takes your forwards in the command history.

`^-C` will cancel the command you are writing, and give you a fresh prompt.

`^-R` will do a reverse-search through your command history. This is very useful.

You can also review your recent commands with the `history` command. Just enter:

```
$ history
```

to see a numbered list of recent commands, including this just issued `history` command. You can reuse one of these commands directly by referring to the number of that command.

If your history looked like this:

```
259  ls *
260  ls /usr/bin/*.sh
261  ls *R1*fastq
```

then you could repeat command #260 by simply entering:

```
$ !260
```

(that's an exclamation mark). You will be glad you learned this when you try to re-run very complicated commands.

Exercise

1. Find the line number in your history for the last exercise (listing files in /bin) and reissue that command.

We now know how to move around the file system and look at the contents of directories, but how do we look at the contents of files?

The easiest way (but really not the ideal way in most situations) to examine a file is to just print out all of the contents using the command `cat`. Enter the following command:

```
$ cd /scratch/dc_sample_data/sra_metadata
$ cat SraRunTable.txt
```

This prints out the all the contents of the the `SraRunTable.txt` to the screen.

Note: `cat` stands for concatenate; it has many uses and printing the contents of a files onto the terminal is one of them.

What does this file contain?

`cat` is a terrific command, but when the file is really big, it should be avoided; `less`, is preferred for files larger than a few bytes. Let's take a look at the fastq files in `/scratch/dc_sample_data/untrimmed_fastq`. These files are quite large, so we probably do not want to use the `cat` command to look at them. Instead, we can use the `less` command.

Move to the `untrimmed_fastq` directory and enter the following command:

```
$ cd /scratch/dc_sample_data/untrimmed_fastq/
$ less SRR098026.fastq
```

`less` opens the file, and lets you navigate through it. The commands are identical to the `man` program.

Some commands in ‘‘less‘‘

key	action
“space”	to go forward
“b”	to go backwarsd
“g”	to go to the beginning
“G”	to go to the end
“q”	to quit

`less` also gives you a way of searching through files. Just hit the `/` key to begin a search. Enter the name of the word you would like to search for and hit enter. It will jump to the next location where that word is found. If you hit `/` then “enter”, `less` will just repeat the previous search. `less` searches from the current location and works its way forward. If you are at the end of the file and search the word, `less` will not find it. You need to go to the beginning of the file and search.

For instance, let’s search for the sequence `GTTGATC` in our file. You can see that we go right to that sequence and can see what it looks like.

Remember, the `man` program actually uses `less` internally and therefore uses the same commands, so you can search documentation using `/` as well!

There’s another way that we can look at files, and in this case, just look at part of them. This can be particularly useful if we just want to see the beginning or end of the file, or see how it’s formatted.

The commands are `head` and `tail` and they just let you look at the beginning and end of a file respectively.

```
$ head SRR098026.fastq

$ tail SRR098026.fastq
```

The `-n` option to either of these commands can be used to print the first or last `n` lines of a file. To print the first line of the file use:

```
$ head -n 1 SRR098026.fastq
```

3.6 Creating, moving, copying, and removing

Now we can move around in the file structure, look at files, and search files. But what if we want to do normal things like copy files or move them around or get rid of them.

Our raw data in this case is fastq files. We don’t want to change the original files, so let’s make a copy to work with.

Lets copy the file using the `cp` command. The copy command requires 2 things, the name of the file to copy, and the location to copy it to. Navigate to the `untrimmed_fastq` directory and enter:

```
$ cp SRR098026.fastq SRR098026-copy.fastq

$ ls -F
```

```
SRR097977.fastq  SRR098026-copy.fastq  SRR098026.fastq
```

Now `SRR098026-copy.fastq` has been created as a copy of `SRR098026.fastq`

Let's make a `backup` directory where we can put this file.

