Basic command line

Kirstyn Brunker

Contents

Credit: Modified from GECO Viral Bioinformatics training material

1 1: Basic Command Line

There are a number of handy commands and shortcuts that are useful to know before diving into the Linux command line. Let's try some!

1.1 1.1: Tab Autocomplete

When using the command line, often typing is the slowest part. Linux is aware of this and lets you use the tab button to autocomplete for certain filenames or directories.

While typing in a folder/filename, if you type the first few letters and hit the tab button, Linux will attempt to autocomplete the name for you. Let's try the text below, followed by hitting the tab button once:

```
rage_1:~$ cd D
```

The shell has managed to autocomplete some of the name, but not all of it. This likely means there is not enough information to uniquely identify the file. This can be verified by hitting the tab button twice which will produce the following output:

```
rage_1:~$ cd D-
Documents/ Downloads/
```

There are two folders that start with the name "D-" so must manually complete (at least until the text is sufficient to pinpoint the file i.e. in this case `Doc`, then tab, would be enough to find "Documents/")

1.2 1.2: Exiting a program

Sometimes you might want to exit a program if for example it is running for too long, or is stuck in an infinite loop. We can do this with `ctrl+c`.

For example, run the command `top`, which shows a list of the processes currently running on the machine.

```
rage_1:~$ top
```

There is no clear way out! But now try `ctrl+c` to exit.

1.3 1.3 Typing on the command line

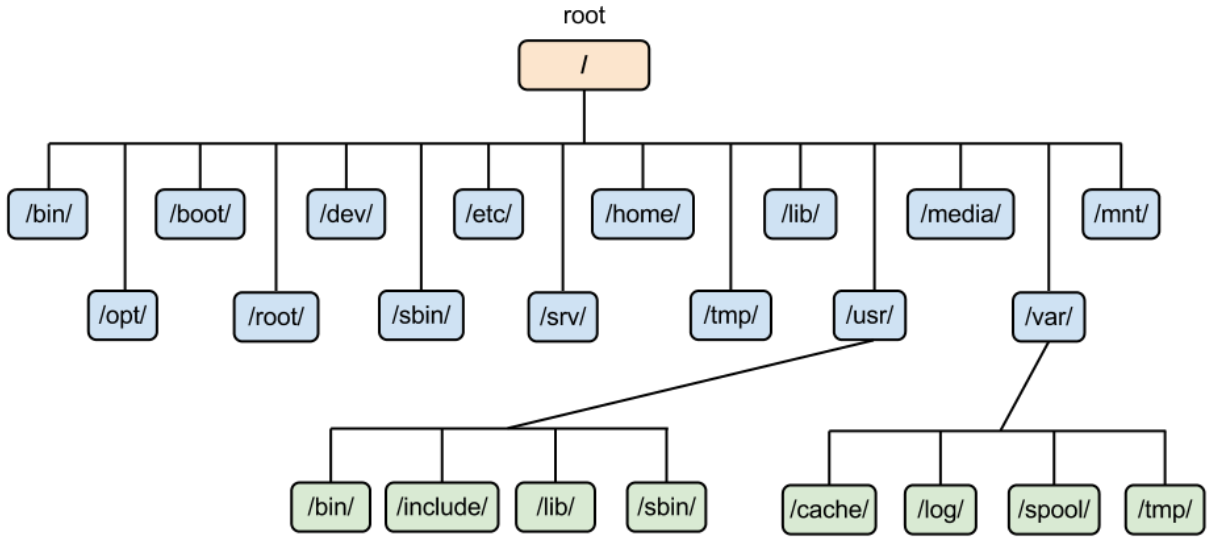
When typing a command, you might want to jump to the front of the command if you have made a typo or want to edit the command. We can do this using `ctrl+a` to move to the front of a command and `ctrl+e` to move to the end.

1.4 1.4 Reuse old commands

Often we want to reuse commands we have already run before. in Linux you can do this using the up and down arrow keys on the keyboard to flick through the most recently used commands. This saves a lot of time typing, especially if you are writing and re-running scripts to check they work correctly.

2 2: Navigation in Linux

The file system of linux is organised as a hierarchy of files and directories(folders). The root directory is the top of the hierarchy with all other files and directories in the operating system located below it.



When you log in to your account on a linux machine, normally you will be placed into a directory named after your username (for example, if my username is kirstyn, my directory will be found at **'/home/kirstyn'**). This is located inside another directory called home, which is contained within the root directory (which is called /). On the RAGE-on-SSD the default username is rage_1.
In order to use Linux via the terminal, it is important to know how to navigate around the operating system using commands.

2.1 2.1: `pwd` (Print Working Directory)

The first thing you might want to know when you enter the command line is where exactly you are in the operating system. The `pwd` is used for exactly that. The command stands for **"print working directory"** and will print the users current location.

```
rage_1:~$ pwd
/home/rage_1
```

As we are in the home directory of the user, the command returns `/home/rage_1`.

