1. ls- which is short for listing. This command will list the contents of the current directory.

ls prints the names of the files and directories in the current directory. We can make its output more comprehensible by using the -F **option** (also known as a **switch** or a **flag**) , which tells ls to classify the output by adding a marker to file and directory names to indicate what they are:

* 1. a trailing / indicates that this is a directory
  2. @ indicates a link
  3. indicates an executable

ls is the **command**, with an **option** -F and an **argument** /. We’ve already encountered options (also called **switches** or **flags**) which either start with a single dash (-) or two dashes (--), and they change the behaviour of a command. Arguments tell the command what to operate on (e.g. files and directories). Sometimes options and arguments are referred to as **parameters**. A command can be called with more than one option and more than one argument: but a command doesn’t always require an argument or an option.

Each part is separated by spaces: if you omit the space between ls and -F the shell will look for a command called ls-F, which doesn’t exist. Also, capitalization can be important. For example,

1. ls -s will display the size of files and directories alongside the names, while ls -S will sort the files and directories by size

Getting help

1. ls has lots of other **options**. There are two common ways to find out how to use a command and what options it accepts:

We can pass a --help option to the command ls – help

1. ls -t lists items by time of last change instead of alphabetically.
2. The command ls -r lists the contents of a directory in reverse order.
3. ***cd*** stands for ‘change directory’, which is a bit misleading: the command doesn’t change the directory, it changes the shell’s idea of what directory we are in. The command to change locations is cd followed by a directory name to change our working directory. , The cd command is akin to double clicking a folder in a graphical interface to get into a folder.
4. -a stands for ‘show all’; it forces ls to show us file and directory names that begin with ., such as .. The special directory ’ ..’ doesn’t usually show up when we run ls. If we want to display it, we can add the -a option to ls -F. The command is ***ls -F -a***
5. pwd - which stands for ‘print working directory’). Directories are like *places* - at any time while we are using the shell we are in exactly one place, called our **current working directory**. Commands mostly read and write files in the current working directory, i.e. ‘here’, so knowing where you are before running a command is important. pwd shows you where you are.
6. If your screen gets too cluttered, you can clear your terminal using the ***clea****r* command. You can still access previous commands using ↑ and ↓ to move line-by-line, or by scrolling in your terminal.

### Create a directory

Let’s create a new directory called thesis using the command mkdir thesis (which has no output):

1. mkdir means ‘make directory’
2. Note that mkdir is not limited to creating single directories one at a time. The -p option allows mkdir to create a directory with any number of nested subdirectories in a single operation:

mkdir **-p** thesis/chapter\_1/section\_1/subsection\_1

1. The -R option to the ls command will list all nested subdirectories wtihin a directory. Let’s use ls -FR to recursively list the new directory hierarchy we just created beneath the thesis directory

### Create a text file

Let’s change our working directory to thesis using cd, then run a text editor called Nano to create a file called draft.txt:

1. nano draft.txt
2. touch my\_file.txt

**Lecture 2**

## **Moving files and directories**

1. mv, which is short for ‘move’

One has to be careful when specifying the target file name, since mv will silently overwrite any existing file with the same name, which could lead to data loss. An additional option, mv -i (or mv --interactive), can be used to make mv ask you for confirmation before overwriting.

Note that mv also works on directories.

mv -i BSIT/first.txt BSSE

Files can be renamed using mv

mv thesis/draft.txt thesis/quotes.txt

## **Copying files and directories**

1. The cp command works very much like mv, except it copies a file instead of moving it.

cp quotes.txt thesis/quotations.txt

1. ls with two paths as arguments — like most Unix commands, ls can be given multiple paths at once

ls quotes.txt thesis/quotations.txt

1. We can also copy a directory and all its contents by using the [recursive](https://en.wikipedia.org/wiki/Recursion) option -r, e.g. to back up a directory

cp **-r** thesis thesis\_backup

## **Removing files and directories**

1. The Unix command we’ll use for this is rm (short for ‘remove’)

rm quotes.txt

if we try to remove the thesis directory using rm thesis, we get an error message:

This happens because rm by default only works on files, not directories.

rm can remove a directory and all its contents if we use the recursive option -r, and it will do so without any confirmation prompts:

rm **-r** thesis

1. Given that there is no way to retrieve files deleted using the shell, rm -r should be used with great caution (you might consider adding the interactive option rm -r -i).
2. \* matches zero or more characters in a filename, so \*.txt matches all files ending in .txt.
3. ? matches any single character in a filename, so ?.txt matches a.txt but not any.txt.

# **Pipes and Filters**

1. wc is the ‘word count’ command: it counts the number of lines, words, and characters in files (from left to right, in that order).

If we run the command wc \*.pdb, the \* in \*.pdb matches zero or more characters, so the shell turns \*.pdb into a list of all .pdb files in the current directory:

Note that wc \*.pdb also shows the total number of all lines in the last line of the output.

1. If we run wc -l instead of just wc, the output shows only the number of lines per file:

The -m and -w options can also be used with the wc command, to show only the number of characters or the number of words in the files.

1. Which of these files contains the fewest lines? It’s an easy question to answer when there are only six files, but what if there were 6000? Our first step toward a solution is to run the command:

wc **-l** **\***.pdb > lengths.txt

The greater than symbol, >, tells the shell to **redirect** the command’s output to a file instead of printing it to the screen.

1. We can now send the content of lengths.txt to the screen using cat lengths.txt. The cat command gets its name from ‘concatenate’ i.e. join together, and it prints the contents of files one after another. There’s only one file in this case, so cat just shows us what it contains:

cat lengths.txt

1. cat in this lesson, for convenience and consistency, but it has the disadvantage that it always dumps the whole file onto your screen. More useful in practice is the command less, which you use with less lengths.txt. This displays a screenful of the file, and then stops. You can go forward one screenful by pressing the spacebar, or back one by pressing b. Press q to quit.
2. If we run sort on a file

If we run sort -n on the same input

We will also use the -n option to specify that the sort is numerical instead of alphanumerical. This does not change the file; instead, it sends the sorted result to the screen

1. Another command called head to get the first few lines in sorted-lengths.txt

head **-n** 1 sorted-lengths.txt

1. echo The echo command prints text
2. > operator overwrites the content whil. >> appends the content to a file
3. head **-n** 3 animals.txt > animals-subset.txt
4. $ tail **-n** 2 animals.txt >> animals-subset.txt
5. sort **-n** lengths.txt | head **-n** 1

The vertical bar, |, between the two commands is called a **pipe**. It tells the shell that we want to use the output of the command on the left as the input to the command on the right.

1. Nothing prevents us from chaining pipes consecutively. That is, we can for example send the output of wc directly to sort, and then the resulting output to head. Thus we first use a pipe to send the output of wc to sort:

wc **-l** **\***.pdb | sort **-n**

1. 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 character. A character used in this way is a called a **delimiter**. In the example above we use the -d option to specify the comma as our delimiter character. We have also used the -f option to specify that we want to extract the second field (column).

cut **-d** , **-f** 2 animals.txt