Introduction to UNIX

John A. Juma

Animal and Human Health

International Livestock Research Institute (ILRI)

March, 2023

j.juma@cgiar.org

Objectives

- What is UNIX
- Basic UNIX commands
- Redirection/Pipes
- File system security/permissions
- Other useful UNIX commands

What is UNIX

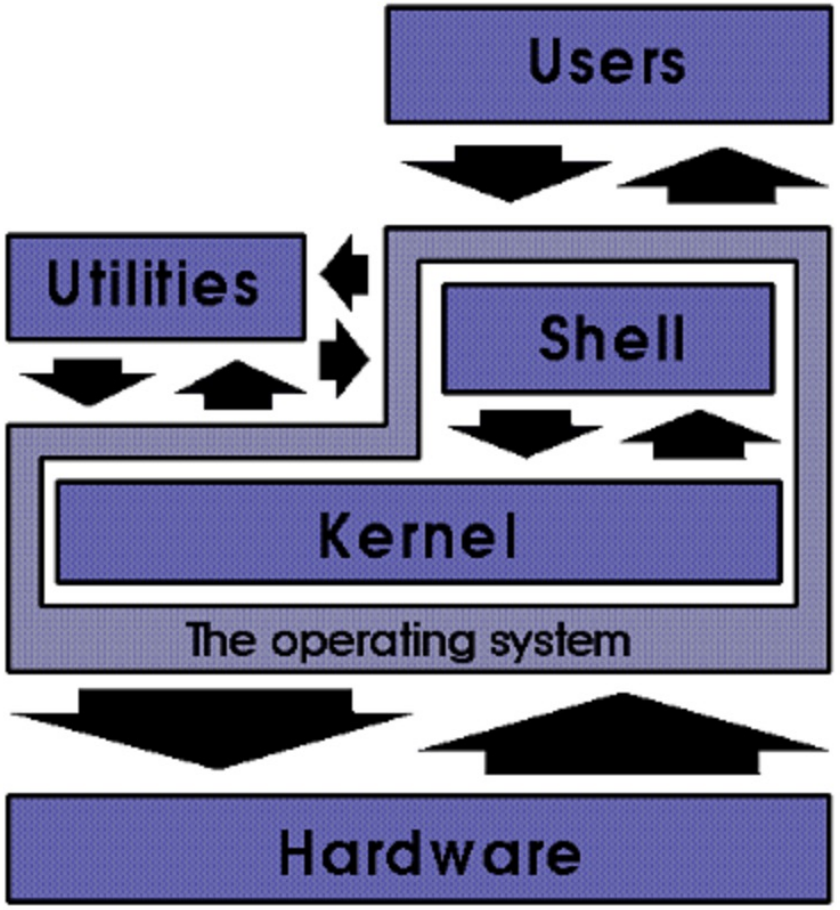
UNIX is an operating system (OS) first developed in the 1960s by Ken Thompson and Dennis Ritchie.

Still under constant development ever since.

Features

- Stable, multi-user, multi-tasking system for servers, desktops and laptops.
- Flexible and easy to adopt to specific needs. UNIX philosophy – 'small is good' - each program is designed to do one job well.
- Written in a machine independent language – UNIX can run on various hardware (laptops, desktops, super-computers).
- Open source/freely available with open-source codes.
- UNIX systems also have a graphical user interface (GUI) like Microsoft Windows which provides an easy-to-use environment.

UNIX Anatomy



UNIX is multi-layered system

Kernel

Bottom layer of the system that provides a means for starting application programs (often also called user programs).

