Bioinformatics Practical Part 1

Introduction to the Linux command line and Bash scripting

Exercises

MP235

SS 2022

The following exercises are based on material from © Software Carpentry and have been modified to fit this course.
http://software-carpentry.org/

In this part of the practical you will learn how to interact with the Linux operating system using a command line interface.

Before we get started please consider the following:
Whenever you get stuck, don't hesitate to ask for help. For most commands you can also try using the `man` command or the `--help` option for additional information (for instance `man ls` will bring up the manual page of the `ls` command, more on that later). Also keep in mind that there may be more than one way of achieving the same result. In case anything goes wrong, you can always terminate commands during execution using Ctrl+C (Strg+C). You can also speed up things by using the Tab key for autocompletion. In cases where the autocompletion is ambiguous, hitting Tab twice will bring up the available options.

1 Navigating the File System

In this section you will learn to navigate files and directories using the commands `ls`, `pwd` and `cd`.
Go into the working directory for this part of the practical by entering the following command:

```
$ cd /mnt/practical2022/part1/files
```

1.1 Listing the Items of a Directory

The `ls` command, short for *list*, lists the contents of directories (also called folders). Like most Linux commands, the behaviour of the `ls` command can be modified by including options.

Type `ls --help` into your terminal.

You see a list of all the available options for the `ls` command. Note that some start with a single dash (`-`) and others with a double dash (`--`). Multiple short options (single characters) can be strung together, as long as the options themselves do not take any arguments.

If no argument is provided to the `ls` command, it will default to listing the contents of the current working directory.

Exercise 1.1

What does the option `-l` of the `ls` command do, and what happens if you also add the option `-h`, i.e., `ls -l` and `ls -lh`?
Hint: Try using `man ls` to bring up the manual page of the ls command. You can exit the manual by pressing q on your keyboard. Try out the commands for yourself.

1.2 Where are you and where do you want to go?

The directory you are currently in, your working directory, can be displayed using the command `pwd`, short for *present working directory*. Absolute paths always start from the root directory (`/`) whereas relative paths always start from the current directory.

There are special notations and shorthands for the current directory (`.`), it's parent directory (`..`), the home directory (`~`) and the last visited directory (`-`).

The `cd` command, short for *change directory*, can be used to change your current working directory, by passing the respective path as an argument. When no argument is provided it defaults to the users home directory, which is the default working directory when a user logs in. On the other hand, the working directory refers to the user's current directory at any given time.
On our VM, the home directory is `/home/ubuntu`. So typing "`cd /home/ubuntu`" is equivalent to typing "`cd ~`".

Let's try out a hypothetical example.

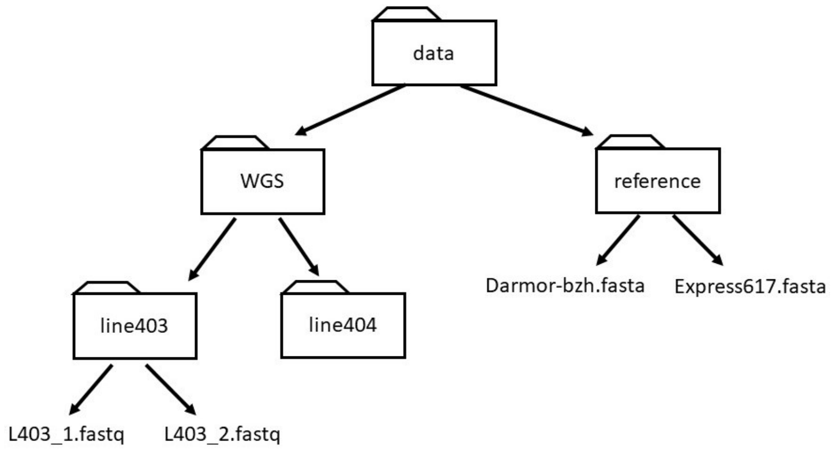
Exercise 1.2

Starting from /Users/amanda/data, which of the following commands could Amanda use to navigate to her home directory, which is /Users/amanda?