The `mkdir` command is used to make a directory. Just enter `mkdir` followed by a space, then the directory name.

```
$ mkdir backup
```

We can now move our backed up file in to this directory. We can move files around using the command `mv`. Enter this command:

```
$ mv *-copy.fastq backup

$ ls -al backup
```

```
total 52
drwxr-xr-x 2 upendra_35 iplant-everyone  4096 Jun  4 19:20 .
drwxr-xr-x 3 upendra_35 iplant-everyone  4096 Jun  4 19:20 ..
-rw------- 1 upendra_35 iplant-everyone 43332 Jun  4 19:19 SRR098026-copy.fastq
```

The `mv` command is also how you rename files. Since this file is so important, let's rename it:

```
$ cd backup

$ mv SRR098026-copy.fastq SRR098026-copy.fastq_DO_NOT_TOUCH!

$ ls
```

```
SRR098026-copy.fastq_DO_NOT_TOUCH!
```

Finally, we decided this was silly and want to start over.

```
$ rm SRR098026-copy.fastq_DO_NOT_TOUCH!
```

Note: The `rm` file permanently removes the file. Be careful with this command. The shell doesn't just nicely put the files in the Trash, they're really gone!

Same with moving and renaming files. It will **not** ask you if you are sure that you want to "replace existing file".

1. Change directories to the `untrimmed_fastq` folder and create a backup directory called `new_backup`

3.7 2. Copy both fastq files files there with 1 command

By default, `rm`, will NOT delete directories. You can tell `rm` to delete a directory using the `-r` option. Let's delete our `backup` and `new_backup` directories we just made. Enter the following command:

```
$ rm -r backup
$ rm -r new_backup
```

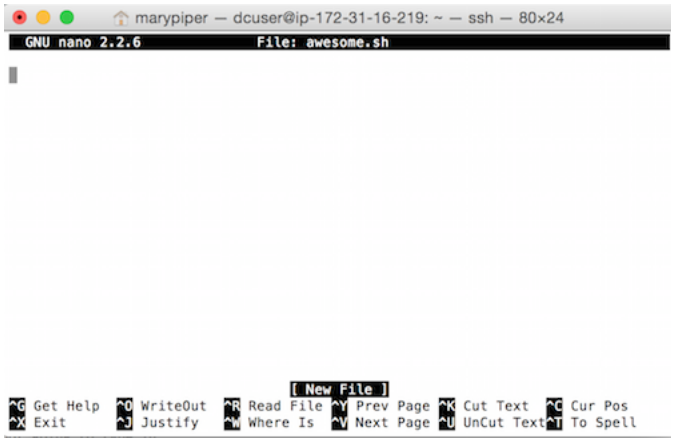
3.8 Writing files

We've been able to do a lot of work with files that already exist, but what if we want to write our own files. Obviously, we're not going to type in a FASTA file, but you'll see as we go through other tutorials, there are a lot of reasons we'll want to write a file, or edit an existing file.

To write in files, we're going to use the program `nano`. We're going to create a file within your home direcory with the name `awesome.sh`.

```
$ cd /scratch
$ nano awesome.sh
```

Now you have something that looks like

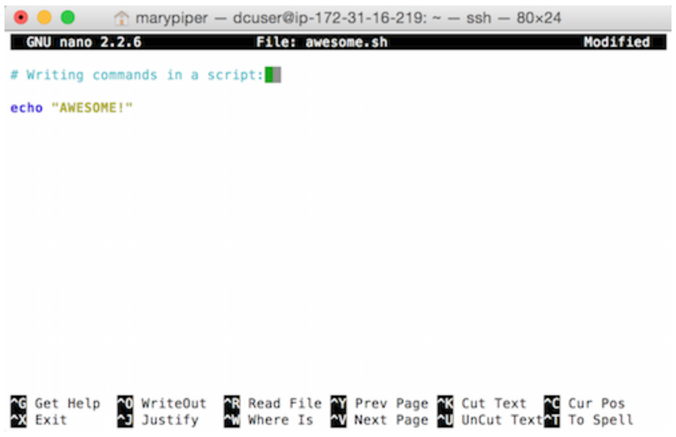


```
  GNU nano 2.2.6               File: awesome.sh



                             [ New File ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Type in your command, so it looks like



```
  GNU nano 2.2.6               File: awesome.sh                        Modified

# Writing commands in a script:

echo "AWESOME!"



^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Now we want to save the file and exit. At the bottom of nano, you see the "^X Exit". That means that we use Ctrl-X to exit. Type `Ctrl-X`. It will ask if you want to save it. Type `y` for yes. Then it asks if you want that file name. Hit 'Enter'.

