Data Science for Biologists - 5023Y

Philip Leftwich

2021-09-26

Contents

Chapter 1

Introduction

1.1 Approach and style

This book is designed to accompany the module BIO-5023Y for those new to R looking for best practices and tips. So it must be both accessible and succinct. The approach here is to provide just enough text explanation that someone very new to R can apply the code and follow what the code is doing. It is not a comprehensive textbook.

A few other points:

This is a code reference book accompanied by relatively brief examples - not a thorough textbook on R or data science

This is intended to be a living document - optimal R packages for a given task change often and we welcome discussion about which to emphasize in this handbook

Top tips for the course:

DON'T worry if you don't understand everything

DO ask lots of questions!

Xie (2020)

Chapter 2

Unix

Unix is very likely the most fundamental skillset we can develop for bioinformatics (and much more than bioinformatics). Many of the most common and powerful bioinformatics approaches happen in this text-based environment, and having a solid foundation here can make everything we're trying to learn and do much easier. This is a short introductory tutorial to help us get from being completely new to Unix up to being friendly with it

2.1 What is Unix/Linux?

UNIX is a computer operating system. It was first developed in 1969 at Bell Labs. Unix is written in the programming language `C`.

Unix is proprietary software, whereas Linux is *basically* free and open-source Unix.

The Linux Operating System is highly flexible, free, open-source (like R) and uses very little RAM to run (Unlike Windows OS) - as such you find most supercomputers run on Linux. Operationally Linux is almost identical to Unix, and so we often refer to it under the umbrella term of "unix-Like" systems.

2.1.1 Some terms

Here are some terms worth knowing, don't worry about memorising them, it can just be useful to have these to refer to in the future.

Term	What it is
shell	what we use to talk to the computer; anything where you are pointing and clicking with a mouse
command line	a text-based environment capable of taking input and providing output
Terminal	A program that runs a shell

Unix	a family of operating systems (we also use the term “Unix-like” because one of the m
Linux	a ”Unix-like” OS
bash	the most common programming language used at a Unix command-line
flag	a way to set options for a function, a specific type of argument usually preceded by a

Note

You should be very familiar with using a GUI (RStudio), but remember we have spent a lot of time working with files and directories using the the command line (CLI) in R. This is useful practice, because most supercomputers lack a GUI, you must work entirely using the command line.

2.2 Why Learn Unix?

Most sequencing data files are large, and require a lot of RAM to process. As a result most of the work Bioinformaticians do is not hosted on their own computers, instead they “remote-connect” to high performance supercomputers or cluster computers. Almost all of these high performance computers use “Unix-like” operatings systems, the most common of which is Linux.

As stated above Linux is free (so no expensive licenses), open-source so lots of developers, its also well known for being stable, secure, reliable and efficient.

You already have some experience with using a Linux OS - every time you log into RStudio Cloud you are connecting to a supercomputer that runs on Linux. Normally we do not interact directly with the OS, instead we use R and RStudio directly.

But when you click on the RStudio Terminal it provides direct access to a command-line where we can execute commands and functions directly in Linux.

This allows us to start using programs other than R, and potentially use multiple programs & programming languages to work together.

Note

This series of practicals is designed for you to have a first introduction to Bioinformatics, it’s about exposure, not memorising or mastering anything. Don’t worry about the details!

2.3 Getting started

Before we get started we need a terminal to work in.

- Open the Bioinformatics RStudio Cloud Project in the 5023Y workspace

- Click on the `Terminal` tab next to `Console` in the bottom-left pane of the RStudio GUI, this opens a command-line *Shell*

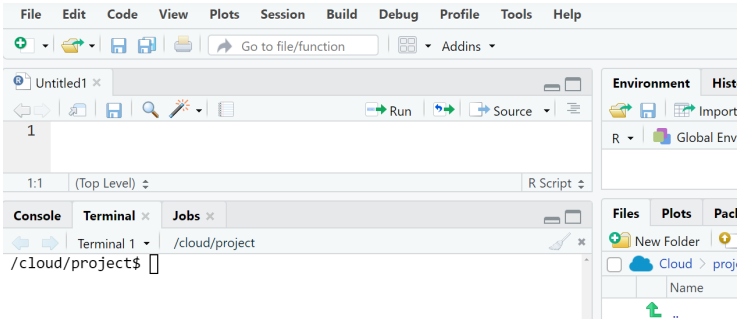


Figure 2.1: Here is an example of the Terminal tab, right next to the R console

- This is our "command line" where we will be typing all of our commands. We type our commands in `bash`
- The $ is where you start typing from, left of this it tells you what folder you are currently in (`working directory`)
- If you need to, you can exit the Terminal and start a new session easily with options in RStudio