1. `cd .`
2. `cd /`
3. `cd /home/amanda`
4. `cd ../..`
5. `cd ~`
6. `cd home`
7. `cd ~/data/..`
8. `cd`
9. `cd ..`

1.3 Relative Paths

The following diagram shows a partial file system starting at the data directory which is located in Amandas home directory. From this we can see the organization of files and directories within the data directory. It contains reference genomes and data from whole-genome-sequencing experiments.



Exercise 1.3

Using the partial file system diagram above, if `pwd` displays `/Users/amanda/data/reference`, what will `ls -F ../WGS` display?

Hint: The `-F` option of the `ls` command marks items with an indicator for each type of item, e.g. `/` for directories, `*` for executable files and nothing for regular files.

1. `../WGS: No such file or directory`
2. `line403/ line404/`
3. `Darmor-bzh.fasta Express617.fasta`
4. `L403_1.fastq L403_2.fastq`

1.4 `ls` Reading Comprehension

Exercise 1.4

Using the same partial file system diagram as before, if `pwd` again displays `/Users/amanda/data/reference`, and `-r` tells ls to display things in reverse order, what command(s) will result in the following output:

```
Output:  Express617.fasta Darmor-bzh.fasta
```

1. `ls -r pwd`
2. `ls -r`
3. `ls -r -F /Users/amanda/data/reference`
4. `ls -Fr .`

2 Working With Files and Directories

In this section you will learn how to create, manipulate and delete files and directories using the commands `mkdir`, `touch`, `mv`, `cp` and `rm`.

2.1 Creating Files and Directories

Empty files can be created by using the `touch` command. There are also other ways of creating files which we will learn about later.

Directories can be created using the `mkdir` command, short for *make directory*.

By default, `ls` lists the contents of a directory in alphabetical order by name. The command `ls -t` lists items by time of last change instead of alphabetically. The option `-r` lists the contents of a directory in reverse order.

Exercise 2.1

Within the `/mnt/practical2022/part1/files` directory, create a directory called `my_directory` using the `mkdir` command and within that directory create a file called `my_first_file` using the `touch` command. Then create a second file within the same directory called `my_second_file`.

```
$ cd /mnt/practical2022/part1/files
$ mkdir my_directory
$ touch my_directory/my_first_file
$ touch my_directory/my_second_file
```

Which file is displayed last when you combine the `-t` and `-r` options of the ls command? Hint: You may need to use the `-l` option to see the last changed dates (`ls my_directory -lrt`).

Exercise 2.2

What are the file sizes of the newly created files?

Exercise 2.3

You're starting a new experiment and would like to duplicate the directory structure from your previous experiment so you can add new data.
Assume that the previous experiment is in a folder called 2022-09-12, which contains a data folder that in turn contains folders named raw and processed that contain data files. The goal is to copy the folder structure of the 2022-09-12 folder into a folder called 2022-09-13 so that your final directory structure looks like this:

```
2022-09-13/
|__ data
   |__ processed
   |__ raw
```

Which of the following set of commands would achieve this objective? What would the other commands do?
Hint: use `man mkdir` to find out what the `-p` option does.

1.
   ```
   $ mkdir 2022-09-13
   $ mkdir 2022-09-13/data
   $ mkdir 2022-09-13/data/processed
   $ mkdir 2022-09-13/data/raw
   ```

2.
   ```
   $ mkdir 2022-09-13
   $ cd 2022-09-13
   $ mkdir data
   $ cd data
   $ mkdir raw processed
   ```

3.
   ```
   $ mkdir 2022-09-13/data/raw
   $ mkdir 2022-09-13/data/processed
   ```

4.
   ```
   $ mkdir -p 2022-09-13/data/raw
   $ mkdir -p 2022-09-13/data/processed
   ```

5.
   ```
   $ mkdir 2022-09-13
   $ cd 2022-09-13
   $ mkdir data
   $ mkdir raw processed
   ```

2.2 Moving Files to Another Folder

You can move files into different folders using the `mv` command. You need to provide at least one file and one destination as arguments to the `mv` command. You can move multiple files at once, however the last argument you provide always corresponds to the destination. If the destination you provide is a directory, then the file will be moved into that directory, however, if the destination you provide is a file, then the file is not only moved, but also the name of the file changes according to the file name you provided.

