SYSB 3036 W01: Introduction to Unix

Version 2

Frank Aylward[1]

[1]Virginia Tech

dx.doi.org/10.17504/protocols.io.xhffj3n

Frank Aylward
Virginia Tech

PROTOCOL STATUS

Working
We use this protocol in our group and it is working

start

1 A Unix or Unix-like operating system generally has a wide variety of build-in functions that are extremely useful for navigating between folders, exploring the contents of files, and getting various summary statistics that are useful before undertaking a bioinformatic analysis.

Here we will explore some of the more useful Unix commands that you will find useful throughout this course and future assignments. Make sure you get comfortable using these commands, since they will make your life a lot easier later on.

If you are ever curious about how to use a command, you can type "man" right before the command name and hit enter ("man" is short for "manual" in this case, so it will give you the command's manual).

mkdir

2 **mkdir**

The "mkdir" command will make an empty directory (folder) with a specified name. Often we may wish to organize our work into separate folders, so it's nice to be able to easily create new ones.
Let's start by creating a folder called "test_project"

```
mkdir test_project
```

When you make folders in a Unix OS it is always a good idea to keep the names simple and avoid special characters like *#%^$ etc. Special characters can sometimes cause errors when we begin to parse through folder names with python or R scripts. Also, whatever you do, DO NOT PUT SPACES IN YOUR FOLDER NAMES, as these cause lots of issues. If you need some kind of space for readability you can use an underscore instead.

ls

3 **ls**

Now that we've created a new folder, it would be nice to know that it exists.
To do this we can use the "ls" command, which will list all folders and files present in our current folder.

```
ls
```

or to get a list-like output with human-readable values

```
ls -lh
```

You should see the folder "test_project" there.
It's generally a good idea to run "ls" after every step of a tutorial just to see what new files may have been created.

If you only want particular files listed, you can use a wildcard notation to select for them.

```
ls *.txt
```

Will list only ".txt" files, for example.

autocomplete in the command line

4 One nice thing about the command line is that it will try to autocomplete the name of any file or folder you type in if you hit the "tab" button.

So, for example, if you type "ls test_" and then hit a tab, it should autocomplete to "ls test_project".

This is handy since it is often tedious to always type exact file names in the command line, especially since even one typo will results in an error message.

If there is any ambiguity about how to autocomplete, the command line will autocomplete up to the first ambiguity. So, for example, if you had two folders called "test_project_1" and "test_project_2", you could type "te" and then hit tab, and the command line would autocomplete to "test_project_". Try creating another folder called "test_project_2" and giving this a try.

5 **cd and pwd**

Now let's say we want to move into the folder test_project so we can do something there. To navigate between folders we can use the "cd" command.

```
cd test_project
```

and to move back out:

```
cd ..
```

Or to move back to our home folder immediately, regardless of where on the computer you are, just type

```
cd
```

The text before your cursor in the command line should tell you the location of the folder in which you are currently located (aka, your PATH). However, sometimes full PATHs can get long and may be truncated. If ever want the command line to print out what your full PATH is, use "pwd".

```
pwd
```

wget

6 **wget**

One command that is very useful is the "wget" command. This is a standard Unix command that will let you download a file directly to whatever folder your command line is situated in. There are many options, but all that is required is the full URL and an internet connection.

This is useful if you want to download genome files from a public server such as the National Center for Biotechnology Information (NCBI). On NCBI the URL for the genome of the bacterium Staphylococcus aureus is:

```
ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/009/585/GCF_000009585.1_ASM958v1/GCF_000009585.1_ASM958v1_genomic.fna.gz
```

So if I want to download this file all I need to type is:

```
wget
ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/009/585/GCF_000009585.1_ASM958v1/GCF_000009585.1_ASM958v1_genomic.fna.gz
```

gunzip

7 gunzip

You will note that the file we just downloaded had the strange ending ".gz". That is because this is a compressed gzip file that we will need to uncompress in order to inspect further.
Do this this we can use the "gunzip" command:

```
gunzip GCF_000009585.1_ASM958v1_genomic.fna
```

And if you run "ls" now you should see the same file without the .gz ending.
The new file has a ".fna" ending, which stands for FASTA Nucleic Acid. This contains the raw genomic information for the Yersinia pestis genome.

head and tail

8 **head, tail,** and **more**

Sometimes files can be too large and unweildy to open up directly, but we will still want to take a look inside to see what the file format looks like. We can use the "head" and "tail" commands to quickly look at the first and last few lines of the file, respectively.

```
head GCF_000009585.1_ASM958v1_genomic.fna
```

and

```
tail GCF_000009585.1_ASM958v1_genomic.fna
```

Based on this you can get an idea of what FASTA format looks like. Generally, there are header lines that start with a ">" symbol and have the name of the sequence, and subseqent lines have the actual DNA sequence (ATGC letters).