The kernel facilitates four basic types of services:

- creation and management of processes
- a filesystem
- communications
- a means to start the system

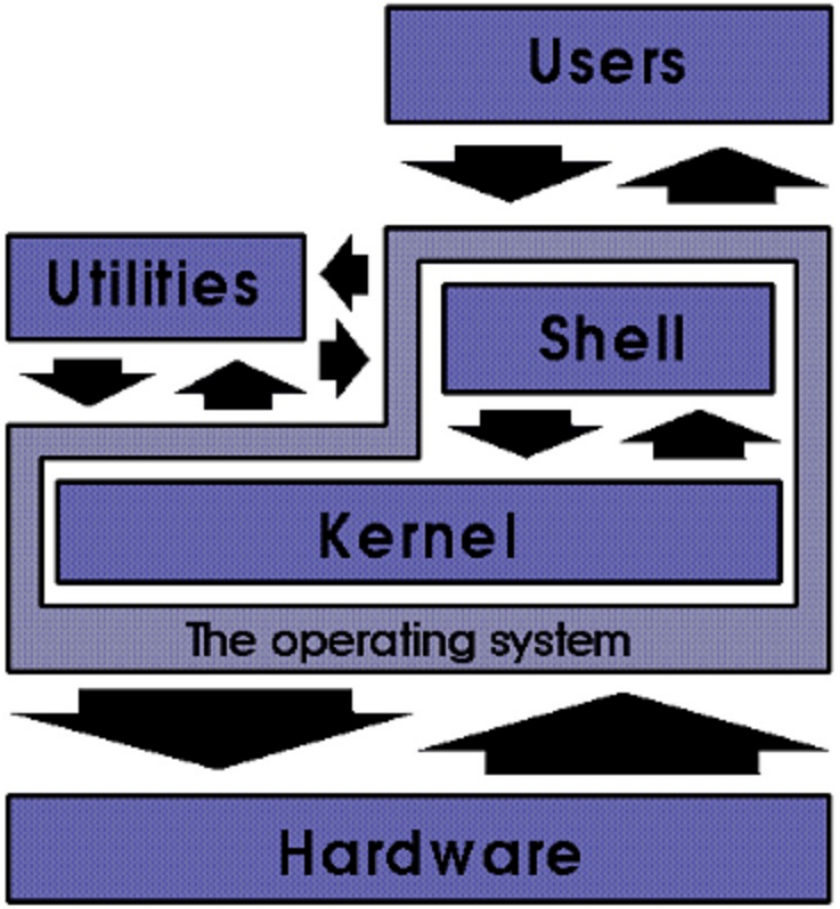
Kernel functions, such as **allocation of memory** and **CPU**, are performed **without being explicitly requested by user processes**.

Other functions of the kernel, such as **resource allocation** and **process creation and management**, are initiated by **requests from processes**.

These requests from processes come in the form of ***system calls***. A system call can be thought of as a low-level request to the operating system (e.g., kill, fork, exec, exit, close)

Source: http://ibgwww.colorado.edu/~lessem/psyc5112/usail/concepts/anatomy-of-unix/anatomy.html

UNIX Anatomy



Process

In UNIX, a running program is called a **process**. A program that a process can be created from is called and **executable**.

The Shell

UNIX system user accesses the services of the kernel through an interface called a **shell**.

The shell is a **command interpreter** that allows the user to initiate processes to perform a nearly infinite variety of tasks.

Utilities

Utilities are programs that perform system functions. Utility can also refer to a command that is used to do work of some sort, such as *mv* to move files or directories.

Source: http://ibgwww.colorado.edu/~lessem/psyc5112/usail/concepts/anatomy-of-unix/anatomy.html

The Terminal

Originally, computers did not have **integrated hardware** for humans to use – e.g., they did not have integrated screens, keyboards, mice, track pads, etc.

Such **human computer interface devices** only arose with the advent of small personal computers (PC's) designed and built largely for direct human use.