Exercise 2.4

After running the following commands, Jamie realizes that the files `alignment1.bam` and `alignment2.bam` had landed in the wrong folder. The files should have been placed in the `alignments/` folder.

```
$ ls -F
alignments/ reference/ reads/
$ ls -F reference
reference_genome.fasta alignment1.bam alignment2.bam
$ cd reference
```

Fill in the blanks to move these files to the `alignments/` folder (i.e. the folder where they belong)

```
$ mv alignment1.bam alignment2.bam ____/____
```

2.3 Copying and Renaming Files

You can copy a file using the `cp` command. It works analogously to the `mv` command. You need to provide at least two arguments, with the last argument corresponding to the destination, which can be a directory or may also include a file name for the copy. Duplicate file names in the same directory, however, are not allowed. It is also possible to copy entire folders with everything contained within, by using the `-r` option. Renaming a file in Linux is equivalent to moving it to the same folder (using the `mv` command), but giving it a different name.

Exercise 2.5

Suppose that you created a plain-text file in your current directory to contain a list of the statistical tests you will need to do to analyze your data, and named it: statstics.txt
After creating and saving this file you realize you misspelled the file name! You want to correct the mistake, which of the following commands could you use to do so?

1. `cp statstics.txt statistics.txt`
2. `mv statstics.txt statistics.txt`
3. `mv statstics.txt .`
4. `cp statstics.txt .`

Exercise 2.6

What is the output of the closing `ls` command in the sequence shown below?

```
$ pwd
/Users/jamie/data
$ ls
variants.vcf
$ mkdir variants
$ mv variants.vcf variants/
$ cp variants/variants.vcf ../variants-saved.vcf
$ ls
```

1. `variants-saved.vcf variants`
2. `variants`
3. `variants.vcf variants`
4. `variants-saved.vcf`

2.4 Removing Files and Directories Safely

We can remove files using the `rm` command and, analogous to the `cp` command, directories and all of their contents can be removed recursively, by using the `-r` option. Be careful, however, deleted files do not go into a trash bin, and can not be recovered! The `rm` command should therefore be used with caution. Linux assumes you know exactly what you are doing and will not ask for confirmation unless you explicitly tell it to.

Exercise 2.7

Within the /mnt/practical2022/part1/files directory, what happens if we execute rm -ir my_directory?
Why would we want this protection when using rm?

2.5 Copying Multiple Files

It is possible to copy multiple files at once, by providing more than two arguments to the cp command. When doing so, the last argument must be the directory into which the files should be copied.

Exercise 2.8

For this exercise, you can test the commands in the /mnt/practical2022/part1/files/herbs directory.
In the example below, what does cp do when given several file names and a directory name?

```
$ mkdir backup
$ cp herbs/mint.dat herbs/oregano.dat backup/
```

In the example below, what does cp do when given three or more file names?

```
$ cd herbs
$ ls -F
basil.dat  mint.dat  oregano.dat
$ cp mint.dat oregano.dat basil.dat
```

2.6 List file names Matching a Pattern

* is a wildcard, which matches zero or more characters.

Go into the /mnt/practical2022/part1/files/chromosomes directory, which contains fasta index files (.fai) with information about chromosomes/contigs of genome assemblies:

```
$ cd /mnt/practical2022/part1/files/chromosomes
```

List all files that end in .fai.

```
$ ls *.fai
Beta_vulgaris.fasta.fai    Phaseolus_vulgaris.fasta.fai
Brassica_napus.fasta.fai   Solanum_lycopersicum.fasta.fai
Hordeum_vulgare.fasta.fai  Solanum_tuberosum.fasta.fai
```

If you instead list all files that contain the pattern S*.fai, only files which start with a capital S and end with .fai will be listed:

```
$ ls S*.fai
Solanum_lycopersicum.fasta.fai  Solanum_tuberosum.fasta.fai
```

? is also a wildcard, but it matches **exactly** one character (i.e., can not replace zero or more than one character). Wildcards can also be used in combination with each other e.g. ?????um_*.fai matches five characters followed by um_, which is then followed by any number of characters (including zero) and lastly .fai.

```
$ ls ?????um_*.fai
Hordeum_vulgare.fasta.fai       Solanum_tuberosum.fasta.fai
Solanum_lycopersicum.fasta.fai
```

When the shell sees a wildcard, it expands the wildcard to create a list of matching file names before running the command that was asked for. If a wildcard expression does not match any file, Bash will pass the expression as an argument to the command as it is. However, generally commands like wc and ls see the lists of file names matching these expressions, but not the wildcards themselves. It is the shell, not the other programs, that deals with expanding wildcards.

