Week 1: Foundations of Computing, UNIX

MSc/MRes CMEE 2014-15

Samraat Pawar

Imperial College
London

October 10, 2014

SOME LOOSE ENDS

- Library induction and help – Elizabeth Killeen, *e.killeen@imperial.ac.uk* (slides will be available soon)
- Won't use Wallace till week 9 – keep tuned into emails
- Has everybody been getting my emails?
- New Laptops
- Introductions (again!) – who are you, where were you before, why CMEE?

Overview of the rest of the week

- UNIX
- Shell scripting
- Version control with Git
- LaTeX

UNIX: WHAT?

- Operating system developed in the 1970s by AT&T programmers (notably Brian Kernighan and Dennis Ritchie, fathers of C)
- Multi-user and network-oriented by design
- Uses plain text files for storing data and strictly hierarchical file system
- Machine independent OS
- Linux and Mac OS are Unix-like (or UN*X or *nix) operating systems
- Ubuntu is a Linux distribution

UNIX: WHY?

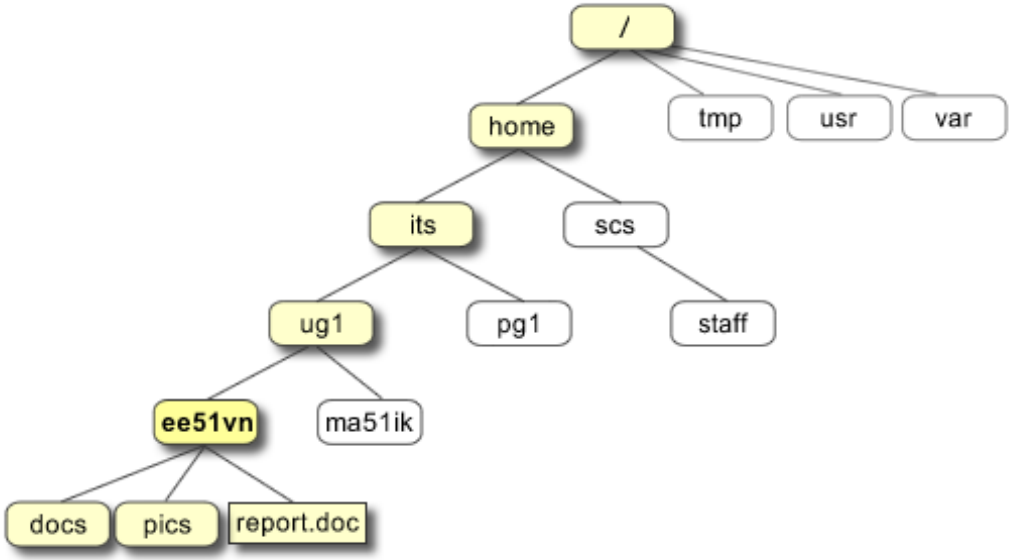
- Designed for developing code and storing data
- Scores of small programs available to perform simple tasks – can be combined easily
- Easy to automate tasks using (e.g., shell scripts)
- Multi-user (multiple users can log in concurrently use computer)
- Multi-tasking (can perform many tasks at the same time)
- Network-ready (easy to communicate between computers)
- Robust, stable, secure (very few UNIX viruses and malware)
- Free and open source!

UNIX: WHY?

- Designed for developing code and storing data
- Scores of small programs available to perform simple tasks – can be combined easily
- Easy to automate tasks using (e.g., shell scripts)
- Multi-user (multiple users can log in concurrently use computer)
- Multi-tasking (can perform many tasks at the same time)
- Network-ready (easy to communicate between computers)
- Robust, stable, secure (very few UNIX viruses and malware)
- Free and open source!

See `http://www.whylinuxisbetter.net/` to aid your brain-washing

UNIX DIRECTORY STRUCTURE



http://pathanruet.files.wordpress.com/2012/05/unix-tree.png

`/` Is the "root" directory

`/bin` Contains basic programs

`/etc` Contains configuration files

`/dev` Contains files connecting to devices (keyboard, mouse, screen, etc.).

