Introduction to Linux

Christopher Werner

What you will learn

- Introduction of Linux and the command line
- Navigating through files and directories
- Working with files and directories
- Useful tools
- Loops
- Bash scripting
- SSH keys

You can also follow along at https://ichec-learn.github.io/intro-to-linux/

ICHEC
Irish Centre for High-end Computing

Introduction to UNIX

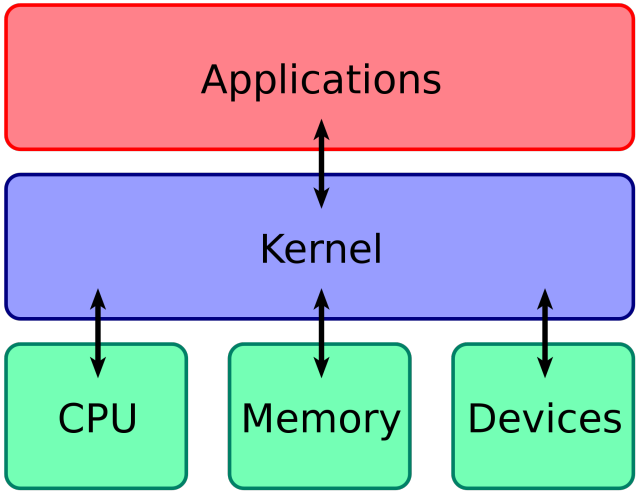
- Operating system
- Why UNIX?
  - Supercomputer operating system (100% of TOP500)
  - Reliable, more secure
  - Open source
  - Massive toolset for programming
  - Flexible
  - It's everywhere!
- UNIX systems also have GUIs
- Most popular varieties of UNIX are GNU/Linux, MacOS X

Starting a terminal

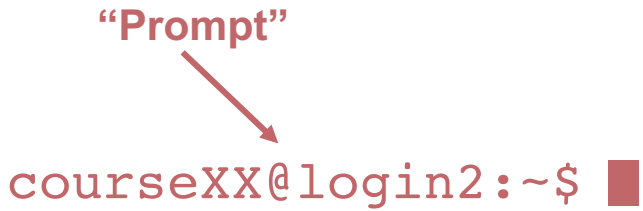
- Start up a terminal 
  - Mac users can use Terminal app
  - Windows use Git Bash or MobaXterm
  - NB: MobaXterm
    1. In Settings Tab → Terminal → Terminal features
       - Ensure “Paste using right click” is unticked
    2. In Settings Tab → General → Persistent home directory
       - Enter _ProfileDir_ into the box. Otherwise you may not be able to log into Kay for HPC course

Shell & Kernel

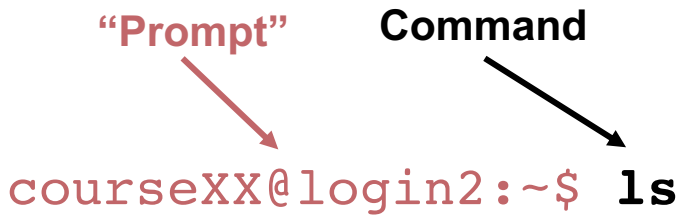
- **Shell** → Computer program that takes commands and gives them to OS to perform. It is the main interface between user and the Kernel
- **Kernel** → Computer program at the heart of the computer operating system
- Process for executing a command
  - Shell searches for program
  - Requests kernel to execute program
  - When process finishes, shell returns to prompt, waiting for further commands



