Introduction to the Unix Shell for biologists

The source code can be found at:

https://github.com/konrad/Introduction_to_the_Unix_Shell_for_biologists/

Motivation and background

In this course you will learn the basics of how to use the Unix Shell. Unix is a class of operating systems with many different flavors including well known ones like GNU/Linux and the BSDs. The development of Unix and its shell (also known as command line interface) dates back to the late 1960s. Still, their concepts lead to very powerful tools. In the command line you can easily combine different tools into pipelines, avoid repetitive work and make your workflow reproducible. Knowing how to use the shell will also enable you to run programs that are only developed for this environment which is the case for many bioinformatical tools.

Work environment and test files

During this course all of you are working on Ubuntu (version 14.04) which is a widely used GNU/Linux distribution. The systems boots from a USB stick which offers you to run a live system or to install Ubuntu on your computer. We will run the live mode which will not change the system installed on your PC. After shutting the live system down and removing the stick everything on the computer will be as before.

To get test data click on the `Dash` button on the top left of your screen, type `terminal` and click on the Terminal icon. You will learn later what you are doing but for the moment just type the following commands into the command line interface. Do not write the dollar sign(`$`). It just indicates the so called prompt:

```
$ wget http://data.imib-zinf.net/unix_course_files.tar.gz
$ tar xfz unix_course_files.tar.gz
$ rm xfz unix_course_files.tar.gz
```

The basic anatomy of a command line call

Running a tool in the command line interface follows a simple pattern. At first you have to write the name of the command (if it is not globally installed it's

precise location needs to be give - we will get to this later). Some programs additionally require parameters and arguments. Parameters usually start with a dash (`-`). The common pattern looks like this (`<>` indicated obligatory items, `[]` indicated option items):

```
<program name> [parameters] [arguments]
```

An example is calling the program `ls` which list the content of a directory. You can simply call it without any parameter

```
$ ls
```

or with one or more parameters

```
$ ls -l
$ ls -lh
```

or with one or more arguments

```
$ ls test_folder
```

or with one or more parameters and arguments

$ ls -l test_folder

The results of a command is written usually to the so called *standard output* of the shell which is the screen shown to you. We will later learn to redirect this e.g. to the *standard input* of another program.

How to get help and documentation

Especially in the beginning you will have a lot of questions what a command does and which arguments and parameters need to be given. One rule before using a command or before asking somebody about it is called RTFM (please check the meaning yourself). Maybe the most important command is `man` which stands for *manual*. Most commands offer a manual and with `man` you can read those. To get the documentation of `ls` type

```
$ man ls
```

Additionally or alternatively many tools offer some help via the parameter `-h`, `-help` or `--help`. For example `ls`:

$ ls –help

Bash keyboard shortcuts

There are different implementations of the Unix Shell. You are currently working with Bash. Bash has several keyboard shortcuts that improve the interaction. Here is a small selections of some of them:

- Ctrl-a - Jump to the beginning of a line
- Ctrl-e - Jump to the end of a line
- Ctrl-u - Remove everything before the cursor position
- Ctrl-k - Remove everything after the cursor position
- Ctrl-l - Clean the screen
- Ctrl-r - Search in command history
- Tab - extend the commands and file/folder names

Files, folder, location

Topics:

- `ls`
- `pwd`
- `cd`
- `mkdir`
- Relative vs. absolute path
- ~/

In this part you will learn how to navigate through the file system, explore the content of folders and create folder.

At first we need to know where we are. If you open a new terminal you should be in your home directory (we explain will this below). To test this call the program `pwd` which stands for *present work directory*.

```
$ pwd
/home/ubuntu
```

The default user of the Ubuntu live system is called `ubuntu`. In general each user has a folder with its user name which is a subfolder of the folder `home`. The next command we need and which has been already mentioned above is `ls`. It simply lists the content of a folder. If you call it without any argument is will show the content of the current folder. Using `ls` we want to get a rough overview how a common Unix file system tree looks like and learn how to address files and folders. The root folder of a systems starts with `/`. Call

```
$ ls /
```