`/home` Your home directory – this is where you will usually work

`/tmp` Contains Temporary files

MEET THE UNIX SHELL

- The shell is a text command processor to interface with the OS' "kernel"

MEET THE UNIX SHELL

- The shell is a text command processor to interface with the OS' "kernel"
- We will use the popular (yes!) `bash` shell

MEET THE UNIX SHELL

- The shell is a text command processor to interface with the OS' "kernel"
- We will use the popular (yes!) `bash` shell
- To launch `bash` shell, do `Ctrl + Alt + t` (or use `Meta` key)

MEET THE UNIX SHELL

- The shell is a text command processor to interface with the OS' "kernel"
- We will use the popular (yes!) `bash` shell
- To launch `bash` shell, do `Ctrl + Alt + t` (or use `Meta` key)
- The shell automatically starts in your home directory `/home/yourname/`, also called ~ (important to remember!)

MEET THE UNIX SHELL

- The shell is a text command processor to interface with the OS' "kernel"
- We will use the popular (yes!) `bash` shell
- To launch `bash` shell, do `Ctrl + Alt + t` (or use `Meta` key)
- The shell automatically starts in your home directory `/home/yourname/`, also called ~ (important to remember!)
- Use the `Tab` key – very handy (try `ls` with `Tab Tab`)

MEET THE UNIX SHELL

- The shell is a text command processor to interface with the OS' "kernel"
- We will use the popular (yes!) `bash` shell
- To launch `bash` shell, do `Ctrl + Alt + t` (or use `Meta` key)
- The shell automatically starts in your home directory `/home/yourname/`, also called ~ (important to remember!)
- Use the `Tab` key – very handy (try `ls` with `Tab Tab`)
- Navigate commands you typed using the up/down arrows

MEET THE UNIX SHELL

- Other useful keyboard shortcuts:

`Ctrl + A`	Go to the beginning of the line
`Ctrl + E`	Go to the end of the line
`Ctrl + L`	Clear the screen
`Ctrl + U`	Clear the line before cursor position
`Ctrl + K`	Clear the line after the cursor
`Ctrl + C`	Kill whatever you are running
`Ctrl + D`	Exit the current shell
`Ctrl + right arrow`	Move cursor forward one word
`Ctrl + left arrow`	Move cursor backward one word

MEET THE UNIX SHELL

- Other useful keyboard shortcuts:

`Ctrl + A`	Go to the beginning of the line
`Ctrl + E`	Go to the end of the line
`Ctrl + L`	Clear the screen
`Ctrl + U`	Clear the line before cursor position
`Ctrl + K`	Clear the line after the cursor
`Ctrl + C`	Kill whatever you are running
`Ctrl + D`	Exit the current shell
`Ctrl + right arrow`	Move cursor forward one word
`Ctrl + left arrow`	Move cursor backward one word

- *Practice, Practice, Practice!*

sudo, AND INSTALLING SOFTWARE

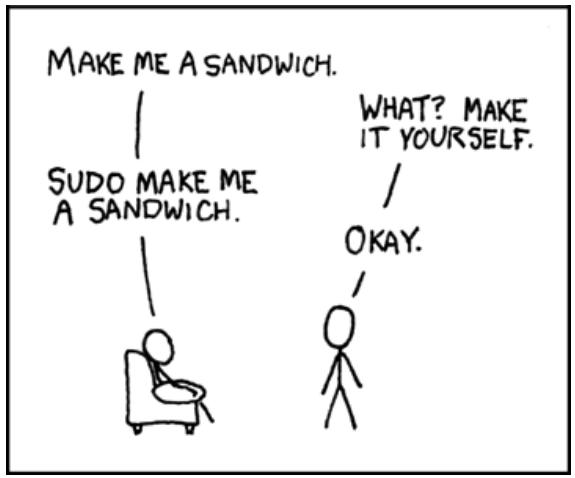
- You can install software in your `/home` directory
- In UNIX you originally had to login as `root` (administrator) to install software
- In Ubuntu, it is sufficient to add `sudo` (Super User DO) in front of a command:

```
$ sudo apt-get install geany geany-plugins geany-plugin-latex
geany-plugin-addons
```