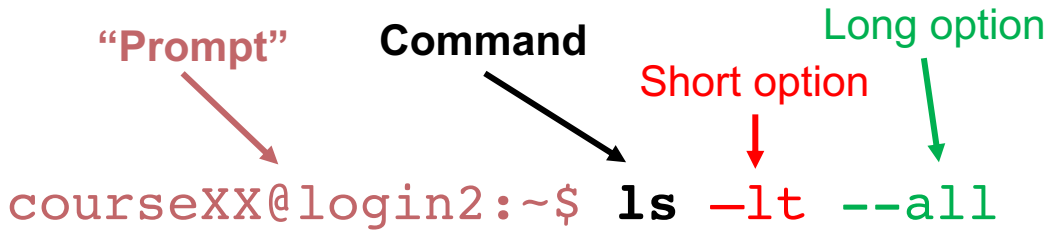
Command line



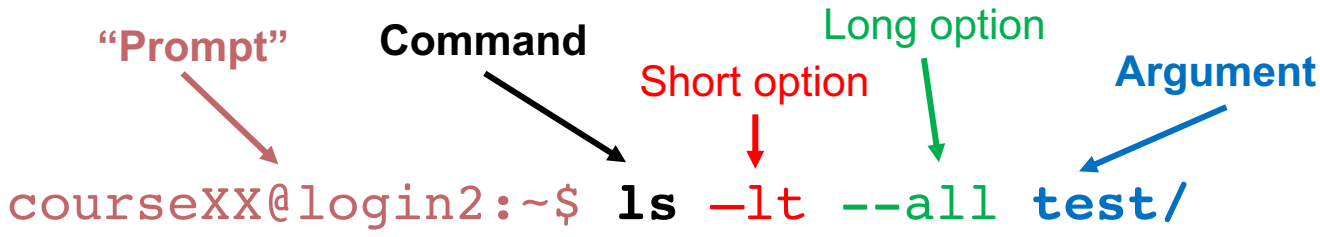
Command line



Command line

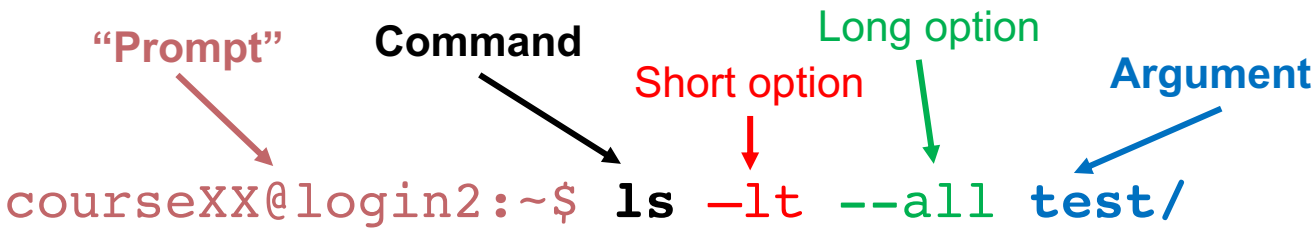


Command line



Command line

- Most command line programs need information on what they need to work on. These are command line arguments.
- Options come after a command, denoted with hyphen (-). These change the behaviour of the command
- They can be short (single letter) and grouped together or long form (full word)



Command line

- Order matters! 
- `la -s`
- `ls --a`
- `ls -`
- `ls -z`
- `-l ls`
- `ls -all --l prog/`
- `ls --all -l prog/`
- `ls -l all`
- `prog/ ls`
- `ls -l --all prog/`
- `ls -l--all prog/`

Command line: Errors & editing

- You will make mistakes, and the command line will tell you

```
cd: dir1: No such file or directory
```

- If you see a mistake at the beginning of a long command, don't worry you don't have to type it all out again.

Move to:	
left	`left-arrow`
right	`right-arrow`
Beginning of line	`ctrl-a`
End of line	`ctrl-e`
Previous command	`up-arrow`
Next command	`down-arrow`

Delete:	
Previous character	`Backspace`
Previous word	`ctrl-w`
Next character	`ctrl-d`
Rest of line	`ctrl-k`

- Computers are not humans, so if a command is not **precisely** correct it will complain!

ICHEC
Irish Centre for High-end Computing

Navigating through files and directories

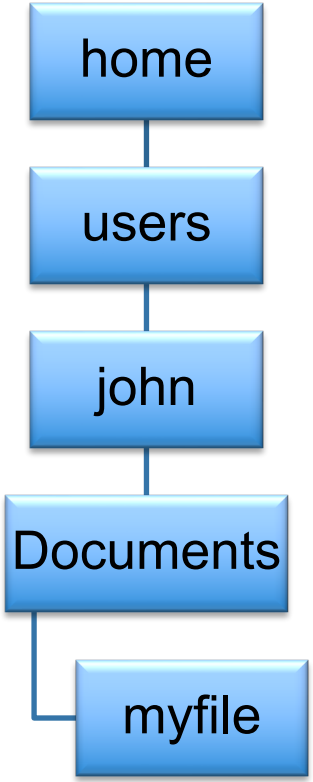
What is the directory tree and what is a path?

How do I move around?

How can I see my files and directories (folders)

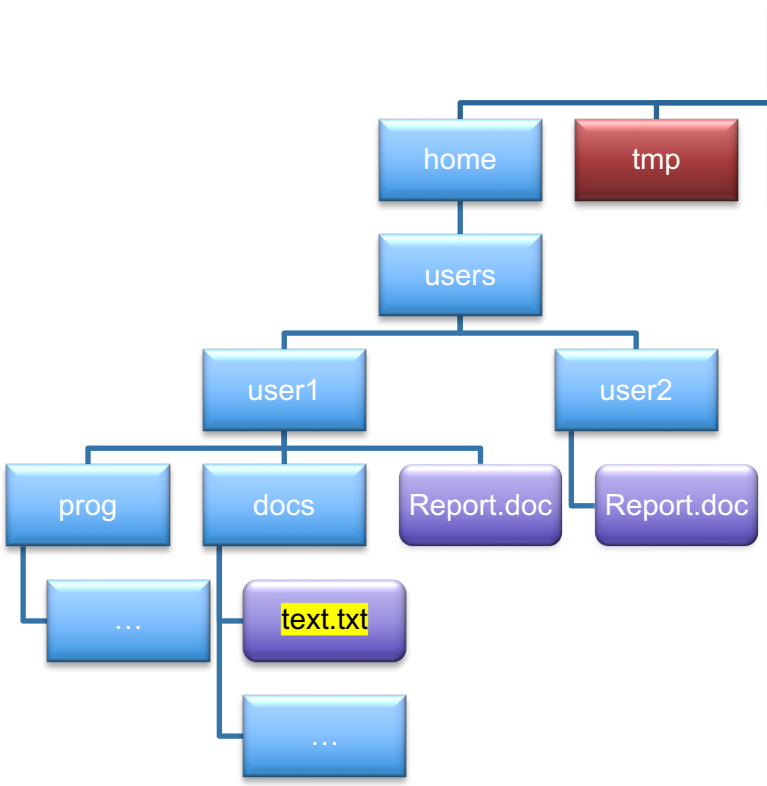
How do I use ‘flags’?

