Unix shell

- Also “prompt”, “command line”, “terminal”
- Can think of this as a direct conversation with the computer instead of clicking on buttons etc.
- This is all about files and directories (a “directory” is the same thing as a “folder”)

Finding the terminal

- On a **Mac**:
  - Click on the Finder – that's the blue face in the bottom left corner
  - In the window that appears, click on applications on the left hand side
  - Scroll down to the bottom of this window, double-click on Utilities
  - Now scroll down again and double-click on Terminal
- This will be different on **Linux**!

pwd – **p**resent **w**orking **d**irectory

Usage: pwd

Prints the current directory
(where am I?)

ls – **l**ist **d**irectory

Usage: ls

Lists the contents of the current directory (what's here?)

cd – **c**hange **d**irectory

Usage: cd *directory_name*

Changes the current directory to another specified directory
(go somewhere else)

cp – **c**o**p**y

Usage: cp *file_from file_to*

Makes a copy of a file. The file to copy is given first, then the name of the new copy.

Tab completion

- Type the start of the filename, press Tab once
- If what you typed is unique in the current directory, the shell will fill in the rest for you
- If it's not unique, it won't – press Tab again to get a list of possible options

Arguments

- The information given to a command is called “arguments” or “parameters”. So if we type:

  cp file1 file2

  cp is the command, **file1** and **file2** are the arguments
- These arguments are separated by spaces

Spaces in filenames

- Because arguments are separated by spaces, spaces in filenames cause problems.
- For example:

  cp My file file2

  is ambiguous
- Two solutions:

1. Single quotes: cp 'My file' file2
2. Don't use spaces in filenames

mkdir – **m**a**k**e **dir**ectory

Usage: mkdir *new_directory*

Makes a new directory

mv – **m**o**v**e

Usage: mv *file_from file_to*

Moves a file

rm – **r**e**m**ove

Usage: rm *file1*

Deletes (i.e. removes) a file

Referring to files

- We can refer to files using just the name, if they are in the current directory

  rm file1

- We can also refer to them using the directory name, if they're in another directory

  rm another_dir/file2

- Finally, we can use an “absolute path”, giving all the directories leading to a file

  rm /Users/atullo2/work/stuff/file3

mv – **m**o**v**e

Usage:
mv *file_from file_to*
mv *file_from1 file_from2 .... dir_to*

Moves one file to another specified location (with a new name)
OR
Moves one or more files to a specified directory, without changing their names

cp – **c**o**p**y

Usage:
cp *file_from file_to*
cp *file_from1 file_from2 .... dir_to*

Copies one file to another specified location (with a new name)
OR
Copies one or more files to a specified directory, without changing their names

rm – **r**e**m**ove

Usage:
rm *file1 file2 file3 ....*
rm -r *file_or_dir1 file_or_dir2 ....*

Removes one or more files
OR
With '-r', removes one or more files or directories *with their contents*
(take care, as there is no undo!)

ls – **l**ist **d**irectory

Usage: ls *file_or_dir1 file_or_dir2*

Lists the files given and the contents of the directories given
OR
If no files or directories are given, list the current directory

cd – **c**hange **d**irectory

Usage: cd *directory_name*

Changes the current directory to another specified directory
OR
If no directory is specified, changes the current directory to the user's home directory

Shell history

- Use the up and down arrow keys to go backwards and forwards through previous commands that you've typed
- You can edit one and run it again by hitting return
- Or you can run one again without editing it

A few special symbols

- **..** means the directory above this one, e.g.

  cd ..
- ***** means (approximately!) “anything” e.g.

  rm ab*
- **?** means “any single character” e.g.

  rm data???.txt

A few special symbols, 2

- **.** (full stop) means the current directory, e.g.

  mv example_directory/* .
- **~** (a tilde) means your home directory, e.g.

  cp myfile.txt ~/
- cd on its own will take you back to your home directory

Getting help

- Most commands also have a help option
- Try:

  rm --help
- Most commands will interpret -h, --help, or sometimes -help as a request for help
- There's no universal standard for this
- As well as '-r', rm has other options
- This help is a bit cryptic though ...

Manual pages

- Often more comprehensive documentation is available in a manual page
- This is accessed via its own command

man – **man**ual page

Usage: man *command_name*

Displays the manual page for the given command.

Manual pages, 2

- Let's try it:

  man rm
- The manual page is displayed using a program called a “pager”
- This allows you to scroll through a file
- Use the arrow keys, or Space scrolls one page
- Press 'q' to quit
- (Press 'h' for help)

Manual pages, 3

- Try this:

  man cd
- This seems a bit strange!
- 'cd' is **built in** to the shell
- 'rm' (and most others) are **run from** the shell
- For built in commands, you can also use (e.g.):

  help cd

More tab completion

- Note that tab completion isn't just for filenames
- Try typing just:

  ex

  without hitting Return, and then press Tab twice as we did before for tab completion.
- You'll see all the programs which will run from the shell that start with 'ex'

apropos – search commands

Usage: apropos *word*

Displays a list of commands whose descriptions use that word.