sudo, AND INSTALLING SOFTWARE

- You can install software in your `/home` directory
- In UNIX you originally had to login as `root` (administrator) to install software
- In Ubuntu, it is sufficient to add `sudo` (Super User DO) in front of a command:

```
$ sudo apt-get install geany geany-plugins geany-plugin-latex
geany-plugin-addons
```



http://xkcd.com/149/

BASIC UNIX COMMANDS

`man [COMMAND]`	Show help page of a command.
`whoami`	Display your user-name.
`pwd`	Show the current directory.
`ls`	List the files in the directory.
`cd [NAMEOFDIR]`	Change directory. `cd ..` moves one directory up, `cd /` goes to the root, `cd ~` to home directory.
`cp [FROM] [TO]`	Copy a file.
`mv [FROM] [TO]`	Move or rename a file.
`touch [FILENAME]`	Create an empty file.
`echo "string to print"`	Print a string.
`rm [TOREMOVE]`	Remove a file.
`mkdir [DIRECT]`	Create a directory.
`rmdir [DIRECT]`	Remove an empty directory.
`wc [FILENAME]`	Count the number of lines and words in a file.
`sort [FILENAME]`	Sort the lines of a file and print the result.
`uniq`	Shows only unique elements of a list.
`cat [FILENAME]`	Print the file on the screen.
`less [FILENAME]`	Progressively print a file on the screen ("q" to exit).
`head [FILENAME]`	Print the first few lines of a file.
`tail [FILENAME]`	Print the last few lines of a file.
`history`	Show the last commands you typed.
`date`	Print current date.
`file [FILENAME]`	Determine the type of a file.
`passwd`	Change user password.
`chmod [FILENAME]`	Change file permissions.

An important digression

- It is time to start building your CMEE coursework directory structure
- Please follow these rules:
  - Do work in `CMEECourseWork`, located in a suitable place in your `/home` (make mama proud, keep your `home` organized!)

AN IMPORTANT DIGRESSION

- It is time to start building your CMEE coursework directory structure
- Please follow these rules:
  - Do work in `CMEECourseWork`, located in a suitable place in your `/home` (make mama proud, keep your `home` organized!)
  - OK, make `CMEECourseWork` now!

AN IMPORTANT DIGRESSION

- It is time to start building your CMEE coursework directory structure
- Please follow these rules:
  - Do work in `CMEECourseWork`, located in a suitable place in your `/home` (make mama proud, keep your `home` organized!)
  - OK, make `CMEECourseWork` now!
  - Each week's coursework should be in its respective directory: `CMEECourseWork/Week1`, `CMEECourseWork/Week2`, etc

AN IMPORTANT DIGRESSION

- It is time to start building your CMEE coursework directory structure
- Please follow these rules:
  - Do work in `CMEECourseWork`, located in a suitable place in your `/home` (make mama proud, keep your `home` organized!)
  - OK, make `CMEECourseWork` now!
  - Each week's coursework should be in its respective directory: `CMEECourseWork/Week1`, `CMEECourseWork/Week2`, etc
  - Each week's directory will contain directories called `Code`, `Data`, etc (later)

AN IMPORTANT DIGRESSION

- It is time to start building your CMEE coursework directory structure
- Please follow these rules:
  - Do work in `CMEECourseWork`, located in a suitable place in your `/home` (make mama proud, keep your `home` organized!)
  - OK, make `CMEECourseWork` now!
  - Each week's coursework should be in its respective directory: `CMEECourseWork/Week1`, `CMEECourseWork/Week2`, etc
  - Each week's directory will contain directories called `Code`, `Data`, etc (later)
  - You will bring `CMEECourseWork` and all it's contents under version control using Git (later)

AN IMPORTANT DIGRESSION

- It is time to start building your CMEE coursework directory structure
- Please follow these rules:
  - Do work in `CMEECourseWork`, located in a suitable place in your `/home` (make mama proud, keep your `home` organized!)
  - OK, make `CMEECourseWork` now!
  - Each week's coursework should be in its respective directory: `CMEECourseWork/Week1`, `CMEECourseWork/Week2`, etc
  - Each week's directory will contain directories called `Code`, `Data`, etc (later)
  - You will bring `CMEECourseWork` and all it's contents under version control using Git (later)
  - Have you made `CMEECourseWork` yet?