To see the file in the root folder. You should see something like

```
bin   data  etc  lib    lost+found  mnt  proc  run   srv  tmp  var
boot  dev   home lib64  media   opt  root  sbin  sys  usr
```

There are several subfolders in the so called root folder (and yes, to make it a little bit confusing there is even a folder called "root" in the so called root folder). Those are more important if you are the administrator of the system and you do not have the permission to do do changes there. Currently your home directory is your little universe in which you can do whatever you want to do. In here we will learn how work with the paths. A file or folder can be addressed either with is *absolute* or its *relative path*. As you have downloaded a collection of test files and folder you should have a folder `unix_course_files` in you home folder. Assuming you are in you home folder `/home/ubuntu/` the relative path to the folder is simply `unix_course_files`. You can get the content of the folder listed by calling `ls` like this:

```
$ ls unix_course_files
```

This is the so called relative path as is relative to the current work directory `/home/ubuntu/`. The absolute path would start with a `/` and is `/home/ubuntu/unix_course_files`. Call `ls` like this:

```
$ ls /home/ubuntu/unix_course_files
```

There are some conventions regarding relative and absolute paths. One is that a dot (`.`) represents the current folder. The command

```
$ ls ./
```

should return the same as simply calling

```
$ ls
```

Two dots (`..`) mean the parent folder. If you call

```
$ ls ../
```

you should see the content of `/home`. If you call

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

You should see content of the parent folder of the parent folder which is the root folder (`/`) assuming you are in `/home/ubuntu/`. Another convention is that `~/` represents the home directory of the user. The command

```
$ ls ~/
```

should list the content of your home directory independent of your current location in the file system.

Now as we know where we are and what is there we can start to change our location. For this we use the command `cd` (change directory). If you are in you home directory `/home/ubuntu/` you can go into the folder `unix_course_files` by typing

```
$ cd unix_course_files
```

After that call `pwd` to make sure you that you are in the correct folder.

```
$ pwd
/home/ubuntu/unix_course_files
```

To go back into your home directory you have different options. Use the absolute path

```
$ cd /home/ubuntu/
```

or the above mentioned convention for the home direction "~/":

```
$ cd ~/
```

or the relative path in this case the parent directory of /home/ubuntu/unix_course_files:

```
$ cd ../
```

As the home directory is such an important place `cd` uses this as default argument. This mean if you call `cd` without argument you will go to the home directory. Test this behavior by calling

```
$ cd
```

Try now to go to different locations in the file system and list the files and folder located there.

Now we will create our first folder using the command `mkdir` (*make directory*). Go into the home directory and type:

```
$ mkdir my_first_folder
```

Here we can discuss the implementation of another Unix philosophy: "No news is good news." The command successfully created the folder `my_first_folder`. You can check this by calling `ls`. But `mkdir` did not tell you this. If you do not get a message this usually means everything went fine. If you call the above `mkdir` command again you should get an error message like this:

```
$ mkdir my_first_folder
mkdir: cannot create directory ‘my_first_folder’: File exists
```

So if a command does not complain you can usually assume there was no error.

Manipulating files and folder

Topics:

- `touch`
- `cp`
- `mv`
- `rm`

Next we want to manipulate files and folders. We create some dummy files using `touch` which is usually used to change the time stamp of files. But you can also create empty files with it easily. Let's create a file called `touch test_file_1.txt`:

```
$ touch test_file_1.txt
```

Use `ls` to check that it was created.

The command `cp` (copy) can be used to copy files. For this it requires at least two arguments: the source and the target file. In the following example we generate a copy of the file `test_file_1.txt` called `a_copy_of_test_file.txt`.

```
$ cp test_file_1.txt a_copy_of_test_file.txt
```

Use `ls` to confirm that this worked. We can also copy the file in the folder `my_first_folder` which we have created above:

```
$ cp test_file_1.txt my_first_folder
```

Now there should be also a file `test_file_1.txt` in the folder `my_first_folder`. If you want to copy a folder and its content you have to use the parameter `-r`

```
$ cp -r my_first_folder a_copy_of_my_first_folder
```

You can use the command `mv` (move) to rename and to relocated files or folders. To rename the file `a_copy_of_test_file.txt` to `test_file_with_new_name.txt` call

```
$ mv a_copy_of_test_file.txt test_file_with_new_name.txt
```