Source: https://jappavoo.github.io/UndertheCovers/textbook/unix/terminal.html

ASCII Terminal

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

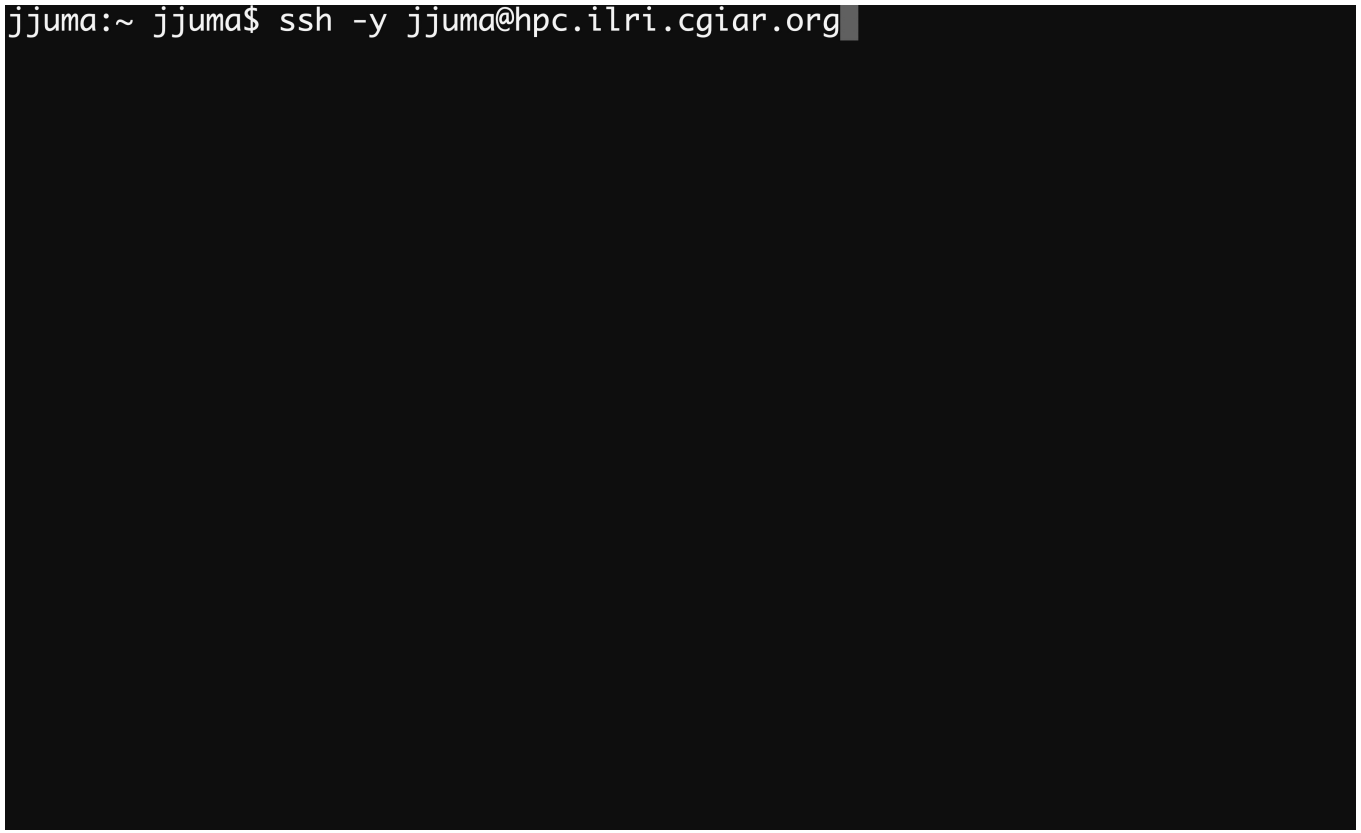
ASCII terminals can translate ASCII encoded text data to display, e.g., "print", the appropriate character on the screen.

For example, if the software running on the computer sent the number **65** to the terminal, the terminal would display capital '**A**' at the current "cursor" position.

Terminals were the "natural" evolution of the classic typewriter.