The directory tree



- In Unix, everything is a file or a process, even directories
- / has 2 meanings, root or a separator between directories
- Every file has a unique id formed from the file name and list of directories
- So even files with the same name have unique identifier in the file system

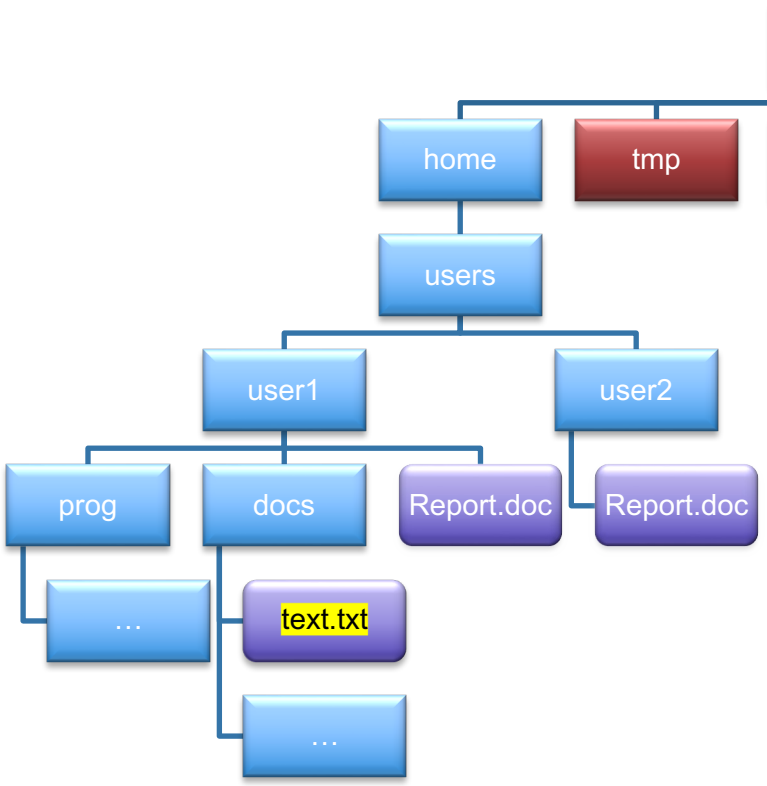
The directory tree



Locating yourself in the directory tree

- What is the absolute path to `text.txt`?
- What is the relative path to it if you are in `user1`?

The directory tree



Locating yourself in the directory tree

- What is the absolute path to `text.txt`?

`/home/users/user1/docs/text.txt`

- What is the relative path to it if you are in `user1`?

`./docs/text.txt`

Nav commands; `pwd`, `ls`, `touch`

- `pwd` → Print working directory
  - Options:
    - `-L` = Symbolic path
    - `-P` = Actual path
- `touch` → Creates an empty file
  - `touch myfile.txt` = Creates an empty file called myfile.txt
- `ls` → list (files and directories)
  - Options:
    - `-a` = Show hidden files
    - `-t` = Sort by modification time
    - `-l` = Long, detailed list
    - `-d` = Express directory as file
    - `-h` = Human readable
  - Example: `ls –thl docs/`

Nav commands; cd

- `cd` → change directory
  - Operations
    - `cd mydir` = move into mydir directory
    - `cd ..` = move back 1 directory
    - `cd ../../` = move back 2 directories
    - `cd ~` OR `cd` = move to user home directory
    - `cd -` = move to previous directory
    - `cd /` = move to root directory

Practice with cd

1. Use cd change plus the following and see what happens
   - `..`
   -
   - `.`
   - `new`
   - `../../`
   - `~`

Practice with cd

1. Match up the situations below with their expected output. Assume you are in a directory which contains a single directory, Documents

1	`cd`	Moves back one directory	A
2	`cd.`	No such file or directory	B
3	`cd Desktop`	Moves you back two directories	C
4	`cd ../..`	Moves you back into the “home” directory	D
5	`cd ..`	Does nothing, as you are already here	E
6	`cd Documents`	command not found	F
7	`cd ~`	Moves you back into the “home” directory	G
8	`cd .`	Moves you into Desktop	H

ICHEC
Irish Centre for High-end Computing

Practice with cd

1. Match up the situations below with their expected output. Assume you are in a directory which contains a single directory, Documents

1	`cd`	Moves you back into the “home” directory	D
2	`cd.`	command not found	B
3	`cd Desktop`	Moves you into Desktop	H
4	`cd ../..`	Moves you back two directories	C
5	`cd ..`	Moves back one directory	A
6	`cd Documents`	No such file or directory	F
7	`cd ~`	Moves you back into the “home” directory	G
8	`cd .`	Does nothing, as you are already here	E

Help commands;

- `man` → displays user manual of any command, including options
  - Example: `man ls` (press q to escape)
- `--help` → displays more info on how to use command. Similar to `man`
- `history` → shows last 500 commands entered
  - Pro tip: `!100` returns the 100th command in your history, great for complicated commands
  - ctrl-r shortcut to reverse search command in history

ICHEC
Irish Centre for High-end Computing

Working with files and directories

- How do I create directories
- How can I copy and move files?

Clone a repo! 

```
git clone https://github.com/ICHEC-learn/intro-to-linux.git
```

Making and removing; `mkdir`, `rm`

- `mkdir` → make a directory & `rmdir` → remove empty directory
  - Options:
    - `-m` = set permissions
    - `-p` = path name
  - Usage
    - `mkdir directory01/ directory02/`
    - `rmdir directory01/`

Creating and removing a directory 

- Create a new directory
- Change into the directory
- Create a new file
- Move back one directory
- Remove it

Making and removing; `mkdir`, `rm`

- `rm` → remove (files and directories): deleting is permanent
  - Options:
    - `-r` = delete directories and subdirectories
    - `-i` = prompt before use of rm (Y or N)
  - Usage

```
rm test.txt file.pdf
```

Using rm safely 

Create a new file using `touch newfile`, and then try and remove it using the `-i` flag.

