Introduction to Unix/Linux – **Part 3**

Víctor Sojo (vsojo@amnh.org)

Contents

I. Hacking your `.bashrc` or `.zshrc` shell session profile 2

II. Downloading files from the web with `wget` (Linux) or `curl -O` (Mac) 3

III. Compressing and uncompressing files with `gzip`, `gunzip`, and `tar` 3

III.1. Using `gzip` and `gunzip` compression 3

III.2. Using `tar` for packaging and compression 3

III.3. Decompressing tar files 4

IV. Sending files to a remote server using `scp` 4

IV.1. Using a variable to access the remote server 5

V. Matching and extracting text with `grep` 6

V.1. Finding lines that contain simple strings using `grep` 6

V.2. Unlocking the true power of `grep` with regular expressions 7

VI. Modifying text with `sed` 7

VII. Using `paste` to put multiple lines in one line 8

VIII. Using `awk` to extract information from a text file and do operations on it 8

IX. Where to from here 9

IX.1. More about some of the commands we used here 9

IX.2. Some helpful books 9

IX.3. Resources for DIY learning and troubleshooting 9

IX.4. Some handy software 9

I. Hacking your .bashrc or .zshrc shell session profile

Running a customised version of ls (e.g. ls -ltrhA in my case) is so common that most people like to create themselves an **alias**, i.e. some sort of shorthand to speed up the command so that they don't have to type the entire thing every single time.
You could create a temporary alias directly on the command line:

```
alias l='ls -ltrhA'
```

But please don't do this.

The problem is that this will work fine during this active session, but the next time you open a new terminal session, the computer will have forgotten that alias.
You can get the computer to remember it forever by hacking your **profile file**, which is in your home directory (~, or $HOME) and is typically called .bashrc and/or .bash_profile if you're on bash, or .zshrc if you're on zsh.

1. Go to your home directory by issuing a simple cd, without any target.
2. Find out if you're on a bash shell or a zsh shell by typing echo $SHELL
3. Do ls -la to take a look at the hidden files. You should see .bashrc or .zshrc there.
4. Create a backup of this file, just in case you mess things up:

```
cp .zshrc .zshrc.backup.20201028
```

Creating backup files is an extremely good idea, particularly when you're messing with high-level files such as .zshrc/.bashrc… or generally with files you don't understand much.

Let's open the file to edit it. I would typically use vi/vim as my editor of choice, but there's no time to introduce it properly here (just note that it's awesome). Let's just use nano instead:

5. Open the file with nano .zshrc (or nano .bashrc if you're on bash).

You'll probably see a few things there already.

6. At the very top, enter the following lines:

```
# ALIASES
alias l='ls -ltrhA'
alias la='ls -la'
alias ll='ls -l'

# SERVERS
export HUX='yourusername@huxley-master.pcc.amnh.org’
alias hux='ssh $HUX'
```

I'll explain those last two later, just make sure to change yourusername to your actual username.
The upper three should be obvious: we're creating quick pseudo-commands to access versions of the ls command quickly.

7. **If you're on zsh**, please also enter the following in the upper # ALIASES section:

```
alias scp='noglob scp'
```

This is important for Mac users. I'll explain why later. Do not do that if you're on bash.

8. We're done here. Save and exit the file.

Now, if you type l or la you'll get an error. That's because we haven't actually loaded the new profile file, we just changed it. To load it, you could simply **open a new terminal tab (⌘T on Mac)**.
Your profile gets loaded every time you start a terminal session (including new tabs/windows).
Alternatively:

9. Load the newly modified profile file with `source .zshrc` (or `source .bashrc`).

Now you can try issuing `l`, or printing the variable we created with `echo $HUX`.

II. Downloading files from the web with `wget` (Linux) or `curl -O` (Mac)

Note: Some of the following sections are inspired by the marvellous Linux tutorial by Adeel Malik & Muhammad Farjan Sjaugi, which is chapter 2 of the *Bioinformatics Practical Handbook* edited by Lloyd Low, ISBN 9789813144743. You are encouraged to acquire that book—it's very good!

We will download a FASTQ file from the internet, so that we can use it in our further analyses.