2.4 A few foundational rules

- Just like in R spaces are special, spaces break things apart, as a rule it is therefore better to have functions and file names with dashes (-) or underscores (_) - e.g. "draft_v3.txt" is preferred to "draft v3.txt".
- The general syntax on the command line is: `command argument`. Again this is very similar to R except we don't use brackets e.g. in R we are used to `command(argument)`
- Arguments can be **optional** e.g. if their is a default argument you may not have to write anything. Some functions *require* that arguments are specified. Again this is just like R.

2.5 Let's get started

We will perform a very simple function and get a flavour of the similarities and differences to working in `R`.

`date` is a command that prints out the date and time. Copy and paste this command into your terminal

```
date
```

This prints out the date/time in UTC

More information on using the date function (here)[https://www.geeksforgeeks.org/date-command-linux-examples/]

We can also ask for the output for a particular timezone using the `TZ` function and `date`

```
TZ=Europe/London date
```

Or we can ask the computer what the date will be next Tuesday…

```
date --date="next tue"
```

2.5.1 Downloading data

We will start by typing in an instruction to download data from an online data repository, unpack the contents and inspect it:

- curl is a command line tool for transferring data to and from the server here we will use this to download data from an online repository.
- tar will *unpack* the data from a compressed file format
- cd change the directory so we *land* in the new folder we have made

```
curl -L -o unix_intro.tar.gz https://ndownloader.figshare.com/files/15573746
tar -xzvf unix_intro.tar.gz && rm unix_intro.tar.gz
cd unix_intro
```

Check each command line has run, in the example above you might find that the first two lines run, to download and unpack data, while the last line to change directory doesn't run until you hit enter

2.5.2 More functions

Unlike date, most commands require arguments and won't work without them. head is a command that prints the first lines of a file, so it requires us to provide the file we want it to act on:

```
head example.txt
```

Here "example.txt" is the required argument, and in this case it is also what's known as a positional argument. Whether things need to be provided as positional arguments or not depends on how the command or program we are using was written.

Sometimes we need to specify the input file by putting something in front of it (e.g. some commands will use the -i flag, but it's often other things as well).

Q. What's in the text file? - Click here for Answer

Pretty boring, each line contains the text "This is line" followed by the line number e.g.

- *This is line 1*

- *This is line 2*

etc.

There are also optional arguments for the head command. The default for head is to print the first 10 lines of a file. We can change that by specifying the -n flag, followed by how many lines we want:

```
head -n 5 example.txt
```

How would we know we needed the -n flag for that? There are a few ways to find out. Many standard Unix commands and other programs will have built-in help menus that we can access by providing –help as the only argument:

```
head --help
```

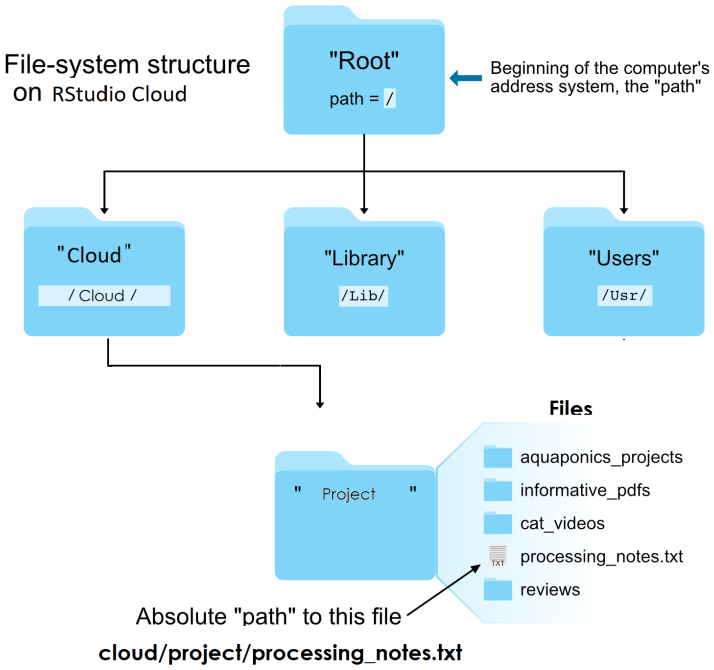
Again this is very similar to the logic in which R commands are strucutred e.g. `??ggplot()` The synatx is similar even if the specific icons or arguments are different.