Exercise 2.9

When run in the chromosomes directory, which ls command(s) will produce this output?
`Beta_vulgaris.fasta.fai` `Hordeum_vulgare.fasta.fai`
`Phaseolus_vulgaris.fasta.fai`

1. `ls *vulgar??.fasta.fai`
2. `ls *v*`
3. `ls *vulgar*.fasta`
4. `ls *vulgar?*.fasta.fai`

Exercise 2.10

Jamie is working on a project and she sees that her files aren't very well organized:

```
$ ls -F
alignments/  leaf_rna.sam    raw/   root_rna.sam
```

The leaf_rna.sam and root_rna.sam files contain the output from aligning RNA-seq reads to a reference genome. Using at least one wildcard, which command(s) could she run so that the commands below will produce the output shown?

```
$ ls -F
alignments/   raw/
$ ls alignments
leaf_rna.sam    root_rna.sam
```

3 Pipes and Filters

In this section we will learn how to use filters, such as the `wc`, `sort` and `uniq` commands, in order to transform streams of data and how to pipe data from one command to the next using the pipe operator (`|`). Furthermore, we will also learn to store outputs in files using the `>` operator and how to use the commands `echo`, `cat`, `less`, `head` and `tail`.

3.1 Counting File Contents

`wc` is the '*word count*' command: it counts the number of lines, words, and characters in files (from left to right, in that order).
For instance, if we go into the `/mnt/practical2022/part1/files/chromosomes` directory and enter `wc Brassica_napus.fasta.fai`, we get the following output:

```
$ cd /mnt/practical2022/part1/files/chromosomes
$ wc Brassica_napus.fasta.fai
19  95 541 Brassica_napus.fasta.fai
```

The options `-l`, `-w` and `-m` can be used to only display the number of lines, words and characters respectively.

Exercise 3.1

What happens if you enter the command `wc -l *.fai`?

3.2 Sorting File Contents

The contents of files can be sorted using the `sort` command.

Exercise 3.2

The file `/mnt/practical2022/part1/files/numbers.txt` contains the following lines:

```
10
2
19
22
6
```

If we run `sort` on this file, the output is:

```
10
19
2
22
6
```

If we run `sort -n` on the same file, we get this instead:

```
2
6
10
19
22
```

Explain why `-n` has this effect.

3.3 Routing the Output of a Command into a File

The `echo` command prints whatever text you provide as an argument.

```
$ echo The echo command prints text
The echo command prints text
```

Instead of displaying the output of a command directly in the terminal (standard output) we can redirect it into a file using `command > file_name`. Note that this operator will create a new file, if it doesn't exist, or will **overwrite** an existing file that has the same name.

The contents of a file can be displayed using the command `cat`, which dumps the entire contents of a file onto the screen. If, instead, you want to browse through the contents of a file which has many more lines than what fits on the screen, then you are better off using the `less` command which lets you scroll through file contents using the up and down arrow keys. To stop viewing the file contents you can press Q, as in *quit*.

Exercise 3.3

There is another operator (`>>`) which looks similar, but behaves a bit differently than '`>`'. Test out the commands below to reveal the difference between the two operators:

```
$ echo hello > testfile01.txt
```

and:

```
$ echo hello >> testfile02.txt
```

Hint: Try executing each command twice in a row and then examining the output files using the `cat` or the `less` command.