Moving and copying; mv, cp

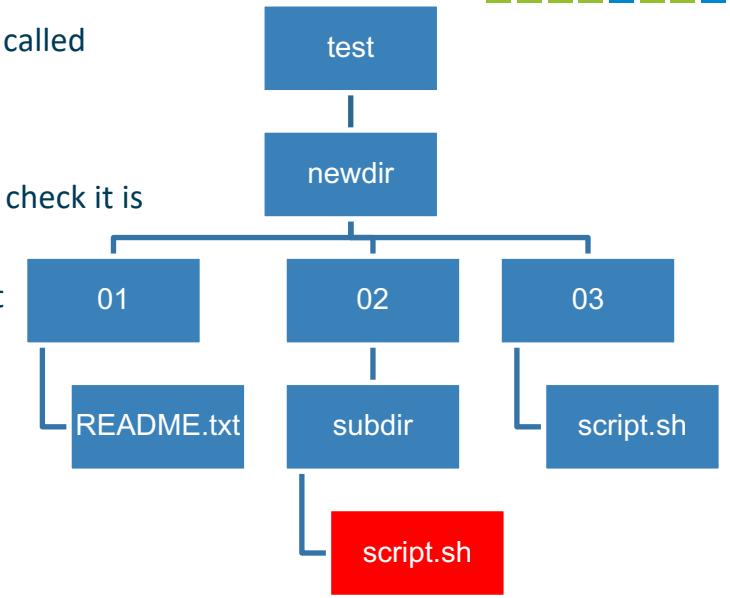
- `mv` → move files / change name of file/directory
  - Formats
    - `mv test.txt dir/` = move test.txt into dir directory
    - `mv test.txt ../` = move test.txt back 1 directory
    - `mv test.txt test01.txt` = rename test.txt to test01.txt
    - `mv dir/ dir01/` = rename dir/ to dir01/
    - `mv -f 01.txt test.txt` = force renames 01.txt to test.txt
    - `mv –v 01.txt test.txt` = renames file, verbose option
    - `mv other/file.txt .` = move file.txt to current directory

- `cp` → copy files or directory
  - Examples:
    - `cp test.txt dir/test.txt` = `mv test.txt dir/test.txt`
    - `cp –r dir01/ dir02/` = copy directory (needs `–r` switch)

Creating a directory hierarchy

1. Navigate to your home directory, confirm it with `pwd`
2. List the files in this directory, then list all hidden files with the `–a` flag
3. Create a new directory called `01` and `cd` into it
4. Create a file called `README.txt`
5. Change back one directory to your home directory
6. Create 2 more directories `02` and `03`. In `02` create a new directory called `subdir`
7. In `03`, create a file called `script.sh`
8. Copy `script.sh` to the `02/subdir` folder, then move into it to check it is there
9. Remove the copied file using the `–i` flag for a confirmation prompt
10. Continue experimenting with navigating and working with files

- **Tips**: At every step use `pwd` and `ls` to confirm you know where you are and what files are located there
- **Commands**: `pwd, ls, touch, cd, mkdir, rm, mv, cp, man` *command*



ICHEC
Irish Centre for High-end Computing

Useful Tools

Writing to and viewing files

Tar and zip files

Wildcards

Permissions

Searching

Sed and awk

Writing to and Viewing files

- `echo` → prints a message to the screen
  - Formats
    - `echo "Hello"` = prints "Hello" to the screen
    - `echo "Hello" > hello.txt` = writes "Hello" to a file called hello.txt
- `cat` → print contents of a file to the screen
  - eg:
    - `cat hello.txt` = prints out "Hello"
- `less` → prints contents of a file to a separate window
  - eg:
    - Less hello.txt = shows content
- `more` → Combines features of `cat` and `less`
- `head` → lists first 10 lines of a file
- `tail` → lists last 10 lines of a file

Familiarize yourself with the viewing commands 

File commands; `wc`, `diff`, `sort`

- `wc` → word count, as well as lines and characters
  - Usage: `wc text.txt` = number of lines, characters, words
  - Options:
    - `-c` = number of bytes
    - `-l` = number of lines/rows
    - `-m` = number of characters
    - `-w` = word count
- `diff` → used to view differences between files; (<) : file 1; (>) = file 2
  - Usage: `diff file1 file2`
  - Options:
    - `-a` = add
    - `-c` = change
    - `-d` = delete
  - The output will give instructions using a, c, d options to indicate which lines need changing to make the files identical.
- `sort` → sorts file but does not change file
  - Usage: `sort in.txt > out.txt` = puts sorted output to new file
  - Options
    - `-n` = numerical sort
    - `-r` = reverse sort
    - `-k 2n` = sort based on 2nd col.
    - `-u` = remove duplicates
    - `-o` = write output
    - `-M` = sort by month

Special file operations; `tar`

- The `tar` command creates maintains, modifies and extracts files archived in the tar (tape archive)
- Why tar? → Stores multiple files in one single file
  - Options:
    - `c` = create archive file
    - `x` = extract archive
    - `v` = show file progress
    - `f` = filename
    - `t` = view file contents
    - `j` = filter archive through bzip2
    - `z` = filter archive through gzip
    - `r` = append/update files to archive
    - `W` = verify archive file
  - Useful Commands (*works for `.gz`, `.bzip2` extensions)
    - `tar –cvf dir.tar dir/` = create normal tar archive file
    - `tar –cvzf dir.tar dir/` = create gzipped tar archive file
    - `tar –xvf dir.tar (-C /home/)` = untar files (in other directory) (*)
    - `tar -tvf dir.tar` = list content of archive (*)
    - `tar -rvf dir.tar` = add text.txt to archive (*)
    - `tar –xvf dir.tar text.txt` = extract text.txt from archive
    - `tar -zxvf dir.tar.gz –wildcards '*.txt'` = extract group of files using wildcards