Source: https://jappavoo.github.io/UndertheCovers/textbook/unix/terminal.html

Terminal Emulators



- Apple's MacOSX Terminal
- Window's Terminal
- Others: PuTTY, MobaXTERM

Both OSX and Windows provides a program called Secure Shell/Secure Shell Socket (SSH).

SSH allows you to use a terminal emulator running on your computer to connect to remote computers.

Exercises

1. A terminal provides a text-oriented way for a human to interact with a computer (TRUE/FALSE).
2. What would be displayed on an ASCII terminal if the following byte values were sent to it (note the byte values are given in hexademical notation)? `0x48 0x65 0x6c 0x6c 0x6f 0x20 0x57 0x6f 0x72 0x6c 0x64 0x21`.
3. If the following keys were typed on the keyboard of an ASCII terminal (in order from left to right), which byte values would be sent to the computer? Values should be given in hexadecimal notation. **Nice :-).**
4. Translate the phrase '**UNIX DIEHARDS**' to ASCII hexadecimal notation?
5. Which is the bottom layer of a UNIX system?
6. In UNIX, a task or job running on the system can be termed as a?
7. A program built from a process that can be executed in a UNIX system is called?
8. What program can you use to remotely connect to another computer?

Filesystem

A *filesystem* can be defined as a data structure or a collection of files.

UNIX system regards everything as a file. Files can be divided into three categories;

- Ordinary or plain files.
- Directories.
- Special or device files.

In UNIX, filesystem can refer to two very distinct things; *the directory tree* or the *arrangement of files on disk partitions*. The latter can be thought of as the physical filesystem as it has a tangible physical location.

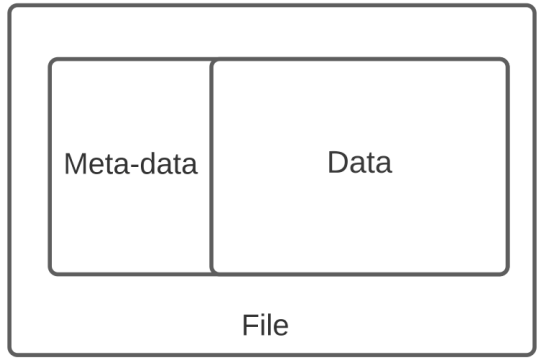
Filesystem: Files

A file is an abstract object that can store information. We can write and read contents to and from it.

Data of a file contain a **collection of bytes**. The bytes of the file might encode ASCII data, image data, audio data, an executable program, etc.

Metadata of a file are associated descriptive facts. Examples of these facts include:

- who owns the file
- the length of the file (measured in bytes)
- who has permissions to read or write its contents
- the time the contents was last modified
- the time that it was last read
- the time that the descriptive facts where last changed (e.g., the file permissions were modified)



A file is an abstract object that can store Data and has associated with it additional descriptive facts we call Meta-data