10. Open a new terminal session and create a directory **in your local machine** (i.e. don't log in to Huxley) to put the file into:

```
mkdir Linux_class_03
```

Note: It's common practice to name directories using `Starting_upper_snake_case_style`.

11. Go into the directory

```
cd Linux_class_03
```

12. Now let's download the file with `wget` (Linux) or `curl -O` (Mac):

```
wget https://eduversum.org/eDNA_AC_VOI26.fastq.gz
```

Important: `wget` will not work on Mac! You'll need to use `curl -O` instead (that's an upper-case letter 'o', not a zero, which looks like this `0` in the font I'm using):

```
curl -O https://eduversum.org/eDNA_AC_VOI26.fastq.gz
```

This will get a file from the internet and put it in whichever directory you happen to be in. There are ways to get the file to end up somewhere else, but typically, it's best to just go to the place where you want the file to be using `mkdir` and `cd`, as we did above.

13. Check that the download was successful by using `ls -lh`

It's a good habit—and it will inevitably become a reflex—to use `ls` to check that your downloads and commands have worked. The file should be around `19 MB`.

III. Compressing and uncompressing files with `gzip`, `gunzip`, and `tar`

III.1. Using `gzip` and `gunzip` compression

You will have noticed that the file we downloaded has a `.gz` extension. Do remember that **extensions mean absolutely nothing to Unix**, but `.gz` typically denotes a "gzipped" compressed file. You can very easily de-compress it (or "unzip" it):

14. Decompress the file with `gunzip eDNA_AC_VOI26.fastq.gz`

If you run our new nifty `l` alias (or just `ls -l`), you'll see that the .gz extension disappeared and the uncompressed file is much larger at ~`109 MB`. You can easily regenerate the compressed file:

15. Compress the file again with `gzip eDNA_AC_VOI26.fastq.gz`

This will take a couple of seconds; be patient. You'll see that the file is `19 MB` again.

Leave it compressed for now.

III.2. Using `tar` for packaging and compression

Another very common tool to compress files is `tar`, which works roughly as follows:

```
tar -zcvf name_of_the_packaged_file.tar.gz what*.ever you_want/to put/in* the/package
```

Actually, `tar` is a package maker—it bundles bunches of files into a single "tarred" file for easier distribution—so you'll be using it very often to move stuff around or store it for posterity.

The options we used are:

- c : create a package (i.e. I want to make a new package, not extract things out of an existing one, which we'll do later using the x option instead of c).
- z : gzip it (i.e. compress it with gzip).
- v : verbose" (show me everything you're doing. This is optional and you can leave it out for quieter output).
- f : let me specify the name of the packaged file that I will create.

Try to get familiar with tar, you'll be using it quite a bit.
We could go on and recompress the file, as in:

```
tar -zcvf eDNA.tar.gz eDNA_AC_VOI26.fastq.gz
```

But please don't. We will instead compress the directory that holds it.

16. Go up one directory with cd ..
17. Make sure you're in the right place with pwd and ls (or our mighty l alias).

ls should show you the directory we created earlier, which contains the downloaded file.

18. **Package and compress** the entire directory with tar:

```
tar -zcvf class03.tar.gz Linux_class_03/
```

Remember to use TAB to fill in the file names (won't work for the compressed file, since it doesn't exist yet and the computer cannot—yet—read your mind; but it should work for the directory). You should now have a compressed file called class03.tar.gz. But actually, please notice that we also have the original directory here, so:

Unlike, gzip, tar does not destroy the original file when it packages/compresses it.

19. Check the size of the new compressed file with l (or ls -lh).

You'll see that the size is the same as before. We tried to compress the directory with gzip by passing the -z flag to tar. But the directory contains a file that was already compressed by gzip. So, not much happens with the size when we tell gzip to try to compress something that it has already compressed—if anything it could get larger. There is no endless *compressinception* here.

III.3. Decompressing tar files

You could decompress the file by issuing:

```
tar -zxvf class03.tar.gz
```

But I strongly suggest you don't do that just yet in our example. Instead, let's send it to the cluster.