ICHEC
Irish Centre for High-end Computing

Special file operations; `gzip`

- The **`gzip`** command compresses files. Each single file is compressed into a single file
- Original file is deleted using **`gzip`**, use **`–c`** option to write compressed file to stdout
- The **`gunzip`** command unzips the file. You can also use **`gzip –d`**
  - Usage:

`gzip file`	= zips file, deleting it, creating `file.txt.gz`
`gzip –c file gzips/file`	= move `test.txt` back 1 directory
`gzcat file.txt.gz`	= view contents of `file.txt.gz`
`gunzip file.txt.gz`	= undo the effects of `gzip`
`gzip -d file.txt.gz`	= undo the effects of `gzip`
`gunzip –c a.txt | more`	= Write uncompressed content to stdout and pipe to `more` for an easy read

- If zipping a **`tar`** file, you can name **`file.tar.gz`** as **`file.tgz`** or use a specified **`tar`** command

ICHEC
Irish Centre for High-end Computing

`tar`-ing and `gzip`-ing a directory

Use `tar` plus its flags to create a `.tar` archive of the `wildcards/` directory. Check the contents of the archive. Now zip the archive using `gzip`.

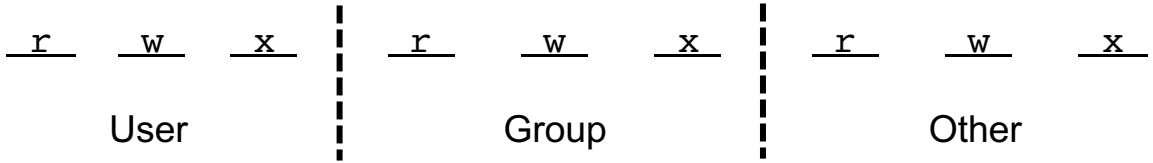
Finally, unzip and untar the archive.

Wildcards; *, ?, [ ]

- For a directory with: `001.txt  002.txt … nnn.txt other.dat`
- `*` → matches one or more occurrence of any character
  - Examples:
  - `cp *.txt docs/txt_files` `ls –l 0*`
- `?` → matches single occurrence of any character
  - Examples:
  - `cp 0?.txt docs/txt_files` `ls –l 0?.txt`
- `[ ]` → any character inside the square brackets
  - Examples:
  - `cp 0[12]1.txt docs/txt_files` `ls –l [02468]1.txt`
- `*?[ ]` → combining wildcards
  - Examples:
  - `cp ??1* docs/txt_files` `ls –l [1-6][4]?.txt`
- Practice using the different wildcards to remove, or list out the different files in the wildcards directory 💻

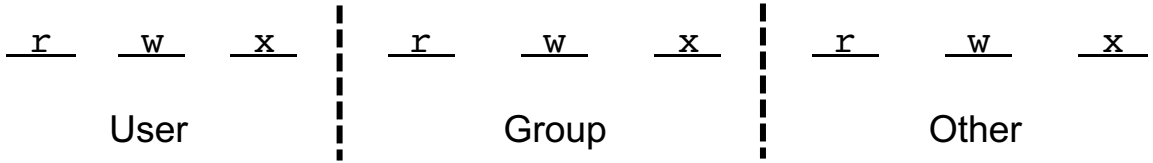
Permissions

- All files have owners (machine user) and group owners (group on machine)
- The long listing format shows:



Permissions

- All files have owners (machine user) and group owners (group on machine)
- The long listing format shows:



- Create a file using touch and see its full details using `ls -l`
  - `r` → file can be opened for reading
  - `w` → file can be opened for writing
  - `x` → file can be opened for execution

Changing Permissions with `chmod`

- `chmod` (change mode) modifies permissions of user (u), group (g) and other (o) to read (r), write (w) and execute (x) the file
  - `chmod [ugo][+-][rwx] file`
- The same can be done with directories… eg home directory
  - `chmod go-rwx ~`

Changing Permissions with chmod

- `chmod` (change mode) modifies permissions of user (u), group (g) and other (o) to read (r), write (w) and execute (x) the file
  - `chmod [ugo][+-][rwx] file`
- The same can be done with directories… eg home directory
  - `chmod go-rwx ~`
- There are also number notations, each one referring to u, g, o;

0 == ---	2 == -w-	4 == r--	6 == rw-
1 == --x	3 == -wx	5 == r-x	7 == rwx

- What's the equivalent numeric command?

OWNER			GROUP			OTHER		
r	w	x	r	w	x	r	w	x
1	1	1	0	0	0	0	0	0

Changing Permissions with chmod

- `chmod` (change mode) modifies permissions of user (u), group (g) and other (o) to read (r), write (w) and execute (x) the file

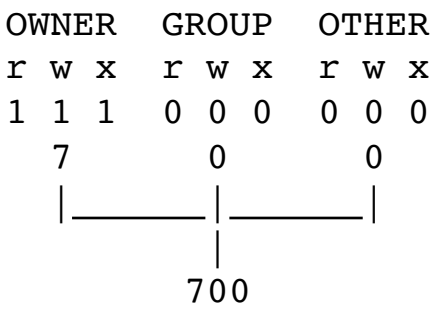
```
chmod [ugo][+-][rwx] file
```

- The same can be done with directories… eg home directory

```
chmod go-rwx ~
```

- There are also number notations, each one referring to u, g, o;

0 == ---	2 == -w-	4 == r--	6 == rw-
1 == --x	3 == -wx	5 == r-x	7 == rwx



- What's the equivalent numeric command?

```
chmod 700 ~
```

- Changing file permissions 
  - Create a new file, add executable for user and group, remove reading permissions for other

ICHEC
Irish Centre for High-end Computing

Changing Permissions with chmod

Who has what permissions? 