Exercise 3.4

The `head` command is similar to the `cat` command, but instead of dumping the entire content of a file onto the screen, it only prints the first 10 lines from the start of a file by default. `tail` is the counterpart to `head`, which instead prints lines from the end of a file.
Consider the file `Brassica_napus.fasta.fai` and the `chromosomes` folder. After executing the following commands, select the answer that corresponds to the newly created file `Bnapus_subset.txt`:

```
$ head -n 3 Brassica_napus.fasta.fai > Bnapus_subset.txt
$ tail -n 2 Brassica_napus.fasta.fai >> Bnapus_subset.txt
```

1. The first three lines of `Brassica_napus.fasta.fai`
2. The last two lines of `Brassica_napus.fasta.fai`
3. The first three lines and the last two lines of `Brassica_napus.fasta.fai`
4. The second and third lines of `Brassica_napus.fasta.fai`

3.4 Routing the Output of a Command into another Command

We can use the output of one command as the input into another command by using the vertical bar (`|`), called the **pipe operator**.
If we consider the command used in subsection 3.1 (`wc -l *.fai`) which was run in the `chromosomes` directory, the number of lines in each file was displayed directly in the terminal. However, if we were only interested in knowing what the total number of lines is, we could do the following:

```
$ wc -l *.fai | sort -n | tail -n 1
70 total
```

Exercise 3.5

In our current directory, we want to find the 3 files which have the **lowest** number of lines. Which command listed below would work?
Note that the `sort -n` command sorts numerically in **ascending order**.

1. `wc -l * > sort -n > head -n 3`
2. `wc -l * | sort -n | head -n 1-3`
3. `wc -l * | head -n 3 | sort -n`
4. `wc -l * | sort -n | head -n 3`

Exercise 3.6

The file called numbers.txt (in the `/mnt/practical2022/part1/files` folder) contains the following data:

```
10
2
19
22
6
```

What text passes through each of the pipes and the final redirect in the pipeline below? Note, the sort -r command sorts in reverse order.

```
$ cat numbers.txt | head -n 4 | tail -n 3 | sort -rn > final.txt
```

Hint: build the pipeline up one command at a time to test your understanding

Exercise 3.7

Take a look at the file `sample_info.txt` in the `/mnt/practical2022/part1/files` folder, which contains information about publicly available RNA-seq data. Consider the following command:

```
$ cut -f 2 sample_info.txt
```

The `cut` command is used to remove or 'cut out' certain sections of each line in the file, and cut expects the lines to be separated into columns by a Tab separator/delimiter. Other delimiters can be specified using the `-d` option. We have used the `-f` option to specify that we want to extract the second field (column). This gives the following output:

```
$ cut -f 2 sample_info.txt
tissue
young_leaf
young_leaf
flower_bud
flower_bud
silique_pericarp
silique_pericarp
root
root
seed
silique
stem
stem
seed
seed
silique
anther
anther
sepal
ovule
silique
pistil
```

The `uniq` command filters out adjacent matching lines in a file. How could you extend this pipeline (using `uniq` and three other commands) to find out the number of unique tissues from which the RNA-seq data was sampled?
Hint: You can use `tail -n +2`, to get the file contents starting from row 2 (excluding the header). Since the `uniq` command only removes adjacent matching lines, you will need to use `sort` before using `uniq` in order to get rid of all duplicates and not just those that are adjacent to each other by chance.

3.5 Recap: Removing Files

Exercise 3.8

Suppose you want to delete your processed data files, and only keep your raw files and processing script to save storage. The raw files end in `.dat` and the processed files end in `.csv`. Which of the following would remove all the processed data files, and only the processed data files? **Do not execute these commands, just think about them!**

1. `rm ?.csv`
2. `rm *.csv`
3. `rm * .csv`
4. `rm *.*`

4 Loops

Loops are a powerful tool for automating repetitive tasks, by performing the same set of instructions for each item in a list.
Go into the `herbs` directory

```
$ cd /mnt/practical2022/part1/files/herbs
```

Suppose we have several hundred data files with names like basil.dat, mint.dat, and oregano.dat. The structure of these files is the same: the common name, classification, and last update are presented on the first three lines, with DNA sequences on the following lines. Let's look at the files:

```
$ head -n 5 basil.dat mint.dat oregano.dat
```

We would like to print out the classification for each species, which is given on the second line of each file. For each file, we would need to execute the command `head -n 2` and pipe this to tail -n 1. We'll use a loop to solve this problem for these 3 files, though in principal we could do the same for hundreds or thousands of files.