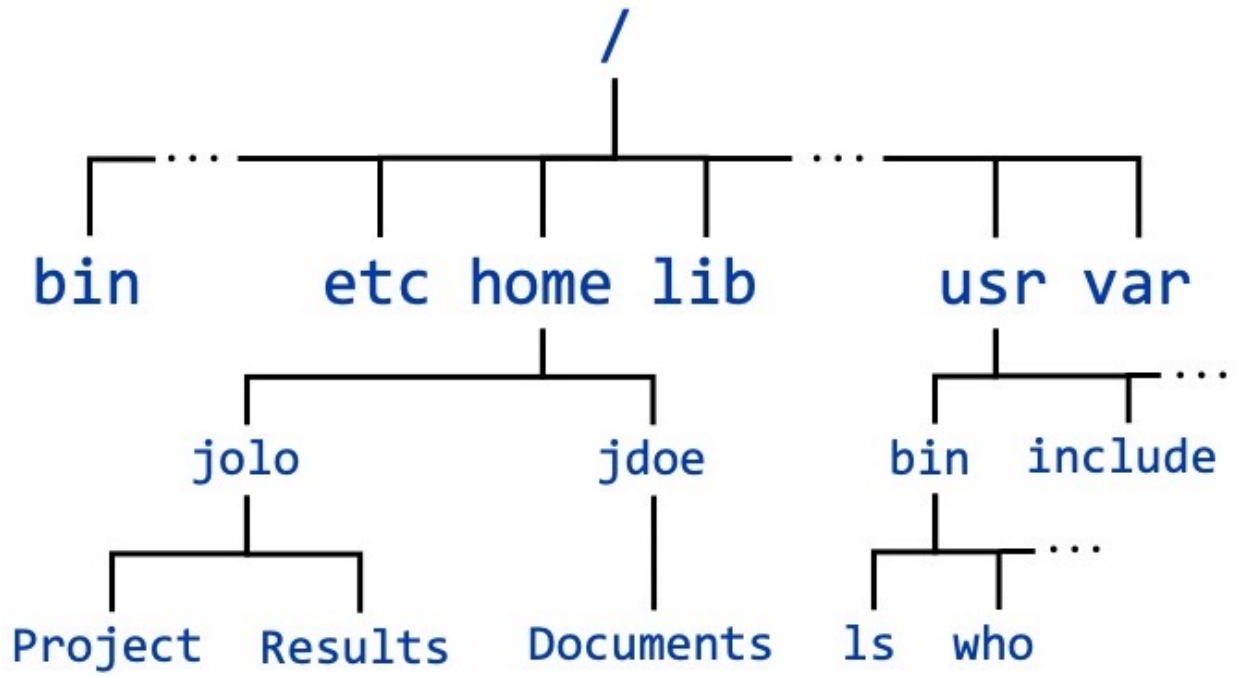
Source: https://jappavoo.github.io/UndertheCovers/textbook/unix/terminal.html

Filesystem: Directory tree

A *directory* contains a list of names each name identifies either a single file or another directory.

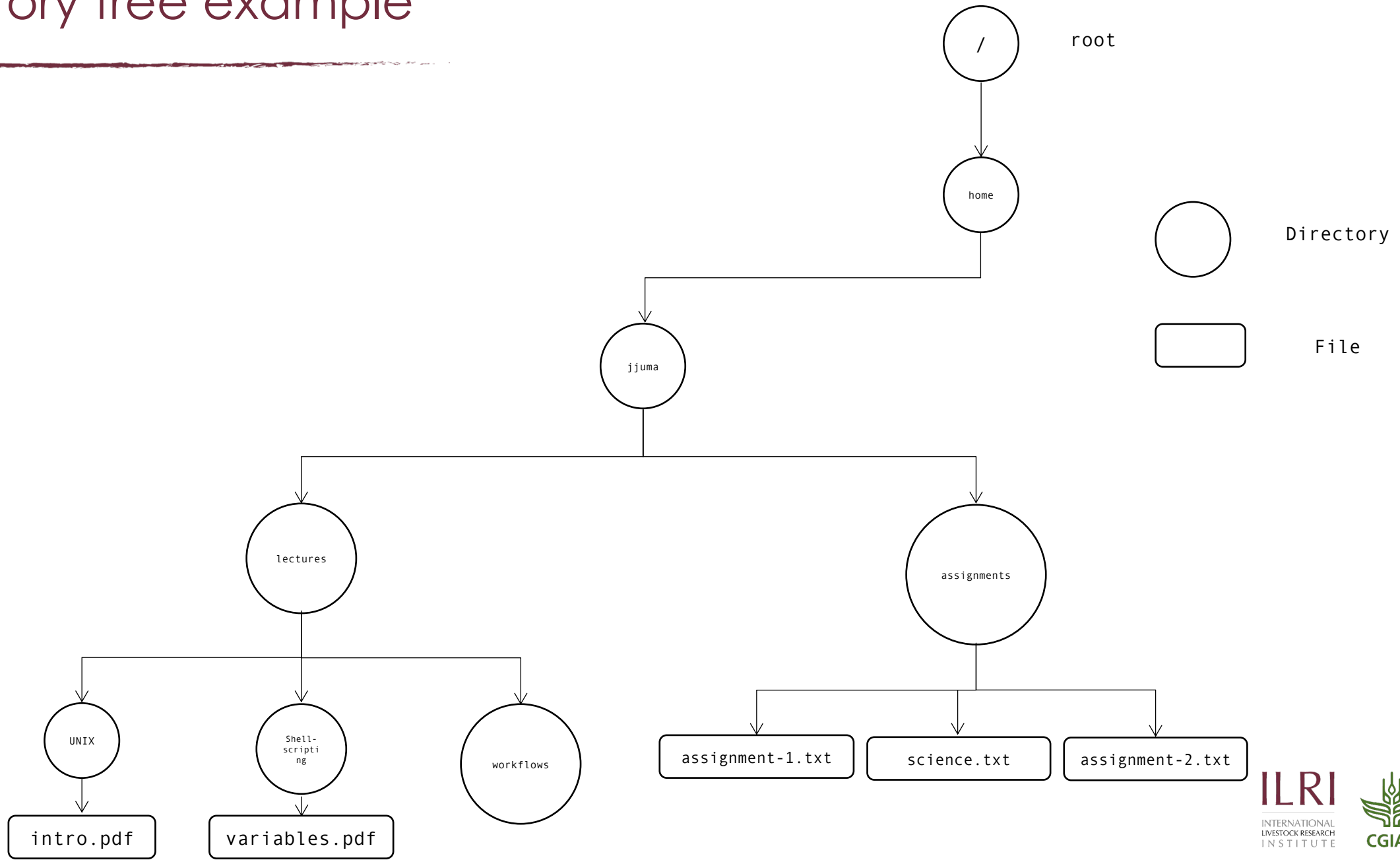
A directory that is within a directory is said to be a sub-directory.