1. rwxrwxr--
2. r--r--r--
3. 755
4. 700
5. rwxrw-r--
6. chmod u+x file
7. chmod go-wx file

Pattern searching using `grep`

- One of the most commonly used and essential commands to master
- Used to find substrings in large bodies of text

```
grep 'int' main.c
```

finds all instances of `int` in the program `main.c` case sensitive

- Options

`-i`	=	case insensitive
`-n`	=	display line occurrence
`-v`	=	omit unwanted strings
`-r`	=	recursive (current/subdirectory)
`-l`	=	all files which contain a string
`-C<num>`	=	num of lines before and after match
`--color=auto`	=	add colour display to selection

Pipes

Which is the best option for ease of use?

1 Tesla Roadster

10 Nissan Leafs

Better to combine smaller commands into a more powerful and useful one

Pipes (|) remove unnecessary temporary files, and send output of one command to the input of another. The syntax is;

```
command_1 | command_2 | ... | command_n
```

Pipes – how do they work?

Which is more powerful and convenient?

```
$ cat -ns text.txt > newfile.txt
$ less newfile.txt

$ cat –ns text.txt | less
```

Not only is this cleaner, but there is less chance of overwriting important files

Very powerful tool particularly when combined with the **`grep`** command

ICHEC
Irish Centre for High-end Computing

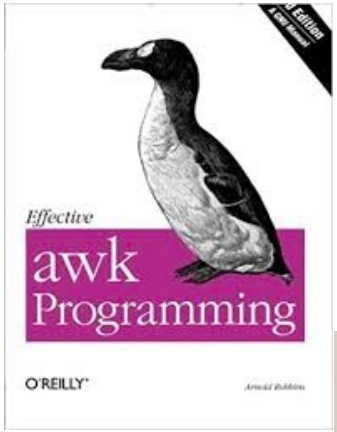
`sed` – UNIX's stream editor

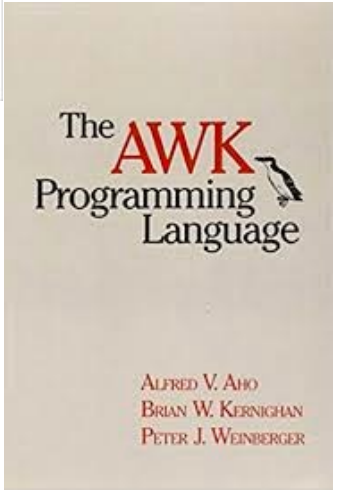
- `sed` is another handy tool in UNIX which can be used for;
  - Searching
  - find and replace
  - Insert / delete
- The format of the sed command is;
  - `sed 'opt/act/flag' file`

- Replacing text: `sed 's/string/replacements/' file`
- Replace nth occurrence: `sed 's/str/repl/2' file`
- Print replaced lines: `sed –n 's/str/repl/p' file`
- Delete 5th line: `sed '5d' file`

`awk` – language within UNIX

- So far you've learnt the language of UNIX, **`awk`** is its own language
- Enables users to write statement sized programs for data extracting and reporting
- Basics only covered here, plenty of commands in plenty of books.
- Awk program is a series of rules in the form;
  - `awk '(CONDITION) {ACTION}' file`





ICHEC
Irish Centre for High-end Computing

awk – language within UNIX

- awk can be used for anything from printing to complex mathematical statements
- It can also be used in bash scripting over multiple lines
- Examples include:

Command	Meaning
`awk '{print}' t.txt`	Prints out contents of t.txt. Works like cat
`awk '/example/ {print}' t.txt`	Searches for lines containing 'example'
`awk '{print $1,$3}' t.txt`	File split by whitespace delimiter $0 = whole line
`awk '{if ($1 > 100) print $1}' t.txt`	Prints any value greater than 100 in t.txt

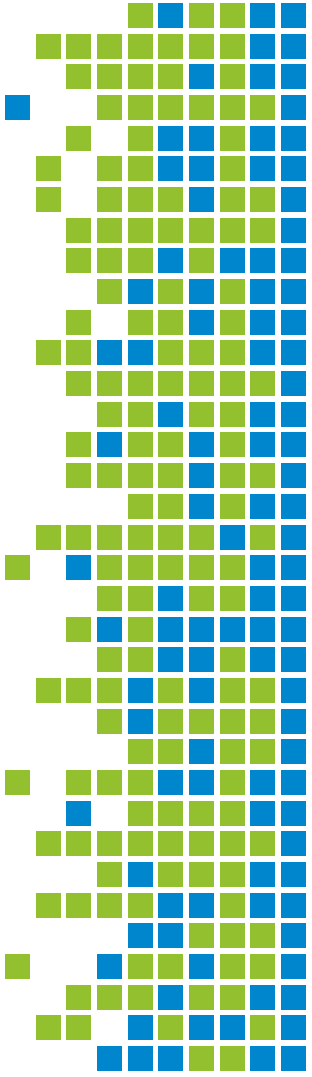
ICHEC
Irish Centre for High-end Computing

Loops and conditionals

Variables

Loops

Conditionals



ICHEC
Irish Centre for High-end Computing

Environment Variables in bash

System-wide variables inherited by all child processes and shells. **`env`** shows environment variables in your session

Denoted with the (**`$`**) symbol. What happens when you type;

```
echo $HOME
```