IV. Sending files to a remote server using scp

Note: scp does not work in zsh the same way as in bash. That's why we added that alias above for alias scp='noglob scp'

An extremely important activity that you will probably need to do on a regular basis is sending files between Unix systems, e.g. between your local machine and the remote server. The most typical way to do this is with scp, which works almost exactly like cp, but between your local machine and a server, and in either direction. So, just as cp does it like:

```
cp the_existing.file the/new/place/.
```

scp works almost identically:

```
scp the_local.file yourusername@remote.server:.
```

The dot means “put it in that directory”. By default scp connects directly to your $HOME directory, so :. means the file will end up there. If you want it somewhere else:

```
scp the_local.file yourusername@remote.server:where/in/there/you/want/it/.
```

Of course, the directory where you’re sending the file needs to exist, otherwise you’ll get an error.

Because of that, many people just put their files in $HOME and then move them from there.

It works exactly the same the other way around:

```
scp yourusername@remote.server:wherever/is/the.file .
```

That last dot means “put it here” (i.e. in whichever directory you happen to be in when you run the command). Needless to say, if you want it somewhere else:

```
scp yourusername@remote.server:wherever/is/the.file ~/where/you/want/it/.
```

Where ~/where/you/… means “from my $HOME directory, go up to where/you/…

Just like with cp, you can also rename the file as you copy it:

```
scp yourusername@remote.server:remote_name.txt local_new_name.txt
```

IV.1. Using a variable to access the remote server

You probably realise that it’s a pain to have to write the name of the server every single time you want to copy something to/from there. That’s why above we created a variable in our .bashrc or .zshrc profile.

20. Take a look at the server name variable: echo $HUX

You should see yourusername@huxley-master.pcc.amnh.org if you’ve been following along.

So instead of having to type the entire thing, we can just use that variable:

```
scp $HUX:where/it/is/remote.file .
```

Nice and neat!

OK. With all that knowledge…

21. Let’s send to the cluster our compressed and packaged directory containing the file we downloaded before:

```
scp class03.tar.gz $HUX:.
```

You will have to be connected to the VPN and will be asked for your password. You will also potentially have to go through two-factor authentication. Not fun, but it is what it is. (actually, there are sneaky ways to avoid this. Look up ssh-keygen, but you didn’t hear this from me)

This will put the file in your home directory on Huxley. If you want to do the whole long version instead, it would look something like this:

```
scp class03.tar.gz yourusername@huxley-master.pcc.amnh.org:.
```

…where of course you have to change yourusername to your own, and potentially also the remote server’s name if you’re using something other than the AMNH’s Huxley.

Good, the file is gone. Now let’s go there and see if it really is there.

We *could* do it old style:

```
ssh yourusername@huxley-master.pcc.amnh.org
```

…but that’s a mouthful. That’s why we created that neat alias to get us there quickly. You can ask the computer to remind you of it:

22. Show what the alias does with which hux

The computer will tell you that `hux` has been aliased to mean `ssh $HUX`, which is exactly what we want. So, for those of you who want to take this neat short way:

23. Go to the remote server by simply entering `hux` in the command line

This will call the alias, which itself calls the variable that we defined to contain the server name with our username attached.

24. You should now be in the remote server. Check that the file was properly sent there by doing `l`

…

Ha! Gotcha! `l` doesn't work here because we've only put the alias in our **local** `.zshrc`/`.bashrc` file, not in the **remote** server's one. Do that if you will. We will move on now.

25. In the remote server, decompress the file with `tar -zxvf class03.tar.gz`

You should see a new directory, inside which will be our file.

26. Go into the directory with `cd Linux_class_03`

You'll see that the file is still gzipped, so let's

27. Decompress the file with `gunzip eDNA_AC_VOI26.fastq`

We now have a `.fastq` file. Let's explore it:

28. Do `head -16 eDNA_AC_VOI26.fastq` to show the first 16 lines

You'll see that there's a pattern repeating every 4 lines. If you don't know what these are:

1. The first line has a unique identifier for each sequence read, followed by a bunch of other things.
2. The second line has the actual sequence.
3. Don't ask me why the hell there's a `+` in the third line, nobody seems to know for sure other than "it's always been there".
4. And the fourth line is the "quality score" of the read, which depends on which machine was used to acquire the read.