The indentation is not required, but it increases legibility.

```
$ for file name in basil.dat mint.dat oregano.dat
> do
>     head -n 2 $file name | tail -n 1
> done
CLASSIFICATION: Ocimum basilicum
CLASSIFICATION: Mentha spicata
CLASSIFICATION: Origanum vulgare
```

Notice how the prompt changes from `$` to `>`, reminding us that the shell expects us to continue the command on the next lines following the `for` keyword, until it receives the `done` keyword.

4.1 Writing your own Loop

As you may have noticed, the keyword `for` is always followed by the name of a variable, then the keyword `in` and a list of items. The list can be simply provided as arguments, each separated by a single space. When accessing the variable within the loop, we simply write the variable name preceded by a `$`.

Exercise 4.1

How would you write a loop that echoes all 10 numbers from 0 to 9?

4.2 Loop Comprehension

Exercise 4.2

This exercise refers to the `/mnt/practical2022/part1/files/chromosomes` directory, where `ls *.fai` gives the following output:

```
$ cd /mnt/practical2022/part1/files/chromosomes
$ ls *.fai
Beta_vulgaris.fasta.fai  Brassica_napus.fasta.fai  Hordeum_vulgare.fasta.fai
Phaseolus_vulgaris.fasta.fai  Solanum_lycopersicum.fasta.fai
Solanum_tuberosum.fasta.fai
```

What is the output of the following code?

```
$ for datafile in *.fai
> do
>     ls *.fai
> done
```

Now, what is the output of the following code?

```
$ for datafile in *.fai
> do
>     ls $datafile
> done
```

Why do these two loops give different outputs?

Exercise 4.3

What would be the output of running the following loop in the `chromosomes` directory?

```
$ for filename in H*
> do
>     ls $filename
> done
```

1. No files are listed.
2. All files are listed.
3. Only `Brassica_napus.fasta.fai` and `Beta_vulgaris.fasta.fai` are listed.
4. Only `Hordeum_vulgare.fasta.fai` is listed.

4.3 Saving Files within Loops

Exercise 4.4

In the `/mnt/practical2022/part1/files/herbs` directory, what is the effect of this loop?

```
for herb in *.dat
do
    echo $herb
    cat $herb > herbs.dat
done
```

1. Prints `basil.dat`, `mint.dat` and `oregano.dat`, and the text from `oregano.dat` will be saved to a file called `herbs.dat`.
2. Prints `basil.dat` and `oregano.dat` and the text from both files is concatenated and saved to a file called `herbs.dat`.
3. Prints `basil.dat` and `mint.dat` and the text from `oregano.dat` will be saved to a file called `herbs.dat`.
4. None of the above.

Exercise 4.5

Also in the `/mnt/practical2022/part1/files/herbs` directory, what would be the output of the following loop?

```
for datafile in *.dat
do
    cat $datafile >> all.dat
done
```

1. All of the text from `basil.dat` and `mint.dat` would be concatenated and saved to a file called `all.dat`.
2. The text from oregano.dat will be saved to a file called `all.dat`.
3. All of the text from `basil.dat`, `mint.dat` and `oregano.dat` would be concatenated and saved to a file called `all.dat`.
4. All of the text from `basil.dat`, `mint.dat` and `oregano.dat` would be printed to the screen and saved to a file called `all.dat`.

4.4 Doing a Dry Run

A loop is a way to do many things at once — or to make many mistakes at once if it does the wrong thing. One way to check what a loop would do is to echo the commands it would run instead of actually running them.

Exercise 4.6

Suppose we want to preview the commands that the following loop will execute without actually running those commands:

```
$ for datafile in *.dat
> do
>     cat $datafile >> all.dat
> done
```

What is the difference between the two loops below, and which one would we want to run? Note that variables can still be called within quotation marks.