BACK TO BASIC UNIX COMMANDS

- Now let's do some fingerwork (start in `CMEECourseWork`): (*don't forget to use the tab key freely!*)

```
$ mkdir Week1
$ cd Week1
$ mkdir sandbox
$ cd Sandbox
bash: cd: Sandbox: No such file or directory
$ cd ..
$ rm sandbox
rm: cannot remove sandbox/: Is a directory
$ mv sandbox Sandbox # OR, "rm -r sandbox" (cuidado!)
$ cd Sandbox
$ pwd
$ ls
$ touch TestFile # OR, "touch TestFile.txt"
$ ls
$ mv TestFile TestFile2
$ rm TestFile2
```

BACK TO BASIC UNIX COMMANDS

- Now let's do some fingerwork (start in `CMEECourseWork`): (*don't forget to use the tab key freely!*)

```
$ mkdir Week1
$ cd Week1
$ mkdir sandbox
$ cd Sandbox
bash: cd: Sandbox: No such file or directory
$ cd ..
$ rm sandbox
rm: cannot remove sandbox/: Is a directory
$ mv sandbox Sandbox # OR, "rm -r sandbox" (cuidado!)
$ cd Sandbox
$ pwd
$ ls
$ touch TestFile # OR, "touch TestFile.txt"
$ ls
$ mv TestFile TestFile2
$ rm TestFile2
```

Note: *Henceforth, all code and output in a colored box, and " \ " means multi-line code (can be entered verbatim in bash/terminal)*

COMMAND ARGUMENTS

- Most UNIX accept arguments that modify their behavior
- e.g., `ls -l` (ls "minus"l) lists the files in longer format
- Some useful arguments:

`cp -r [DIR1] [DIR2]`	Copy a directory recursively (i.e., including all the sub-directories and files).
`rm -i [FILENAME]`	Remove a file, but asks first (for safety).
`rm -r [DIR]`	Remove a directory recursively (i.e., including all the sub-directories and files).
`ls -a`	List all files, including hidden ones.
`ls -h`	List all files, with human-readable sizes (Mb, Gb).
`ls -l`	List all files, long format.
`ls -S`	List all files, order by size.
`ls -t`	List all files, order by modification time.
`ls -1`	List all files, one file per line.
`mkdir -p Dir1/Dir2/Dir3`	Create the directory Dir3 and Dir1 and Dir2 if they do not already exist.
`sort -n`	Sort all the lines, but use numeric values instead of dictionary (i.e., 11 follows 2).

Redirection and pipes

- Output of programs can also be “redirected” to a file:
  - `>` Redirect output from a command to a file on disk. If the file already exists, it will be overwritten.
  - `>>` Append the output from a command to a file on disk. If the file does not exist, it will be created.

REDIRECTION AND PIPES

- Output of programs can also be “redirected” to a file:
  - `>` Redirect output from a command to a file on disk. If the file already exists, it will be overwritten.
  - `>>` Append the output from a command to a file on disk. If the file does not exist, it will be created.
- Examples (make sure you are in `Week1/Sandbox`):

```
$ echo "My first line." > test.txt
$ cat test.txt
$ echo "My second line" >> test.txt
$ ls / >> ListRootDir.txt
$ cat ListRootDir.txt #That's cool! Why?
```

REDIRECTION AND PIPES

- Output of programs can also be "redirected" to a file:
  - `>` Redirect output from a command to a file on disk. If the file already exists, it will be overwritten.
  - `>>` Append the output from a command to a file on disk. If the file does not exist, it will be created.
- Examples (make sure you are in `Week1/Sandbox`):

```
$ echo "My first line." > test.txt
$ cat test.txt
$ echo "My second line" >> test.txt
$ ls / >> ListRootDir.txt
$ cat ListRootDir.txt #That's cool! Why?
```

- We can also concatenate commands using "pipes" with "|"
- e.g., to count how many files are in root (/) directory:

```
$ ls / | wc -l #look up "info wc"
```

WILDCARDS

- We can use wildcards to find files based on their names (Again, in `Week1/Sandbox`!):

```
$ mkdir TestWild
$ cd TestWild
$ touch File1txt
$ touch File2.txt
$ touch File3.txt
$ touch File4.txt
$ touch File1.csv
$ touch File2.csv
$ touch File3.csv
$ touch File4.csv
$ touch Anotherfile.csv
$ touch Anotherfile.txt
$ ls
$ ls | wc -l
```

WILDCARDS

- We will use the following wildcards:
  - `?` Any single character, except a leading dot (hidden files).
  - `*` Zero or more characters, except a leading dot (hidden files).
  - `[A-Z]` Define a class of characters (e.g., upper-case letters).