You can set a temporary environment variable for your current session;

```
export VARNAME="my value"
```

You can set a permanent variable to future sessions, add it to **`.bashrc`**

This process is also known as aliasing

ICHEC
Irish Centre for High-end Computing

Shell Variables in bash

Shell variables are named symbols that represent strings or numeric value, and only apply to current shell instance

Assigning/unassigning variables is easy (watch for whitespace)

```
pi=3.142
echo $pi
unset pi
```

Crucial in bash scripting

ICHEC
Irish Centre for High-end Computing

Variables in bash

- Variables are not protected in longer strings
- From previous example;

```
$ pi=3.14
$ echo $pie          (won't work, we didn't declare pie!)
$ echo ${pi}e        (this will)
3.14e
```

- Arithmetic can also be performed using either `let` or `(( ))`

```
$ x=100; y=50
$ let x++; ((y--))
$ echo $x $y
101 49
```

- There is no floating point arithmetic in bash, but you can use `bc`

```
$ echo '100/3' | bc –l
33.333
```

For loops in bash

- Allows code to be repeatedly executed, and particularly useful for dealing with lots of files
- The `seq –w` command allows us to produce equal width, i.e. sequential output (useful for filenames)

```
for i in a b c d e;
do
    echo $i
done

for x in $(seq -w 1 10);
do
    touch ${x}.txt
done
```

ICHEC
Irish Centre for High-end Computing

Functions in bash

A function is a series of commands that can be called numerous times. It makes code more readable and user friendly

Two formats

SCRIPT	

function_name() {
Commands
}
	function func_name() {
	commands
	}

COMMAND LINE	
function_name() {commands; }	function func_name() {commands; }

ICHEC
Irish Centre for High-end Computing

Comparison Operators

Binary comparisons compare two variables or quantities. There are different operators for strings and integers

Integer Comparison		String Comparison
`-eq`		`= / ==`
`-ne`		`!=`
`-lt / <`		`<`
`-le / <=`		`<=`
`-gt / >`		`>`
`-ge / >=`		`>=`
	String is null	`-z`
	String not null	`-n`
`conditional ["$a" –lt "$b"] / (("$a" < "$b"))`		

ICHEC
Irish Centre for High-end Computing

Conditional statements in bash

Both `if` and `case` statements are supported. The `case` statement is for more complex use, and best saved for your actual code

```
if [ $x –gt 100 ]; then
    echo ‘greater than 100.’

elif [ $x == 100 ]; then
    echo ‘equal to 100.’
else
    echo ‘less than 100.’
fi
```

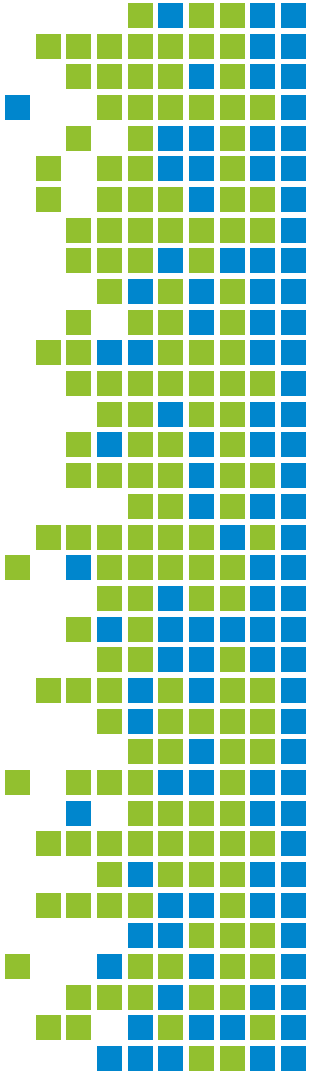
ICHEC
Irish Centre for High-end Computing

Bash scripting

Text editors

Writing a bash script

Conditionals



ICHEC
Irish Centre for High-end Computing

Scripting languages within UNIX

A scripting language is a non-compiled programming language

Usually written in text files and interactively on the command line

Examples: `bash`, `csh`, `ksh`, `zsh`, `perl`, `python`

Here we will look at `bash`, which is saved as a text file with `.sh` extension

ICHEC
Irish Centre for High-end Computing

Text editors

Trying to write a programming language like C or Fortran in the command line won't work!!

Python can work in the command line, but not as a file!

Use text editors for opening, viewing and editing files

Depending on the coding language, different extensions are needed

Examples include; gedit, vim, nano, emacs

ICHEC
Irish Centre for High-end Computing

Text editors – Face-off

	nano	vim	emacs
Pros	• No learning curve • Easy to use • Good for simple edits	• Effective editing of text • Super powerful, complicated edits made easy • Highly effective, keyboard shortcuts for everything	• Customisable, extendable • Powerful edits • Edit files & browse web • Mature integration with tools
Cons	• Complicated edits difficult • No powerful features (macros, simultaneous multiple files)	• Overkill for simple edits • Steep(ish) learning curve (see vimtutor)	• Hard to customise using Lisp, steep learning curve • Not available everywhere
Verdict	• Great for beginners and simple edits.	• Ideal and go-to for programmers. Best choice when mastered.	• For those who want more than a text editor as Emacs can be an environment

* For commands for each see practical sheet

First bash script

- Run the following command;
  - `echo 'hello from the command line'`
- Create a file and use the text editor to edit

```
#!/bin/bash

# Comments denoted by hashtag

echo 'hello from bash'
```