A UNIX user is allowed to create directories and files and name them as they see fit. This ability allows a user to flexibility organize their information in a way they like and makes sense to them.



Source: https://cvw.cac.cornell.edu/linux/filesystem

Directory tree example



Paths

- In UNIX, the top of the tree is the one directory that always exists and is not a sub-directory of any other directory. This is the **ROOT** directory.

- The name of the `variables` file at the bottom of the diagram is composed by joining the name of the directories leading to it along with its name at the end.

```
/ + home + jjuma + lectures + shell-scripting + variables
```

- Notation for a full path name of a file or directory is by joining the independent components with the “ / ” .

```
/home/jjuma/lectures/shell-scripting/variables
```

- `Root path /`

Paths

Absolute path

Details the entire path through the directory structure to get to a file, starting at /, the root directory

```
/home/jjuma/lectures/shell-scripting/variables
```

Relative path

Is the path from where you are now (your present working directory) to the file in question

```
../lectures/shell-scripting/variables
```

Exercises

1. A UNIX file is a physical object that exists on the computer's hard-drive: True or False?
2. What is meta-data versus data with respect to files?
3. The directory structure and name of the directories is fixed: True or False?
4. In the directory tree example in the lecture what is the full path name of `jjuma's workflows` directory?
5. In the directory tree example in the lecture what is the full path name of `jjuma's assignment` files?
6. A directory can contain both files and directories: True or False?

Interacting with the Kernel

A core function of the shell is to be the first running process a human user of a computer can use to interact with the kernel to get things done. The Shell achieves this by:

1) accepting human oriented commands from a user.
2) executing kernel system calls to get the work of the commands done.
3) sending a response back.

The shell includes the ability to locate other executable programs and launch new processes from them.

The shell is text based and is designed with programmers in mind.