Similarly, if we want to browse a file without indicating the number of lines we want to inspect, we can use the more command.

```
more  GCF_000009585.1_ASM958v1_genomic.fna
```

grep

9 grep

Grep (Global Regular Expression Print) is perhaps the most popular and widely used Unix command. It is essentially a robust search tool that will find occurrences of a pattern in a given file.

Given the genome file we downloaded, maybe we want to know how many lines start with ">", since this will tell us how many distinct DNA sequences there are in the file (this is a standard for FASTA formatted data).
To do this we can type:

```
grep "^>" GCF_000009585.1_ASM958v1_genomic.fna
```

Here, the quotes surround the pattern we are searching for, and the "^" symbol specifies that we only want ">" symbols that start at the beginning of a line.

wc

10 wc

The "wc" command gives us the size of a given file. By default it gives us the newline count, the word count, and the byte count. If we want these statistics for our genome file, we can simply type:

```
wc GCF_000009585.1_ASM958v1_genomic.fna
```

pipes

11 Pipes, or the "|" symbol

One very useful aspect of Unix commands is that we can "pipe" the output of one command directly into another. This can simplify workflows substantially, since we don't always have to collect the output of one command and put it into another.

For example, let's combine two commands we already have experience with. Let's say we want to use "grep" to find all lines in our genome file that start with a ">", and then let's ask "wc" to count the occurrences for us:

```
grep "^>" GCF_000009585.1_ASM958v1_genomic.fna | wc
```

This output is simple enough now, but you can imagine if you had thousands of header lines in a file you would not want to count them by hand. Let's try downloading all of the proteins from the Staphylococcus aureus genome and counting them, using pipes and some commands from above:

```
wget
ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/009/585/GCF_000009585.1_ASM958v1/GCF_000009585.1_ASM958v1_protein.faa.gz
gunzip GCF_000009585.1_ASM958v1_protein.faa.gz
head GCF_000009585.1_ASM958v1_protein.faa
grep "^>" GCF_000009585.1_ASM958v1_protein.faa | wc
```

Note that pipes can be used to search with grep over multiple iterations. For example, maybe we want all the FASTA headers that have the term "ribosomal" in them:

```
grep "^>" GCF_000009585.1_ASM958v1_protein.faa | grep "ribosomal"
```

and now we want to count how often this happened:

```
grep "^>" GCF_000009585.1_ASM958v1_protein.faa | grep "ribosomal" | wc
```

12 Writing to output with the ">" symbol

If we want to write the output of a command to a specific file, we can do so with the ">" symbol (this is unrelated to the use of this symbol in FASTA files).

For example, perhaps we want to record only the FASTA header lines and put them in a file called "fasta_headers.txt". We can do this with:

```
grep "^>" GCF_000009585.1_ASM958v1_genomic.fna > fasta_headers.txt
```

Now if you use "ls" and "head" you should see the new file has been created, and the contents are what we would expect.

git clone

13 **git clone**

It may often be useful to download code from a GitHub repository. I have one set up with a variety of different code that we will use in this class. To download a repository you need to find the URL and then use "git clone". This will download a directory with the appropriate files inside. Here we will download a repository called "unix_tutorial"

```
git clone https://github.com/faylward/unix_tutorial
```

After you do this you should confirm the new directory is there with "ls". Them move into the directory using "cd".

sort

14 sort

Inside the directory "unix_tutorial" there should be a file called "protein_lengths.txt". Take a look at it using "head" and "tail".
This file contains the length of all proteins in the Staph aureus genome.
Let's say we want to find the longest and shortest proteins. For this we can use "sort"

```
sort -k 2,2 -n protein_length.txt | head
```

Make sure to pipe the output to a "head" command or you will get all protein lengths output to your command line.

Here "-k 2,2" tells sort that we only want to sort by the second column. "-n" is to specify that we need a numeric sort (as opposed to a alphabetical sort). We can also sort in reverse order using the "-r" flag:

```
sort -k 2,2 -rn protein_length.txt  | head
```

Note that you may get "broken pipe" errors here, but we can ignore them.

cut

15 cut

Sometimes it is useful to select and output only some columns of an output. For this we can use the "cut" command.
For example, to output only the protein lengths from the command above, we can use:

```
sort -k 2,2 -rn protein_length.txt | cut -f 2 | head
```

Or to output only the names:

```
sort -k 2,2 -rn protein_length.txt | cut -f 1 | head
```

Key flags for "cut" are -f, which specifies which columns to output, and -d, which specifies the delimiter (tabs are the default). If we had more columns and we wanted to output more than one, we coud use "-f 1-2" for columns 1 and 2 or "-f 2,3-5" for columns 2, 3, 4, and 5.

touch and rm

16 Lastly, we will go over how to create and remove files with rm.

let's create a dummy file with "touch" first.

```
touch test_file.txt
```

and to remove it:

```
rm test_file.txt
```

BE VERY CAREFUL WITH THE "RM" COMMAND. TYPING SOMETHING LIKE "RM *" WILL DELETE ALL OF YOUR FILES AND THERE WILL BE NO WAY TO RECOVER THEM.