- Run it using

```
bash ./test.sh
```

Running a file with a bash script

- Create a python “Hello world” file;

hello.py

```
print(“Hello world!”)
```

- Use your bash file to run the python file

test.sh

```
#!/bin/bash

# Comments denoted by hashtag

python hello.py
```

- Run it using

```
bash ./test.sh
```

Running a file with a bash script

- Create a python “Hello world” file;

hello.py

```
print(“Hello world!”)
```

- Use your bash file to run the python file

test.sh

```
#!/bin/bash

module load conda/2
source activate python3

python hello.py
```

```
bash ./test.sh
```

ICHEC
Irish Centre for High-end Computing

SSH Keys

Text editors

Writing a bash script

Conditionals

Shell vs SSH (Secure Shell)

Shell

- Computer program that takes commands and gives them to OS to perform. It is the main interface between user and the Kernel
- Most Linux systems (incl. Kay) use bash as the default shell.

```
yourcomputer:dir username$ echo "Hello"
```

SSH (Secure Shell)

- The command `ssh` (secure shell) follows SSH protocol - a remote admin protocol that authenticates a remote user
- Easy in Mac/Linux, not as trivial in Windows

```
ssh {user}@{host}
```

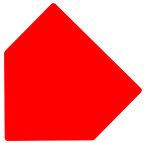
```
ssh yourusername@kay.ichec.ie
```

Public and Private Keys

Generate the ssh-key

Private: id_ed25519

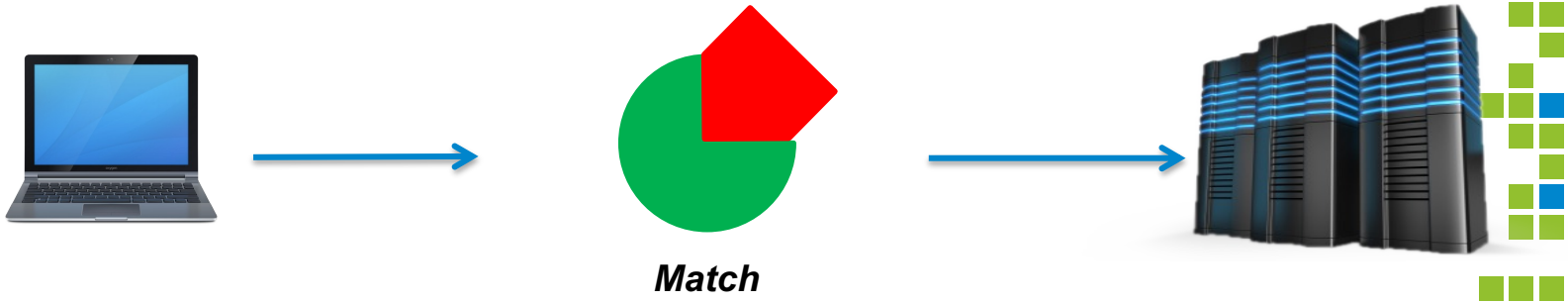
Public: id_ed25519.pub

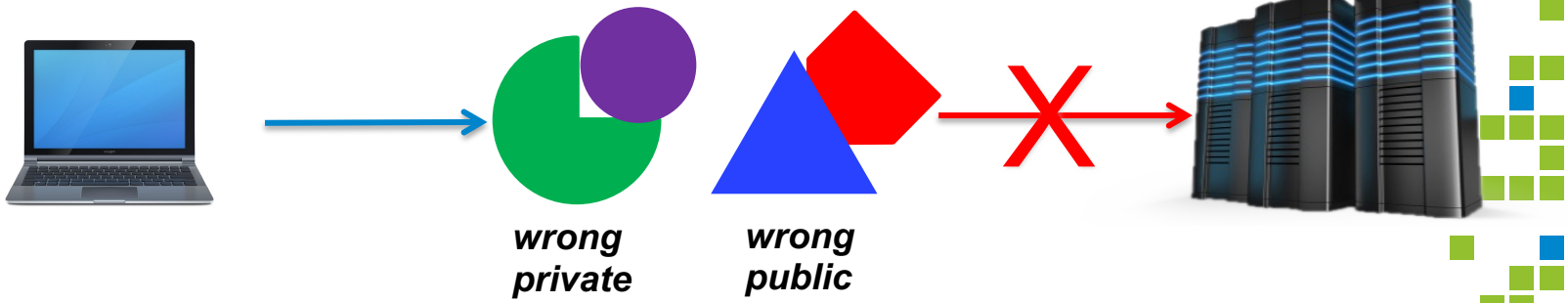


$USER_PROFILE/.ssh/id_ed25519

$HOME/.ssh/authorized_keys

Public and Private Keys





ICHEC
Irish Centre for High-end Computing

Creating an SSH key-pair

- Once an SSH key-pair has been generated, I need to…
  1. Submit my private key, as the account on the supercomputer will be private to me
  2. Submit my public key, as the account needs
  3. I don't need to do anything, as I have both of them
  4. Submit both my public and private keys, as both will be needed for me to log in
  5. Generate a new SSH key pair in a different directory

Creating an SSH key-pair

- Navigate to your home directory and type

```
ssh-keygen –t ed25519
```

- Choose a password you can remember

- Now try logging into Kay using your username in the document in the chat

```
ssh courseXXX@kay.ichec.ie
```

ICHEC
Irish Centre for High-end Computing

Summary

- Introduction to basic Linux commands
  - `pwd`, `cd`, `ls`, `mkdir`, `rmdir`, `rm`, `mv`, `cp` , `man`
- Navigation of directories
- Creating files and directories
- Pattern searching using `grep`
- Using `vim`, and creation of bash script
- Running a basic bash script
- Setting up ssh keys