With `mv` you can also move a file into a folder. For this the second argument has to be a folder. For example, to move the file now named `test_file_with_new_name.txt` into the folder `my_first_folder` use

```
$ mv test_file_with_new_name.txt my_first_folder
```

You are not limited to one file if you want to move them into a folder. Let's create and move two files `file1` and `file2` into the folder `my_first_folder`.

```
$ touch file1 file2
$ mv file1 file2 my_first_folder
```

At this point we can introduce another handy feature most shells offer which is called *globbing*. Let us assume you want to apply the same command to several files. Instead of explicitly writing all the file names you can you a globbing pattern to address them. There are different wildcards that can be used for those patterns. The most important one is the asterisk (`*`). It replaces one or more characters. Let us explore this with asmall example:

```
$ touch file1.txt file2.txt file3
$ ls *txt
$ mv *txt my_first_folder
```

The `ls` will show the two files that are matching the given pattern (i.e. `file1.txt` and `file2.txt`) while dismissing the one not matching (i.e. `file3`). Same for `mv` - it will only move the two files ending with `txt`.

We accumulated several test files that we do not need. Time to clean up a little bit. With the command `rm` (*remove*) you can delete files and folders. Please be aware that there is no such a thing as trash bin if you remove items this way. They will be gone.

To delete a file in the `my_first_folder` call:

```
$ rm my_first_folder/file1
```

To remove the a folder use the parameter "-r" (*recursive*):

```
$ rm -r my_first_folder
```

File content

Topics:

- `less` / `more`
- `cat`
- `echo`
- `head`
- `tail`
- `cut`

Until now we did not care about the content of the files. This will change now. Please go into the folder `unix_course_files`:

```
$ cd unix_course_files
```

There should be some files waiting for you. To read the content with the possibility to scroll around we need a pager program. Most Unix systems offer the programs `more` and `less`. We will use the later one here. Let's open the file "origin_of_species.txt"

```
$ less origin_of_species.txt
```

The file contains Charles Darwin *Origin of species*. You can scroll up and down line wise using arrow key or page wise using the page-up/page-down keys. To quit use the key `q`. The pager programs represent an interactive program but sometimes you just want to have the content of a file given to you. The command `cat` (*concatenate*) does that for one or more files. Let us use it to see what is in the example file `two_lines.txt`. Assuming you are in the folder `unix_course_files/` you can call

```
$ cat two_lines.txt
```

The content of the file is shown to you. You can apply the command to two file and the content is concatenated and returned:

```
$ cat two_lines.txt three_lines.txt
```

This is a good time to introduce the *standard input* and *standard output*. Above I wrote the output it given to you. This means it is written to the so called *standard output*. You can redirect the *standard output* into a file by using `>`. Let us use the call above to generate a new file that contains the combined content of both files:

```
$ cat two_lines.txt three_lines.txt > five_lines.txt
```

Please have a look at the content of this file:

```
$ cat five_lines.txt
```

The *standard output* can also be redirected to other tools as *standard input*. More about his below. With `cat` we can reuse the file content that already exist. To create something new we use the command `echo` which writes a given string to the standard output.

```
$ echo "Something very creative"
```

Do redirect it into a file use `>` a target file.

```
$ echo "Something very creative." > creative.txt
```

Be aware that this can be dangerous. You can overwrite content in a file. For examples if you call now

```
$ echo "Something very uncreative." > creative.txt
```

There will be only the latest string written to the file and the previous one will be gone. To append the output of command to a file without overwriting the content use “>>”

$ echo “Something very creative.” > creative.txt $ echo “Something very uncreative.” >> creative.txt

Now `creative.txt` should contain two lines.