# Version 1

```
$ for datafile in *.dat
> do
>     echo cat $datafile >> all.dat
> done
```

# Version 2

```
$ for datafile in *.dat
> do
>     echo "cat $datafile >> all.dat"
> done
```

4.5 Nested Loops

Exercise 4.7

Suppose we want to set up a directory structure to organize some experiments involving transcriptomic and genomic sequencing data.
What would be the result of the following code:

```
$ for species in basil mint oregano
> do
>     for experiment in rna-seq reseq nanopore
>     do
>         mkdir $species-$experiment
>     done
> done
```

5 Bash Scripting

To save sequences of shell commands, we can write a bash script using a text editor. To open the command line text editor nano, type `nano` with the name of the script as an argument. If the script does not exist yet, a new file will be created.

5.1 Using Variables in Bash Scripts

A bash script can be executed using the `bash` command followed by the script name and possibly a number of positional command line arguments.

Positional command line arguments can be accessed within a script by using the dollar sign followed by a number, the number being the position starting at 1. For instance `$1` for the command line argument at position 1. If we want to refer to all command line arguments at once, we can use `$@`.

Exercise 5.1

In the herbs directory, imagine you have a shell script called script.sh containing the following commands:

```
head -n $2 $1
tail -n $3 $1
```

While you are in the herbs directory, you type the following command:

```
$ bash script.sh "*.dat" 1 1
```

Which of the following outputs would you expect to see?

1. All of the lines between the first and the last lines of each file ending in .dat in the herbs directory
2. The first and the last line of each file ending in .dat in the herbs directory
3. The first and the last line of each file in the herbs directory
4. An error because of the quotes around `*.dat`

Exercise 5.2

Leah has several hundred data files, each of which is formatted like this:

```
COMMON NAME: basil
CLASSIFICATION: Ocimum basilicum
UPDATED: 2020-05-02
CCCCAACGAG
GAAACAGATC
ATTAGAAGAT
CTGTCGCGAA
...
```

An example of this type of file is given in
`/mnt/practical2022/part1/files/herbs/basil.dat`

We can use the command `head -n 3 basil.dat | tail -n 1 | cut -d " " -f 2` to get the date when the file was last updated. In order to avoid having to type out this series of commands every time, a scientist may choose to write a shell script instead.
Write a shell script called dates.sh that takes any number of file names as command-line arguments, and uses a variation of the above command to print a list of the dates when these files had been updated.

Exercise 5.3

Write a shell script called longest.sh that takes the name of a directory and a file name extension as its arguments, and prints out the name of the file with the most lines in that directory with that extension. For example:

```
$ bash longest.sh /data/example txt
```

would print the name of the .txt file in a directory called `/data/example`, which has the most lines.
Feel free to test your script on one of the directories in the `/mnt/practical2022/part1/files` directory, e.g.

```
$ bash longest.sh /mnt/practical2022/part1/files/chromosomes fai
```

5.2 Script Reading Comprehension

Exercise 5.4

For this question, consider the `/mnt/practical2022/part1/files/herbs` directory once again. This contains a number of `.dat` files in addition to any other files you may have created. Explain what each of the following three scripts would do when run as `bash script1.sh *.dat`, `bash script2.sh *.dat`, and `bash script3.sh *.dat` respectively.

```
# Script 1
echo *.*
```

```
# Script 2
for filename in $1 $2 $3
do
    cat $filename
done
```

```
# Script 3
echo $@.dat
```

5.3 Debugging Scripts

Exercise 5.5

Suppose you have saved the following script in a file called do-errors.sh in the /mnt/practical2022/part1/files/expression_data directory:

```
# Calculate stats for data files.
for datafile in "$@"
do
    echo $datfile
    bash stats.sh $datafile stats-$datafile
done
```

When you run it from the expression_data directory:

```
$ bash do-errors.sh RNA-sample?A.tpm RNA-sample?B.tpm
```

the output is blank. To figure out why, re-run the script using the -x option:

```
$ bash -x do-errors.sh RNA-sample?A.tpm RNA-sample?B.tpm
```

What is the output showing you? Which line is responsible for the error?

Hint: use `man bash` to find out what the `-x` option does.

6 Finding Things

In this section we will learn how to find files using the `find` command and how to search for patterns within the contents of files using the `grep` command.