Remember just like with R, one of your best friends is Google! As you get familiar with any language or OS we might remember a few flags or specific options, but searching for options and details when needed is definitely the norm!

2.6 Unix File Structure

There are two special locations in all Unix-based systems: the "root" location and the current user's "home" location. "Root" is where the address system of the computer starts; "home" is usally where the current user's location starts.

Just to be awkward RStudio Cloud actually has us working in a different location "Cloud", which is underneath Root but separate to home which would be in the "Users" folder.



We tell the command line where files and directories are located by providing their address, their "path". If we use the pwd command (for print working directory), we can find out what the path is for the directory we are sitting in.

```
pwd
```

And if we use the ls command (for list), we can see what directories and files are in the current directory we are sitting in.

```
ls
```

Note

Why is it important to know this? Usually when you are working on a Unix-like environment there is no GUI (nice-click and point interface), all commands have to be submitted through the terminal. So you would have to get used to navigating directories with typed commands, and it's useful to know what the standard hierarchy is.

2.7 Absolute vs relative file paths

You should be used to these concepts from your work with R projects.

There are two ways to specify the path (address) of the file we want to do something to:

- An **absolute path** is an address that starts from an explicitly specified location: usually the "root" `/` or the "home" `~/` location. (Side note, because we also may see or hear the term, the "full path", is usually the absolute path that starts from the "root" /.)
- A **relative path** is an address that starts from wherever we are currently sitting (the working directory). For example, let's look again at the head command we ran above:

```
head example.txt
```

What we are actually doing here is using a relative path to specify where the "example.txt" file is located. This is because the command line **automatically looks** in the current working directory if we don't specify anything else about its location.

We can also run the same command on the same file using an **absolute path** - note Rstudio cloud has a slightly unique set-up in that we start from a folder designated cloud:

```
head /cloud/project/unix_intro/example.txt
```

The previous two commands both point to the same file right now. But the first way, head example.txt, will only work if we are entering it while "sitting" in the directory that holds that file, while the second way will work no matter where we happen to be in the computer.

It is **important to always think about where** we are in the computer when working at the command line. One of the most common errors/easiest mistakes to make is trying to do something to a file that isn't where we think it is. Let's run head on the "example.txt" file again, and then let's try it on another file: "notes.txt":

```
head example.txt
```

```
head notes.txt
```

Here the head command works fine on "example.txt", but we get an error message when we call it on "notes.txt" telling us no such file or directory. If we run the ls command to list the contents of the current working directory, we can see the computer is absolutely right – spoiler alert: it usually is – and there is no file here named "notes.txt".

The ls command by default operates on the current working directory if we don't specify any location, but we can tell it to list the contents of a different directory by providing it as a positional argument:

```
ls
```

```
ls experiment
```

We can see the file we were looking for is located in the subdirectory called "experiment". Here is how we can run head on "notes.txt" by specifying an accurate relative path to that file:

```
head experiment/notes.txt
```

2.8 Moving around

We can also move into the directory containing the file we want to work with by using the `cd` command (**c**hange **d**irectory). This command takes a positional argument that is the path (address) of the directory we want to change into. This can be a relative path or an absolute path. Here we'll use the relative path of the subdirectory, "experiment", to change into it

```
cd experiment/
```

```
pwd
```

```
ls
```

```
head notes.txt
```

Great. But now how do we get **back "up"** to the directory above us? One way would be to provide an absolute path, like `cd /cloud/project/unix_intro`, but there is also a handy shortcut. `..` which are special characters that act as a relative path specifying "up" one level – one directory – from wherever we currently are.