WILDCARDS

- We will use the following wildcards:
  - `?` Any single character, except a leading dot (hidden files).
  - `*` Zero or more characters, except a leading dot (hidden files).

  `[A-Z]` Define a class of characters (e.g., upper-case letters).
- Now let's try:

```
$ ls *
$ ls File*
$ ls *.txt
$ ls File?.txt
$ ls File[1-2].txt
$ ls File[!3].*
```

USING *grep* I

- `grep` is a command that matches strings in a file (why is this useful?)
- It is based on regular expressions (more on this later)
- let's explore some basic usage of `grep`
- For a test file let's download a list of protected species from the UN website (to `Sandbox`):

```
$ wget http://www.cep.unep.org/pubs/legislation/spawannxs.txt #Cool!
$ head -n 50 spawannxs.txt #You will see "head" in R as well
```

- Now,

```
$ mkdir ../Data #Note the relative path "../"
$ mv spawannxs.txt ../Data/
$ cd ../Data
$ head -n 50 spawannxs.txt
```

USING *grep* II

- Note that now you have a `Data` directory
- What about falcons?

```
$ grep Falco spawannxs.txt
Falconidae Falco femoralis septentrionalis
Falconidae Falco peregrinus
Falconidae Polyborus plancus
Falconidae Falco columbarius
```

- Using `-i` make the matching case-insensitive:

```
$ grep -i Falco spawannxs.txt
Order: FALCONIFORMES
Falconidae Falco femoralis septentrionalis
Falconidae Falco peregrinus
Falconidae Polyborus plancus
Order: FALCONIFORMES
Order: FALCONIFORMES
Order: FALCONIFORMES
Falconidae Falco columbarius
```

USING *grep* III

- Now let's find the beautiful "Ara" macaws

USING *grep* IV

- But this poses a problem:

```
$ grep -i ara spawannxs.txt
Flacourtiaceae Banaras vanderbiltii
Order: CHARADRIIFORMES
Charadriidae Charadrius melodus
Psittacidae Amazona arausica
Psittacidae Ara macao
Dasyproctidae Dasyprocta guamara
Palmae Syagrus (= Rhyticocos) amara
Psittacidae Ara ararauna
Psittacidae Ara chloroptera
Psittacidae Arao manilata
Mustelidae Eira barbara
Order: CHARADRIIFORMES
```

- We can solve this by specifying `-w` to match only full words:

```
$ grep -i -w ara spawannxs.txt
Psittacidae Ara macao
Psittacidae Ara ararauna
Psittacidae Ara chloroptera
```

USING *grep* V

- And show also line(s) after the one matched, use `-A x`, where `x` is number of lines to use:

```
$ grep -i -w -A 1 ara spawannxs.txt
Psittacidae Ara macao

--
Psittacidae Ara ararauna
Psittacidae Ara chloroptera
Psittacidae Arao manilata
```

- Similarly, `-B` shows the lines before:

```
$ grep -i -w -B 1 ara spawannxs.txt
Psittacidae Amazona vittata
Psittacidae Ara macao
--
Psittacidae Amazona ochrocephala
Psittacidae Ara ararauna
Psittacidae Ara chloroptera
```

USING *grep* VI

- Use `-n` to show the line number of the match:

```
$ grep -i -w -n ara spawannxs.txt
216:Psittacidae Ara macao
461:Psittacidae Ara ararauna
462:Psittacidae Ara chloroptera
```

- To print all the lines that do not match a pattern, use `-v`:

```
$ grep -i -w -v ara spawannxs.txt
```

- To match one of several strings, use `grep "string1\|string2" file`.
- `grep` can be used on multiple files, all files, using wildcards for filenames, etc – explore as and when you need.