6.1 Using `grep`

To find all lines within a File that contain a given pattern of characters, simply provide the pattern as the first argument to the `grep` command and the file from which you want to extract the respective lines as the second argument, i.e. `grep pattern file`.

To only find exact matches with respect to word boundaries you can use the `-w` option, so `grep -w fun` would find all lines containing the word "fun" but not lines that contain the word "function".

The `-n` option shows the line numbers of matching lines in the output and the `-i` option makes your search case sensitive. You can of course combine these options with each other. Lastly the `-v` option inverts your search, so the output contains all lines that **do not** contain the search term and `-r` allows you to specify a directory or a set of files which will be searched recursively.

Exercise 6.1

If we are in the /mnt/practical2022/part1/files directory, which command would result in the following output:

```
SRR3343238      silique PRJNA317510     Lu et al. (2016)        NA
SRR3439930      silique PRJNA317510     Lu et al. (2016)        NA
SRR5885452      silique PRJNA394926     Sun et al. (2017)       poly-A
```

1. `grep silique sample_info.txt`
2. `grep -E silique sample_info.txt`
3. `grep -w silique sample_info.txt`
4. `grep -i silique sample_info.txt`

Exercise 6.2

Leah has several hundred data files containing gene expression data saved in one directory (`/mnt/practical2022/part1/files/expression_data`), each of which is formatted like this (gene names on the left, expression values on the right, separated by a tab):

```
A01p00010_BnaDAR        0
A01p00020_BnaDAR        2.5044
A01p00030_BnaDAR        18.1881
A01p00040_BnaDAR        0.821706
A01p00050_BnaDAR        8.52516
A01p00060_BnaDAR        4.12994
A01p00070_BnaDAR        0.452679
A01p00080_BnaDAR        0
A01p00090_BnaDAR        1.08753
A01p00100_BnaDAR        0.982617
A01p00110_BnaDAR        94.3992
...
```

She wants to write a shell script that takes a gene name as the command-line argument. The script should be run in the directory containing the data files and should return a single file called <gene name>.txt containing a list of sample file names (e.g. "RNA-sample1A.tpm") and the expression value of the respective gene for each sample. For instance, for the gene A01p00090_BnaDAR, the resulting file should have the name A01p00090_BnaDAR.txt and should look something like this:

```
RNA-sample1A.tpm    1.08753
RNA-sample1B.tpm    0
RNA-sample2A.tpm    0.613564
....
```

Below, each line contains an individual command, or operator. Arrange their sequence in one command in order to achieve Leah's goal.

Hint: The `sed` command can be used to edit strings. It is another very useful command to know, but we will not discuss it further. In this instance it is used to replace tabs (`\t`) with colons (`:`). Use `man grep` to look for how to grep text recursively in a directory and `man cut` on how to select more than one field in a line and how to change the output delimiter. The `$` in `$'\t'` makes the shell interpret this expression as a tab instead of the string `'\t'`. Try to build up the pipeline one step at a time.

- `cut -d / -f 2`
- `>`
- `sed 's/\t/:/'`
- `|`
- `|`
- `grep -w $1 -r .`
- `$1.txt`
- `|`
- `cut -d :  -f 1,3 --output-delimiter=$'\t'`

6.2 Finding Files

While `grep` finds lines in files, the find command finds files themselves. For instance the output of

```
$ find .
```

is the names of every file and directory within the current working directory and its subdirectories. This can seem useless at first but find has many options to filter the output.

The `-d` option can be used to only find directories, while the `-f` option only finds files. You can also use the `-name` option to find files with specific name patterns, for instance:

```
$ find . -name *.txt
```

Exercise 6.3

The `-v` option to grep inverts pattern matching, so that only lines which do not match the pattern are printed. Given that, which of the following commands will find all `.dat` files in herbs except oregano.dat? Once you have thought about your answer, you can test the commands in the `/mnt/practical2022/part1/files` directory.

1. `find herbs -name "*.dat" | grep -v oregano`
2. `find herbs -name *.dat | grep -v oregano`
3. `grep -v "oregnao" $(find herbs -name "*.dat")`
4. None of the above.