So we can provide that as the positional argument to cd to get back to where we started:

```
cd ..
```

Moving around the computer like this might feel a bit cumbersome and frustrating at first, but after spending a little time with it, you will get used to it, and it starts to feel more natural.

> **Note**
>
> One way to speed things up is to start using **tab** to perform **tab-completion** often this will auto-complete file names! Press tab twice quickly and it will print all possible combinations.

2.9 Summary

While maybe not all that exciting, these things really are the foundation needed to start utilizing the command line – which then gives us the capability to use lots of tools that only work at a command line, manipulate large files rapidly, access and work with remote computers, and more! These are the fundamental tools that every scientist needs to work with **big data**.

2.9.1 Terms

Term	What it is
path	the address system the computer uses to keep track of files and directories
root	where the address system of the computer starts, /
home	where the current user's location starts, ~/
absolute path	an address that starts from a specified location, i.e. root, or home
relative path	an address that starts from wherever we are
tab-completion	our best friend

2.9.2 Commands

Command	What it is
date	prints out information about the current date and time
head	prints out the first lines of a file
pwd	prints out where we are in the computer (print working directory)
ls	lists contents of a directory (list)
cd	change directories

2.9.3 Special characters

Command	What it is
Characters	Meaning
/	the computer's root location
~/	the user's home location
../	specifies a directory one level "above" the current working directory

2.10 Summary

You won't get used to operating in bash, or moving around directories using just the command line in a single session. So if you think you are interested in developing your bioinformatic skills, spend some time practising.

Here is a link to a couple of extended tutorials you can bookmark if you want to explore this further:

https://datacarpentry.org/shell-genomics/01-introduction/index.html

2.11 Stretch yourself - optional extras to try a couple of other skills

2.11.1 Creation

I want to create a new directory to store some code files I'm going to write later, so I'll use `mkdir` to create a new directory called Code:

Check you are in the `unix_intro` folder - Click here for Answer

```
pwd
```

Make a new directory called Code - Click here for Answer

```
mkdir Code
```

Check this folder has been created using a list function

```
ls
```

Note that I used a relative file path to create the Code directory - but I could have also specified an absolute filepath to generate that folder in whatever location I want.

There are a few ways to make new files on the command line. The simplest is to generate a blank file with the `touch` command followed by the path (relative or absolute) to the file you want to create

Make a new file called data-science-class.txt - Click here for Answer

```
touch data-science-class.txt
ls -l
```

Note here I could just use ls to list all files and folders in a directory, but if i set the flag `-l` *then it will produce a* ***l****ong list of files.*

If the entry in the first column is a `d`*, then the row in the table corresponds to a directory, otherwise the information in the row corresponds to a file.*

The string of characters following the `d` *in the case of a directory or following the first* `-` *in the case of a file represent the permissions for that file or directory - I won't cover that here - but some of the links I provide go into more detail.*

We practice what we preach! This site is created with Git and R markdown, using the `bookdown` package. Go ahead and peek behind the scenes.

Long-term, you should understand more about what you are doing. Rote clicking in RStudio may be a short-term survival method but won't work for long.

- trygit is to (command line) Git as swirl is to R. Learn by doing, in small bites.
- The book Pro Git is fantastic and comprehensive.
- Git in Practice by Mike McQuaid is an more approachable book, probably better than Pro Git for most people starting out. Ancillary materials on GitHub.
- Git for Humans is a great set of slides by Alice Bartlett, originally delivered in 2016 at UX Brighton.
- GitHub's own training materials may be helpful. They also point to many other resources
- Find a powerful Git client (chapter **??**) if you'd like to minimize your usage of Git from the command line.
- Hadley Wickham's book R Packages has an excellent chapter on the use of Git, GitHub, and RStudio in R package development. He covers more advanced usage, such as commit best practices, issues, branching, and pull requests.
- Ten Simple Rules for Taking Advantage of Git and GitHub http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004947
- RStudio's guide Version Control with Git and SVN
- The book *Team Geek* has insightful advice for the human and collaborative aspects of version control. It proposes Git strategies suited to different characteristics of teams.

Bibliography

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.21.