Now you've written a file. You can take a look at it with `less` or `cat`, or open it up again and edit it.

We're going to come back and use this file in just a bit.

CHAPTER 4

Redirection

4.1 Learning objectives

- Practice searching for characters or patterns in a text file using the `grep` command
- Learn about output redirection
- Explore how to use the pipe (`|`) character to chain together commands

4.2 Searching files

We showed a little how to search within a file using `less`. We can also search within files without even opening them, using `grep`. Grep is a command-line utility for searching plain-text data sets for lines matching a string or regular expression. Let's give it a try!

Suppose we want to see how many reads in our file have really bad, with 10 consecutive Ns
Let's search for the string NNNNNNNNNN in file `SRR098026.fastq` in the `untrimmed_fastq` folder:

```
$ cd /scratch/dc_sample_data/untrimmed_fastq/
$ grep NNNNNNNNNN SRR098026.fastq
```

We get back a lot of lines. What is we want to see the whole fastq record for each of these read. We can use the `-B` argument for grep to return the matched line plus one before (-B 1) and two lines after (-A 2). Since each record is four lines and the last second is the sequence, this should give the whole record.

```
$ grep -B1 -A2 NNNNNNNNNN SRR098026.fastq
```

for example:

```
@SRR098026.177 HWUSI-EAS1599_1:2:1:1:2025 length=35
CNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
+SRR098026.177 HWUSI-EAS1599_1:2:1:1:2025 length=35
#!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

1. Search for the sequence GNATNACCACTTCC in SRR098026.fastq. In addition to finding the sequence, have your search also return the name of the sequence.
2. Search for that sequence in both fastq files. ****

4.3 Redirection

We're excited we have all these sequences that we care about that we just got from the FASTQ files. That is a really important motif that is going to help us answer our important question. But all those sequences just went whizzing by with `grep`. How can we capture them?

We can do that with something called "redirection". The idea is that we're redirecting the output to the terminal (all the stuff that went whizzing by) to something else. In this case, we want to print it to a file, so that we can look at it later.

The redirection command for putting something in a file is >

Let's try it out and put all the sequences that contain 'NNNNNNNNNN' from all the files in to another file called 'bad_reads.txt'

```
$ grep -B1 -A2 NNNNNNNNNN SRR098026.fastq > bad_reads.txt
```

The prompt should sit there a little bit, and then it should look like nothing happened. But type `ls`. You should have a new file called `bad_reads.txt`. Take a look at it and see if it has what you think it should.

If we use >>, it will append to rather than overwrite a file. This can be useful for saving more than one search, for example:

```
$ grep -B1 -A2 NNNNNNNNNN SRR097977.fastq >> bad_reads.txt
```

There's one more useful redirection command that we're going to show, and that's called the pipe command, and it is `|`. It's probably not a key on your keyboard you use very much. What `|` does is take the output that scrolling by on the terminal and then can run it through another command. When it was all whizzing by before, we wished we could just slow it down and look at it, like we can with `less`. Well it turns out that we can! We pipe the `grep` command through `less`

```
$ grep -B1 -A2 NNNNNNNNNN SRR098026.fastq | less
```

Now we can use the arrows to scroll up and down and use `q` to get out.

We can also do something tricky and use the command `wc`. `wc` stands for `word count`. It counts the number of lines or characters. So, we can use it to count the number of lines we're getting back from our `grep` command. And that will magically tell us how many sequences we're finding. We're

```
$ grep -B1 -A2 NNNNNNNNNN SRR098026.fastq | wc
```

That tells us the number of lines, words and characters in the file. If we just want the number of lines, we can use the `-l` flag for `lines`.

```
$ grep -B1 -A2 NNNNNNNNNN SRR098026.fastq | wc -l
```

Redirecting is not super intuitive, but it's really powerful for stringing together these different commands, so you can do whatever you need to do.

The philosophy behind these command line programs is that none of them really do anything all that impressive. BUT when you start chaining them together, you can do some really powerful things really efficiently. If you want to be proficient at using the shell, you must learn to become proficient with the pipe and redirection operators: `|`, `>`, `>>`.

4.4 Practicing searching and redirection

Finally, let's use the new tools in our kit and a few new ones to example our SRA metadata file.