FINDING FILES WITH *find* I

- Easy to find files in UNIX using `find`!
- Let's test it (make sure you are in `Sandbox`, not `Data`!)

```
$ mkdir TestFind
$ cd TestFind
$ mkdir -p Dir1/Dir11/Dir111 #what does -p do?
$ mkdir Dir2
$ mkdir Dir3
$ touch Dir1/File1.txt
$ touch Dir1/File1.csv
$ touch Dir1/File1.tex
$ touch Dir2/File2.txt
$ touch Dir2/file2.csv
$ touch Dir2/File2.tex
$ touch Dir1/Dir11/Dir111/File111.txt
$ touch Dir3/File3.txt
```

- Now find particular files:

```
$ find . -name "File1.txt"
./Dir1/File1.txt
```

FINDING FILES WITH *find* II

- Using `-iname` ignores case, and you can use wildcards:

```
$ find . -iname "fi*.csv"
./Dir1/File1.txt
./Dir1/Dir11/Dir111/File111.txt
./Dir3/File3.txt
./Dir2/File2.txt
```

- You can limit the search to exclude sub-directories:

```
$ find . -maxdepth 2 -name "*.txt"
./Dir1/File1.txt
./Dir3/File3.txt
./Dir2/File2.txt
```

FINDING FILES WITH *find* III

- You can exclude certain files:

```
$ find . -maxdepth 2 -not -name "*.txt"
.
./Dir1
./Dir1/File1.tex
./Dir1/File1.csv
./Dir1/Dir11
./Dir3
./Dir2
./Dir2/File2.tex
./Dir2/File2.csv
```

- Find only directories:

```
$ find . -type d
.
./Dir1
./Dir1/Dir11
./Dir1/Dir11/Dir111
./Dir3
./Dir2
```

READINGS/RESOURCES

- Lots of UNIX tutorials out there. Try `http://software-carpentry.org/lessons.html` (Chapter "shell"). (watch video tutorials or read pdfs)
- IC library gives you with access to several e-books on UNIX, some specific to Ubuntu. Browse or search and find a good intro book: `http://imp-primo.hosted.exlibrisgroup.com/primo_library/libweb/action/search.do`
- There are also paper books – again, search library website
- List of UNIX commands along with man page: `www.oreillynet.com/linux/cmd/`
- Have you had a look at the LaTeXand Git resources I gave in the intro week? What do you think?

PRACTICAL: MAKE SURE THE BASICS WORK I

1. Review (especially if you got lost along the way) and make sure you can run and understand all the commands and get the expected outputs we have covered today
2. Make sure you have your directory organized with `Data` and `Sandbox` with the necessary files, under `CMEECourseWork/Week1`
3. Along with the completeness of the practicals/exercises themselves, you will be marked on the basis of how complete and well-organized your directory structure and content is – in all coming weeks as well

PRACTICAL: MAKE SURE THE BASICS WORK II

4. Finally, a small exercise:
   - Here is a more complicated bash command using two pipes:

```
$ find . -type f -exec ls -s {} \; | sort -n | head -10
```

   - What does this command do (Hint: try it on the test directories and files we created in `Sandbox`)?
   - And note that along with the `man` command, you can use the internet to get help on practically everything about UNIX!

ADDENDUM I

- What happens when you use up and down keys in terminal?
- If nothing, then you need to enable reverse searching history.
- To do so, open /texttt/etc/inputrc

```
$ sudo open /etc/inputrc
```

- Then, add the following to it:

```
"\e0A": history-search-backward\\
"\e[A": history-search-backward\\
"\e0B": history-search-forward\\
"\e[B": history-search-forward\\
"\e0C": forward-char\\
"\e[C": forward-char\\
"\e0D": backward-char\\
"\e[D": backward-char\\
```

- Then close current terminal, open new one, and try up and down keys again!

ADDENDUM II

- In terminal you can enter ”f” to open nautilus in current directory by doing the following:

```
$ sudo gedit ~/.bashrc
```

- Then add to the last line:
  `f = 'nautilus .'`

- Then restart terminal or in current terminal:

```
$ source ~/.bashrc
```

TOMORROW

- Shell scripts in UNIX
- Git (do each of you now have a bitbucket account?)
- LaTeX