2.1.1 Task 1

Find your current directory using pwd.

2.2 2.2: `ls` (List)

The next thing we might want to do is to have a look at what files and directories are in our home directory. We can do this using the `ls` command, which lists the contents of the current working directory (i.e where the user currently is).

```
rage_1:~$ ls
lost+found  shared-public  shared-team
```

The output of `ls` on my machine shows that I have 3 directories called **lost+found**, **shared-public**,and **shared-team**. Yours will produce a different output! The output will vary depending on the contents of your home directory. `ls` can also be used with flags to change the output in various ways. Flags are specified by adding a `-` followed by a character that indicates the flag type. An example of an `ls` flag is the `-l` flag which prints each item in the list on a separate line. We can also use the -h flag to make the output (specifically any output relating to file sizes) more human readable:

```
rage_1:~$ ls -l -h
total 16K
drwxrws--- 2 root   rage_1s 16K Jun 20 20:31 lost+found
lrwxrwxrwx 1 rage_1 users  14 Jun 27 15:58 shared-public -> /shared/public
lrwxrwxrwx 1 rage_1 users  12 Jun 27 15:58 shared-team -> /shared/team
```

We can see that each file and directory is printed on its own line, along with some extra information about its user permissions, last edit dates and file sizes. Again, your output will look slightly different.

Most Linux commands have optional flags and follow a similar pattern of use i.e: `command_name [-flags] [parameters]`. A full example of `ls` may look like `ls -l -h shared-team`, where the output would be a listed print of all files and directories in the shared-team directory in a human readable form.

2.2.1 Task 2

Use `ls` to list the folders in your current directory.

2.2.2 Task 3

Use `ls` to list each item in the directory in a new line.

2.2.3 Task 4

Repeat task 3, but make the file sizes “human readable”.

2.3 2.3: `mkdir` (Make Directory)

The `mkdir` command is used to create a directory in Linux. Lets use it to make an example directory to navigate into. We do this by using `mkdir` followed by the name of the directory we wish to create:

```
rage_1:~$ mkdir example_directory
```

2.3.1 Task 5

Create 2 new directories using `mkdir`

2.3.2 Task 6

List the contents of one of these new directories.

2.4 2.4: `cd` (Change Directory)

Now lets try and navigate around the operating system and explore a bit. To do this we will need to use the `cd` command which allows us to move up or down the file system. At its most basic, `cd` works as follows:

```
rage_1:~$ cd directory_name
```

If the directory name is correct, then the command will move the user to that directory. We can check this has worked using `pwd`.

```
rage_1:~$ cd example_directory/
rage_1:~/example_directory$ pwd
/home/rage_1/example_directory
```

Depending on the distro of Linux you are using, you might notice that the `~` before the `$` has changed to `~/example_directory`. This is actually equivalent to the output of `pwd`, with the exception that `~` is being used to represent `/home/rage_1/`. In Linux systems, `~` is used to represent the users home directory. This means that if we ever want to quickly return to our home directory, we can do so with the following:

```
rage_1:~/example_directory$ cd ~
rage_1:~$ pwd
/home/rage_1/
```

So we can now enter a directory and return home using `~`, how do we move up to a directory above our current directory? In Linux, we can use `..` to indicate we want to move up a level in the hierarchy.

```
rage_1:~$ cd ..
rage_1:/home$ pwd
/home
```

`pwd` shows that we are now in the home directory, which is indeed one directory above the rage_1 directory. We can navigate back down to our home using either `cd ~` or `cd username` (i.e. `cd rage_1`).

While the examples here show moving up and down to folders that are directly above or below our working directory, `cd` can be given all sorts of filepaths.

```
rage_1:~$ cd ../rage_1/example_directory/
rage_1:~/example_directory$ pwd
/home/rage_1/example_directory
```

This example is a bit unnecessary, but to summarise, we moved up one directory using `..`, then back down to our current directory with `/rage_1`, then down again with `/example_directory`. We could do the same thing using either `cd /home/rage_1/example_directory` or `cd ~/example_directory` or `cd example_directory` if we are in the home directory already.

The path `/home/rage_1/example_directory` is actually an example of a **absolute path**, meaning that the path begins at the root of the filesystem and ends at the destination. Absolute paths are handy when we know files are stored at a static location that is unlikely to change (configuration files are a good example of this) but since they require the whole path to be listed, they appear very long. As such, **relative paths** are often used. These identify the location of a file relative to either the current working directory (`cd example_directory` is an example of using a relative path) or relative to some symbol like `~` (again representing the rage_1s home directory) or `.` (representing the directory the rage_1 is currently in, i.e equivalent to the working directory).

2.4.1 Task 6

Navigate to the root directory

2.4.2 Task 7

Navigate from the root directory back to the home directory