```
$ cd ../sra_metadata/
$ ls
```

Take a look at the metadata file, `SraRunTable.txt`:

```
$ less SraRunTable.txt
```

Let's ask a few questions about the data

1. How many of the read libraries are paired end?

First, what are the column headers?

```
$ head -n 1 SraRunTable.txt
```

```
BioSample_s InsertSize_l    LibraryLayout_s Library_Name_s  LoadDate_s  MBases_l   ␣
↪MBytes_l    ReleaseDate_s Run_s SRA_Sample_s Sample_Name_s Assay_Type_s␣
↪AssemblyName_s BioProject_s Center_Name_s Consent_s Organism_Platform_s SRA_Study_s␣
↪g1k_analysis_group_s g1k_pop_code_s source_s strain_s
```

That's only the first line but it is a lot to take in. 'cut' is a program that will extract columns in tab-delimited files. It is a very good command to know. Lets look at just the first four columns in the header using the `\|` redirect and 'cut'

```
$ head -n 1 SraRunTable.txt | cut -f 1-4
```

```
BioSample_s InsertSize_l      LibraryLayout_s   Library_Name_s
```

`-f 1-4` means to cut the first four fields (columns). The LibraryLayout_s column looks promising. Let's look at some data for just that column.

```
$ cut -f3 SraRunTable.txt | head -n 10
```

```
LibraryLayout_s
SINGLE
SINGLE
SINGLE
SINGLE
SINGLE
SINGLE
SINGLE
SINGLE
PAIRED
```

We can see that there are (at least) two categories, SINGLE and PAIRED. We want to search all entries in this column for just PAIRED and count the number of hits.

```
$ cut -f3 SraRunTable.txt | grep PAIRED | wc -l
```

```
2
```

2. How many of each class of library layout are there?

We can use some new tools 'sort' and 'uniq' to extract more information. For example, cut the third column, remove the header and sort the values. The `-v` option for grep means return all lines that DO NOT match.

```
$ cut -f3 SraRunTable.txt | grep -v LibraryLayout_s | sort
```

This returns a sorted list (too long to show here) of PAIRED and SINGLE values. Now we can use 'uniq' with the '-c' flag to count the different categories.

```
$ cut -f3 SraRunTable.txt | grep -v LibraryLayout_s |   sort | uniq -c
```

```
 2 PAIRED
35 SINGLE
```

3. Sort the metadata file by PAIRED/SINGLE and save to a new file We can use if '-k' option for sort to specify which column to sort on. Note that this does something similar to cut's '-f'.

```
$ sort -k3 SraRunTable.txt > SraRunTable_sorted_by_layout.txt
```

4. Extract only paired end records into a new file Do we know PAIRED only occurs in column 4? WE know there are only two in the file, so let's check.

```
$ grep PAIRED SraRunTable.txt | wc -l
```

```
2
```

OK, we are good to go.

```
$ grep PAIRED SraRunTable.txt > SraRunTable_only_paired_end.txt
```

1. How many sample load dates are there?
2. Filter subsets into new files bases on load date ****

4.5 Where can I learn more about the shell?

- Software Carpentry tutorial - The Unix shell
- The shell handout - Command Reference
- explainshell.com
- http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
- man bash
- Google - if you don't know how to do something, try Googling it. Other people have probably had the same question.
- Learn by doing. There's no real other way to learn this than by trying it out. Write your next paper in nano (really emacs or vi), open pdfs from the command line, automate something you don't really need to automate.

4.6 Bonus:

alias

.bashrc

ssh and scp

Regular Expressions

Permissions

Chaining commands together

md5sum

CHAPTER 5

Writing Scripts

5.1 Learning Objectives

- Capture previous commands into a script to re-run in one single command
- Understanding variables and storing information
- Learn how to use variables to operate on multiple files

Now that you've been introduced to a number of commands to interrogate your data, wouldn't it be great if you could do this for each set of data that comes in, without having to manually re-type the commands?

Welcome to the beauty and purpose of shell scripts.

5.2 Shell scripts

Shell scripts are text files that contain commands we want to run. As with any file, you can give a shell script any name and usually have the extension `.sh`. For historical reasons, a bunch of commands saved in a file is usually called a shell script, but make no mistake, this is actually a small program.

We are going to take the commands we repeat frequently and save them into a file so that we can **re-run all those operations** again later by typing **one single command**. Let's write a shell script that will do two things:

1. Tell us what is our current working directory
2. Lists the contents of the directory

First let's move into the `/scratch/dc_sample_data` directory and create a new file using `nano`:

```
$ cd /scratch/dc_sample_data
$ nano listing.sh
```

Type in the following lines in the `listing.sh` file:

```
echo "Your current working directory is:"
pwd
echo "These are the contents of this directory:"
ls -l
```

> The `echo` command is a utility for writing to standard output. By providing text in quotations after the command we indicated what it is we wanted written

Save the file and exit `nano`. Now let's run the new script we have created. To run a shell script you usually use the `bash` or `sh` command.

```
$ sh listing.sh
```

> Did it work like you expected?
>
> Were the `echo` commands helpful in letting you know what came next?

This is a very simple shell script. Let's write some interesting ones.

One thing we will commonly want to do with sequencing results is pull out bad reads and write them to a file to see if we can figure out what's going on with them. We're going to look for reads with long sequences of N's like we did before, but now we're going to write a script, so we can run it each time we get new sequences, rather than type the code in by hand each time.

Bad reads have a lot of N's, so we're going to look for NNNNNNNNNN with grep. We want the whole FASTQ record, so we're also going to get the one line above the sequence and the two lines below. We also want to look in all the files that end with .fastq, so we're going to use the * wild card.

```
$ grep -B1 -A2 NNNNNNNNNN *.fastq > scripted_bad_reads.txt
```

We're going to create a new file to put this command in. We'll call it `bad-reads-script.sh`. The `sh` isn't required, but using that extension tells us that it's a shell script.

```
$ nano bad-reads-script.sh
```

Type your grep command into the file and save it as before.

Now comes the neat part. We can run this script. Type:

```
$ bash bad-reads-script.sh
```

It will look like nothing happened, but now if you look at `scripted_bad_reads.txt`, you can see that there are now reads in the file.

How many bad reads are there in the two FASTQ files combined?

Bonus: How many bad reads are in each of the two FASTQ files? (Hint: You will need to use the cut command with the -d flag.)

We want the script to tell us when it's done.

Open `bad-reads-script.sh` and add the line echo "Script finished!" after the grep command and save the file.

5.3 Run the updated script.

5.4 Making the script into a program

We had to type `sh` or `bash` because we needed to tell the computer what program to use to run this script. Instead we can turn this script into its own program. We need to tell it that it's a program by making it executable. We can do this by changing the file permissions. We talked about permissions in an earlier episode.

First, let's look at the current permissions.

```
$ ls -l bad-reads-script.sh
-rw-r--r-- 1 upendra_35 iplant-everyone 57 Jun  4 20:05 bad-read-script.sh
```

We see that it says `-rw-r--r--`. This shows that the file can be read by any user and written to by the file owner (you). We want to change these permissions so that the file can be executed as a program. We use the command `chmod` like we did earlier when we removed write permissions. Here we are adding (+) executable permissions (+x).

```
$ chmod +x bad-reads-script.sh
```

Now let's look at the permissions again.

```
$ ls -l bad-reads-script.sh
-rwxr-xr-x 1 upendra_35 iplant-everyone 57 Jun  4 20:05 bad-read-script.sh
```

Now we see that it says `-rwxr-xr-x`. The x's that are there now tell us we can run it as a program. So, let's try it! We'll need to put `./` at the beginning so the computer knows to look here in this directory for the program.