Sometime you just want to have small overview of a file for examples you just want to get the first of last lines of it. For this the commands `head` and `tail` can be used. Per default 10 lines are shown. You can used the parameter `-n <NUMBER>` (e.g. `-n 20` or just `-<NUMBER>` (e.g. `-20`) to modify the number lines to be displayed. Test the tools with the file `origin_of_species.txt`:

```
$ head origin_of_species.txt
$ tail origin_of_species.txt
```

But you cannot only select vertically but also horizontally using the command `cut`. Let us extract only the first 10 character of each line in the file `origin_of_species.txt`:

```
$ cut -c 1-10
```

The tool `cut` can be very useful to extract certain columns form CSV (comma/character separated file). Have a look at the content of the file `genes.csv`. You see that is has different columns that are tabular separated. You can extract selected column with `cut`:

```
$ cut -f 1,4 genes.csv
```

Working with the file content

Topics:

- `wc`
- `sort`
- `uniq`
- `grep`
- `cut`
- `tr`

There are several tools that let you manipulate the content of a plain text file or return information about it. If you want for example some statistics about the number of character, words and lines use the command `wc`. Let us count the number of lines in the file `origin_of_species.txt`:

```
$ wc -l unix_course_files/origin_of_species.txt
```

You can use the command `sort` to sort a file alpha-numerically. Test the following calls

```
$ sort unsorted_numbers.txt
$ sort -n unsorted_numbers.txt
$ sort -rn unsorted_numbers.txt
```

and try understand the output of them.

The tool `uniq` take a sorted list of lines and remove line-wise the redundancy. Please have a look at the content of the file `redundant.txt`. Then use uniq to generate a non-redundant list:

```
$ uniq redundant.txt
```

If you call uniq with `-c` you the then number of occurrence for each remaining entry:

```
$ uniq -c redundant.txt
```

With the tool `grep` you can extract lines that match a given pattern. For instance, if you want to find all lines in `origin_of_species.txt` that contain the word `species` call

```
$ grep species origin_of_species.txt
```

As you can see we only get the lines that contain `species` but not the one that contain `Species`. To make the search case-insensitive use the parameter `-i`.

```
$ grep -i species origin_of_species.txt
```

If you are only interested in the number of lines that match pattern use `-c`:

```
$ grep -ic species origin_of_species.txt
```

The program `tr` (translate) exchanges one character by another. It reads from the standard input and perform the replacement. To direct the content of a file as standard input into a program `<` is used. Have quick look at the content of the file `DNA.txt`.

$ cat DNA.txt

We now want to replace the `T`s there with `U`s. For this we call:

$ tr T U < unix_course_files/DNA.txt

Connecting tools

Another piece of the Unix philosophy is to build small tools that do one thing optimally and use the standard input and standard output. The real power of Unix builds on the capability to easily connect tools. For this so called *pipes* are used. To use the standard output of one file as standard input of another tool the vertical bar `|` is used. For example, extract the first 1000 lines from `origin_of_species.txt`, search for lines that contain `species`, then search in those lines that contain `wild` and finally replace the `ws` by `m` call:

```
$ head -n 1000 origin_of_species.txt | grep species \
  | grep wild | tr "w" "m"
```

Examples analysis

Equipped with fine selection of useful programs and basic understanding how to combine them, we will no apply them perform some analysis of real biological data.

Retrieving data

You have used the tool `wget` above to get the example files. It is very useful especially if you want to retrieve large data sets. We download the fasta file of *Salmonella* Thyphimuirum SL1344's chromosome by calling (in this document the URL is split in three parts it. Please write in one line in the shell and remove the \)

```
wget ftp://ftp.ncbi.nih.gov/genomes/Bacteria/\
     Salmonella_enterica_serovar_Typhimurium_SL1344_uid86645/\
     NC_016810.fna
```

Additionally we download the annotation in GFF format of the same replicon:

```
wget ftp://ftp.ncbi.nih.gov/genomes/Bacteria/\
     Salmonella_enterica_serovar_Typhimurium_SL1344_uid86645/\
     NC_016810.gff
```

Counting the number of features

Use `less` to have a look at `NC_016810.gff`. It is a tabular separated file. The first 5 lines start with `#` and are called header. Then several lines with 9 columns follow. The third column contains the type of the entry (gene, CDS, tRNA, rRNA, etc). If we want to know the numbers of tRNA entries we could try to apply grep and use `-c` to count the number of hit line.

```
$ grep -c tRNA NC_016810.gff
```