Searching for commands

- Try this:

  apropos manual
- In the resulting list, you should be able to find the listing for the “man” command
- This can be useful if you can't remember the name of a command
- These lists often include a **lot** of stuff, especially if you search for a common word

echo – print some text

Usage: echo *text*

Prints some text to the standard output

What is the standard output?

- When a program produces some output it goes to the **standard output**
- For example, try:

  echo hello!
- When we use (for example) the 'ls' command, this is also going to the standard output
- Note that this does **not** include errors, which are different

Redirection

- Using '>' we can **redirect** the standard output to a file
- Let's try this:

  echo hello! >greeting
- Use 'ls' to list the current directory, you'll see a file called 'greeting'
- Let's see what's in it
- How do we see what's in it? (from the shell)

cat – con**cat**enate

Usage: cat *filename*

Prints the contents of a file to standard output

Taking a look at a file

- Now try:

  cat greeting
- You'll see 'hello!' (or whatever you typed before)

Use of 'cat'

- Let's create another file:

  echo goodbye! >farewell
- Check the contents of this file:

  cat farewell
- And now, to use cat for its declared purpose:

  cat greeting farewell

Even more redirection

- The 'cat' command **concatenates** the contents of the two files

  (of course it was concatenating one file before!)
- We can also redirect the **result** of the concatenation to a file:

  cat greeting farewell >conversation
- Now look at it, using cat to send it to standard output:

  cat conversation

There's also a standard input ....

- We can use standard input to 'feed' input to commands, too

  cat <greeting

Redirection example

- Let's look at a real example of redirection
- First we're going to create a new text file
- Type:

  nano

nano – a simple text editor

Usage: nano *filename*

Edit (or create) a text file

Creating a text file

- In nano, type a few lines of text
- For our example, use the letters 'a' and 'e' a few times in the text
- To save,
  - hit Ctrl-X (also written ^X)
  - answer Y (Yes) to the question
  - type a filename (e.g. example_text) and hit Return

    (recall that spaces in filenames are a Bad Idea)

tr – **tr**anslate characters

Usage: tr *chars_in chars_out*

Take text from standard input, and translate each of the characters in *chars_in* to its equivalent in *chars_out.*

Let's try it ....

- I'm going to use 'example_text' as the filename, but you should write whatever you used in nano
- Try this:

  tr e E <example_text

- What happened?

Character translation

- 'tr' has changed all of the 'e' characters in your text into 'E'
- This **didn't** affect your file, though

  (go and have a look, if you like, using 'cat')
- Instead a changed copy of the text was sent to the standard output
- As before, we could redirect this to a file:

  tr e E <example_text >example_replaced

Some more 'tr' examples

- Try these:

  tr e a <example_text

  tr ea 34 <example_text

  tr ' ' _ <example_text

  tr aeio uuuu <example_text

grep

- 'grep' searches text for matching lines
- We're going to use an example text to search
- Let's use Immanuel Kant's *Critique of Pure Reason:*

  http://edin.ac/1pnVYvL
- Download this to your home directory
- Remember you can use 'pwd' to find out where this is, then 'ls' to check it's actually there

grep – search for matching lines

Usage: grep *pattern file1 file2 ...*
Usage: grep *pattern*

Search the specified files for matching lines, printing them to standard output
OR
Search standard input for matching lines, printing them to standard output

Search for phenomena

- Try this:

  grep phenomena kant.txt
- The simplest kind of grep pattern is just some ordinary text
- Using this pattern, we find all lines containing that text

Search for phenomena, 2

- Try this:

  grep phenomena <kant.txt

- This gives you the same result
- That's because we're using the alternative usage, where we search standard input

Search for phenomena, 3

- Try this:

  grep phenomena <kant.txt >phenom_lines.txt

- As you might expect, this redirects the matched lines to a file

Making grep more useful

- What if we want to see where these lines are?
- At the moment we just get the lines themselves
- Take a look at the manual page for 'grep'
- See if you can find something that would help

Other grep patterns

- grep uses 'regular expressions' in its patterns
- These are beyond the scope of this class
- Here's a quick example:

  Previously we found 'phenomena' and 'phenomenal' in our search. We can search for just 'phenomena' with:

  grep '\bphenomena\b' kant.txt

- The special sequence \b matches the start or end of a word

Pipe

- A pipe character: |
- This is used to connect the output of one program to the input of another
- **PC keyboard:** You can find this at the bottom left of the keyboard next to the left Shift key
- **Mac keyboard:** You can find this above the right Shift key, next to Return

Pipe, example use

- Looking back at our example with tr, we saw that we can translate 'e' to 'E' with:

  tr e E

- Let's try sending the output of another program, to the input of this one:

  ls | tr e E

- Note that the 'ls' command **notices that it's connected to a pipe** and changes its output format!

less – allow the user to scroll through a file

Usage: less *filename*
Usage: less

Allow the user to scroll through the specified file
OR
Allow the user to scroll through lines from standard input

Using 'less'

- Try this:

  less kant.txt

- As with manual pages you can use the Space bar to go through page-by-page

Another pipe example

- Try this:

  grep phenomena kant.txt | less

- So the output from 'grep' is going in to 'less', where you can scroll through it.

Final example (exercise)

- What if I want to list everything in my home directory containing a D, then write these to a file? Use ls, grep, pipe and redirect