```
$ ./bad-reads-script.sh
```

The script should run the same way as before, but now we've created our very own computer program!

CHAPTER 6

Getting your project started

Project organization is one of the most important parts of a sequencing project, and yet is often overlooked amidst the excitement of getting a first look at new data. Of course, while it's best to get yourself organized before you even begin your analyses, it's never too late to start, either.

You should approach your sequencing project similarly to how you do a biological experiment and this ideally begins with experimental design. We're going to assume that you've already designed a beautiful sequencing experiment to address your biological question, collected appropriate samples, and that you have enough statistical power to answer the questions you're interested in asking. These steps are all incredibly important, but beyond the scope of our course. For all of those steps (collecting specimens, extracting DNA, prepping your samples) you've likely kept a lab notebook that details how and why you did each step. However, the process of documentation doesn't stop at the sequencer!

Genomics projects can quickly accumulate hundreds of files across tens of folders. Every computational analysis you perform over the course of your project is going to create many files, which can especially become a problem when you'll inevitably want to run some of those analyses again. For instance, you might have made significant headway into your project, but then have to remember the PCR conditions you used to create your sequencing library months prior.

Other questions might arise along the way:

- What were your best alignment results?
- Which folder were they in: Analysis1, AnalysisRedone, or AnalysisRedone2?
- Which quality cutoff did you use?
- What version of a given program did you implement your analysis in?
- Good documentation is key to avoiding this issue, and luckily enough, recording your computational experiments is even easier than recording lab data. Copy/Paste will become your best friend, sensible file names will make your analysis understandable by you and your collaborators, and writing the methods section for your next paper will be easy! Remember that in any given project of yours, it's worthwhile to consider a future version of yourself as an entirely separate collaborator. The better your documenation is, the more this 'collaborator' will feel indebted to you!

With this in mind, let's have a look at the best practices for documenting your genomics project. Your future self will thank you.

In this exercise we will setup a file system for the project we will be working on during this workshop.

We will start by creating a directory that we can use for the rest of the workshop. First navigate to the `scratch` directory. Then confirm that you are in the correct directory using the `pwd` command.

```
$ cd /scratch
$ pwd
```

You should see the output:

Use the `mkdir` command to make the following directories:

dc_workshop dc_workshop/docs dc_workshop/data dc_workshop/results ——————-

Now run to the directories and sub-directories

```
$ ls -R dc_workshop
```

You should see the following output:

```
dc_workshop/:
data  docs  results

dc_workshop/data:

dc_workshop/docs:

dc_workshop/results:
```

6.1 Organizing your files

Before begining any analysis, it's important to save a copy of your raw data. The raw data should never be changed. Regardless of how sure you are that you want to carry out a particular data cleaning step, there's always the chance that you'll change your mind later or that there will be an error in carrying out the data cleaning and you'll need to go back a step in the process. Having a raw copy of your data that you never modify guarantees that you will always be able to start over if something goes wrong with your analysis. When starting any analysis, you can make a copy of your raw data file and do your manipulations on that file, rather than the raw version. We learned in a previous episode how to prevent overwriting our raw data files by setting restrictive file permissions.

You can store any results that are generated from your analysis in the results folder. This guarantees that you won't confuse results file and data files in six months or two years when you are looking back through your files in preparation for publishing your study.

The docs folder is the place to store any written analysis of your results, notes about how your analyses were carried out, and documents related to your eventual publication.

Documenting your activity on the project:

When carrying out wet-lab analyses, most scientists work from a written protocol and keep a hard copy of written notes in their lab notebook, including any things they did differently from the written protocol. This detailed record-keeping process is just as important when doing computational analyses. Luckily, it's even easier to record the steps you've carried out computational than it is when working at the bench.

The `history` command is a convenient way to document all the commands you have used while analyzing and manipulating your project files. Let's document the work we have done on our project so far.

View the commands that you have used so far during this session using history:

```
$ history
```

The history likely contains many more commands than you have used for the current project. Let's view the last several commands that focus on just what we need for this project.

View the last n lines of your history (where n = approximately the last few lines you think relevant). For our example, we will use the last 7:

```
$ history | tail -n 7
```

Using your knowledge of the shell, use the append redirect >> to create a file called dc_workshop_log_XXXX_XX_XX.txt (Use the four-digit year, two-digit month, and two digit day, e.g. dc_workshop_log_2017_10_27.txt)

You may have noticed that your history contains the history command itself. To remove this redundancy from our log, let's use the nano text editor to fix the file:

```
$ nano dc_workshop_log_2017_10_27.txt
```

Note: Remember to replace the 2017_10_27 with your workshop date.

From the `nano` screen, you can use your cursor to navigate, type, and delete any redundant lines.