Don't worry too much about the molecular genetics or bioinformatics of any of this right now—it's a good file to hack text in any case.

V. Matching and extracting text with `grep`

We can use `grep` to extract the lines of a file that match a desired pattern:

```
grep "regex" the_file_in_which_I_am_search.ing
```

Where `regex` is a "**regular expression**". No time to cover those here, but they are one of the most powerful things invented by humankind. Consider yourself emphatically encouraged to learn about them if you will be a bioinformatician of any sort.

V.1. Finding lines that contain simple strings using `grep`

The simplest regular expression is just simple text. Actually, I always wanted to know whether the string "GATTACA" from that cool 1997 film (watch it if you haven't!) appears in my sequences. Let's check:

29. Find all lines that contain a desired string:

```
grep GATTACA eDNA_AC_VOI26.fastq
```

So, you don't really need the quotes if you're only looking for a simple string.

I got a bunch of lines that actually do have the string. How many?

30. Count the number of lines that contain a desired string with `grep -c`

```
grep -c GATTACA eDNA_AC_VOI26.fastq
```

And which lines are those?

31. Show the line numbers for all lines that contain a desired string with `grep -n`

```
grep -n GATTACA eDNA_AC_VOI26.fastq
```

V.2. Unlocking the true power of `grep` with regular expressions

`grep` is far more powerful than a simple string finder.

I now want to find all lines that contain "GATTACA", but *only* if the line starts with at least two As, which are followed immediately by something that is not a T, then come any number of characters followed by the GATTACA, then any other characters, and then the line ends in either a T or a G.

32. Learn and use regular expressions to get seriously advanced pattern matching:

```
grep -E "^A{2}[^T].*GATTACA.*[GT]$" eDNA_AC_VOI26.fastq
```

Don't forget the quotes; we do need them this time.

The `-E` is to call the "extended", more powerful version of `grep` (you could also call `egrep` directly, which is a shorthand for the same thing).

As you can surely guess, this gets extremely powerful really quickly. We're not even scratching the surface here. The message bears repeating a third time:

Learn as much as you can about **regular expressions**!

They are not exclusively a Unix thing. They are implemented in every language you're ever likely to come across (Python, C, Perl, Java, R, JavaScript, MatLab, Julia, Ruby… and many more), as well as in text editors such as BBEdit (Mac), Notepad++ (Win), or vim.

VI. Modifying text with `sed`

If you take a look at all the #1 lines of the file, you'll see that they all start with "@J00113:238:HFWCLBBXX:4:". That's not informative (we can keep that info elsewhere), so let's just replace most of that long string with a single letter just for reference:

33. Pipe text to `sed` to make replacements.

```
cat eDNA_AC_VOI26.fastq | sed 's/J00113:238:HFWCLBBXX:4/J/' >
                                         eDNA_AC_VOI26.cleaned.fastq
```

Make sure the text above is all on the same line. I just couldn't fit it in this document.

The first `s` before the `/` means "substitute". The two blocks separated by the `/`s are what you're substituting, and what with.

Note that we are creating a new file with `>` so that we don't alter the old one. Be warned that sometimes that's not feasible or convenient in bioinformatics (namely when you're working with gigantic data, which will probably be often). If you're changing a source file, make sure that you really want to do that.

34. Run `head` again to take a look at the changes.

Good!

Actually, I don't like that second block of text on the identifier line, the one that starts `2:N`…; let's get rid of it. But this time, we can do the replacement directly on our new file:

35. Use `sed -i` to replace text directly within a file:

```
sed -i 's/ 2:N.*//' eDNA_AC_VOI26.cleaned.fastq
```

Note the space after the first `/` and that we're using the new `cleaned` file, not the original one! Here we're taking all that text e.g. "` 2:N:0:NGATCAGT+NGAGGATA`" and replacing it with nothing. You'll see that we're using regular expressions again (the `.*` means any characters following). Did I mention how great and powerful RegExes are?