This leads to a suspiciously large number. The issue is that the string `tRNA` also occurs in the attribution column (the 9th column). We just want matches in the third column. We can combine `cut` and `grep` to achieve this.

```
$ cut -f 3 NC_016810.gff | grep -c tRNA
```

To get the number of entries for all other feature we could just replace the `tRNA`. But we can also get the number for all of them at once:

```
$ grep -v "#" NC_016810.gff | cut -f 3 | sort | uniq -c
```

Try to understand what we did here. You can use a similar call to count the number genes on the plus and minus strand:

```
$ cut -f 3,7 NC_016810.gff | grep gene | sort | uniq -c
```

Calculate the GC content of a genome

Let us assume the GC content of the genome is not known to us. We can use a handful of command to calculate this quickly. We can gain the number of nucleotides in the following manner.

```
grep -v ">" NC_016810.fna | grep -o "A" | wc -l

grep -v ">" NC_016810.fna | grep -o "C" | wc -l

grep -v ">" NC_016810.fna | grep -o "G" | wc -l

grep -v ">" NC_016810.fna | grep -o "T" | wc -l
```

As we are only need to get the sum of As and Ts as well as Cs and Gs we can used and extended pattern for grep. The `|` means *or*:

```
grep -v ">" NC_016810.fna | grep -Eo "A|T" | wc -l

grep -v ">" NC_016810.fna | grep -Eo "C|G" | wc -l
```

Once we have the number we can calculate the GC content by piping a formula into the calculator `bc`.

```
echo "scale=5; 2332503/(2332503+2545509)*100" | bc
```

Multiple alignment with `muscle`

We cannot only work with the default tools of the Unix shell but now have access to a plethora of command line tools. Let's assume we want to perform a multiple alignment of the members of the GlmZ family.

We choose `muscle` for this purpose. It's website offers compiled binaries which means we only have download

```
$ wget http://www.drive5.com/muscle/downloads3.8.31/muscle3.8.31_i86linux64.tar.gz
```

and extract a file:

```
$ tar xfz muscle3.8.31_i86linux64.tar.gz
```

As we might need this tool more often (this is purely hypothetical as once you shutdown the live system any data will be gone) we generate a folder `bin` in our home directory. This is by convention a place were those programs are stored.

```
$ mkdir bin
```

Then we move the tool into the folder and rename it:

```
$ mv muscle3.8.31_i86linux64 bin/muscle
```

And clean up a little bit:

```
$ rm muscle3.8.31_i86linux64.tar.gz
```

Now we download the sequences of the RNAs which we want to align (again, please write the URL in one line and remove the \).

```
$ wget -O RF00083.fa "http://rfam.xfam.org/family/RF00083/\
  alignment?acc=RF00083&format=fastau&download=1"
```

Have a look at the content of the file using `less` or `cat`.

If you call `muscle` without anything you will get a list of parameters.

```
$ ./bin/muscle
```

Please be aware that we have to give the path to `muscle`.

We want to specify an input file using (`-in`) and an output file (`-out`):

```
./bin/muscle -in  RF00083.fa -out RF00083_aligned.fa
```

This should take just a moment. Now we have the alignments stored in `RF00083_aligned.fa`.

Very, very basic scripting

One huge advantage of the Unix shell is that you can script actions. For example you can write the command for the multiple alignment into a file e.g. using `echo`

```
$ echo "./bin/muscle -in  RF00083.fa -out RF00083_aligned.fa" \
   > run_me.sh
```

If you want to run the command in that script you can call the script in the following manner:

```
$ bash run_me.sh
```

Shell scripting offers very powerful options to program workflow. Due to time restriction we will not cover this here.

What's next

He we just covered a small selection of tools and possibilities and that you can extend you Unix skills based on this knowledge yourself. Some tool we have not cover but could be important are archiving and compression tools like `tar`, `bzip2` and gzip. For more powerful text manipulation `sed` and `awk` are good choices. We also recommend to get familiar with text editors which can be used to interactively modify text files. Classic Unix environment editor are vi (and derivatives like vim) or Emacs. While they are very powerful they have a steep learning curve. For beginner `gedit` that offers a graphical user interface could be another option.