On Mac, for some silly reason, you need to add an extra `''` (empty singly quotes) after the `-i`:

```
sed -i '' 's/ 2:N.*//' eDNA_AC_VOI26.cleaned.fastq
```

Cool. Now that we're at it, I also don't like the colons `:`, I want to change them to underscores `_`

36. Try with `sed -i 's/:/_/' eDNA_AC_VOI26.cleaned.fastq`

(remember to add the empty `''` after the `-i` if you're on Mac).

That worked, but only partly. You'll see that only the first `:` got replaced. To replace them all, you need to…

37. Use the greedy version of `sed` to make multiple replacements per line.

```
sed -i 's/:/_/g' eDNA_AC_VOI26.cleaned.fastq
```

Nice, so now you know that `sed` will do only one replacement per line unless you tell it to be greedy. There's of course a lot more to `sed`, but we'll stop here.

Because sed can so drastically (and irreversibly) change a file, it is advisable to run preliminary tests with `head my_file.txt | sed 's/change this/to that/'` before proceeding to perform the change on the entire file with `sed -i`.

VII. Using paste to put multiple lines in one line

FASTQ files have 4 lines for each entry. It would be cool to have all that info in one line so that the information is a bit more manageable. We can do that easily with `paste`:

38. Use paste to put multiple lines in one line side by side:

```
cat eDNA_AC_VOI26.cleaned.fastq | paste - - - - > eDNA_AC_VOI26.tsv
```

…where the four dashes mean "put four lines side by side".

VIII. Using awk to extract information from a text file and do operations on it

Let's turn that tab separated file into a good old FASTA file using awk.

39. Do the following:

```
head eDNA_AC_VOI26.tsv | awk '{print $1 "\t" $2}'
```

Pretty neat, aye?

40. OK, let's turn the entire thing into (almost) a FASTA file:

```
cat eDNA_AC_VOI26.tsv | awk '{print $1 "\n" $2}' > eDNA_AC_VOI26.fasta
```

Look at the head. We're nearly there. All we need to do is…

41. Change that `@` to a `>`

```
sed -i 's/@/>/g' eDNA_AC_VOI26.fasta
```

Et voilà, we have a FASTA file!

`awk` is way way way more powerful than this. It can do mathematical operations such as calculating sums and averages by column, and a lot more, but we don't have time to go over it here.

IX. Where to from here

(with thanks to Dean Bobo and Sara Oppenheim for suggestions)

IX.1. More about some of the commands we used here

awk

- https://linuxhandbook.com/awk-command-tutorial/
- https://www.tutorialspoint.com/awk/awk_quick_guide.htm

grep

- https://www.gnu.org/software/grep/manual/
- https://www.linuxteacher.com/grep-command-in-unix-linux-with-examples/

sed

- https://en.wikipedia.org/wiki/Sed

IX.2. Some helpful books

- https://www.amazon.com/Bioinformatics-Python-Cookbook-bioinformatics-computational/dp/1789344697/
- https://www.amazon.com/Bioinformatics-Practical-Generation-Sequencing-Applications/dp/9813144742/
- https://www.amazon.com/Practical-Computing-Biologists-Steven-Haddock/dp/0878933913/

IX.3. Resources for DIY learning and troubleshooting

- https://www.linux.org/forums/linux-beginner-tutorials.123/
- https://www.coursera.org/learn/unix
- https://stackoverflow.com
- https://unix.stackexchange.com/
- https://www.biostars.org/
- https://www.biostarhandbook.com/index.html (paid!)

IX.4. Some handy software

(at your own risk)

iTerm2 (https://www.iterm2.com/), a feature-full and lovely replacement for the native Mac terminal).

Filezilla (https://filezilla-project.org/) a free and easy to use FTP utility allowing you to transfer files from a local computer to a remote computer, and vice versa.

Sublime Text (http://www.sublimetext.com/3) a program for working with text files. It has built-in utilities for editing bash, python, and other scripting languages.

BBEdit (https://www.barebones.com/products/bbedit) a text editor for the Mac.

…or:

Notepad++ (https://notepad-plus-plus.org/) the equivalent for Windows.

Both are great!

EasyFind (https://www.macupdate.com/app/mac/11076/easyfind) is a smarter and cuter version of the Mac Spotlight search.