Shell variants

- Bourne
- BASH (Bourne Again Shell)
- C shell (csh)
- tcsh
- Korn Shell (ksh)
- Z Shell (zsh)

The command line

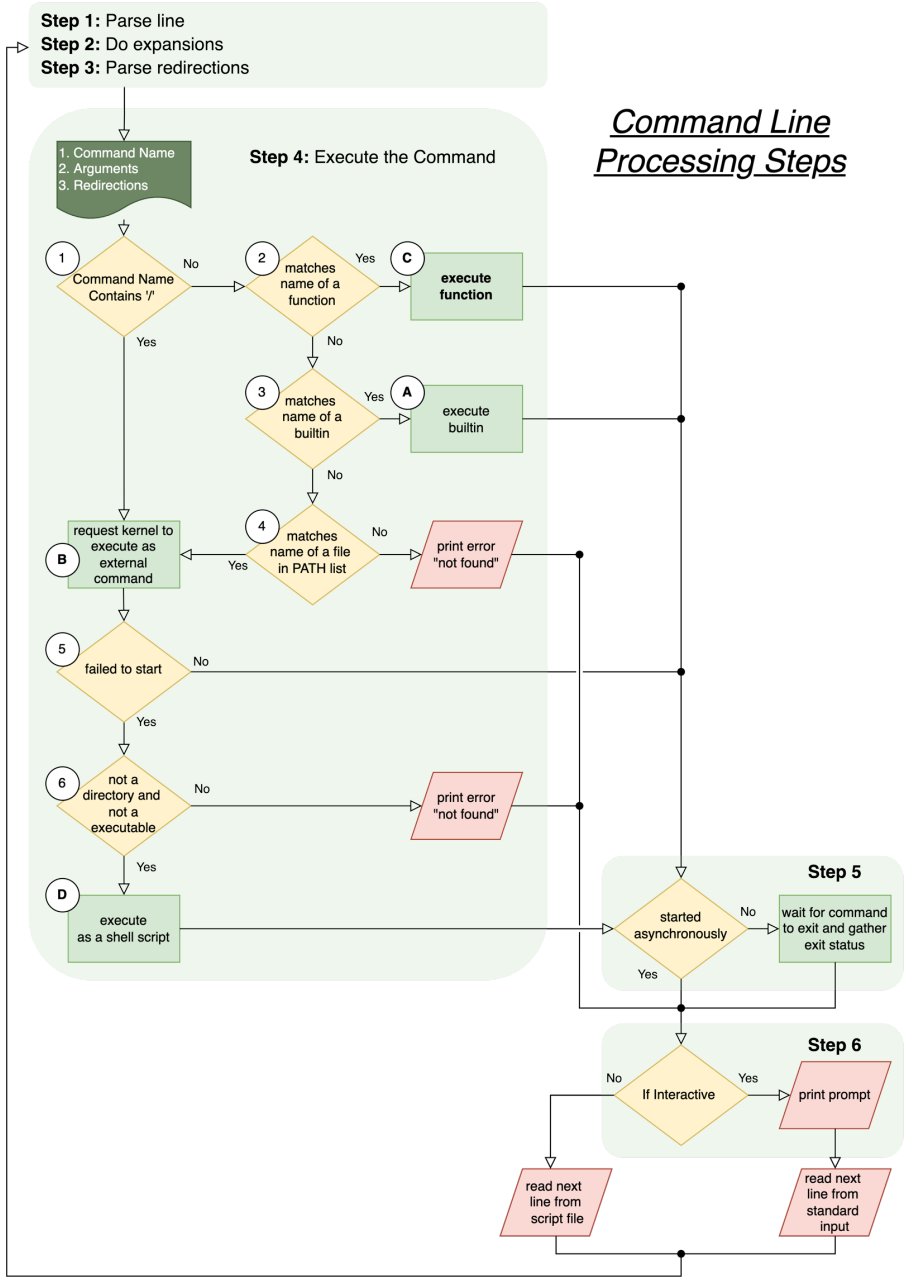
UNIX command lines can get quite complex.

A hallmark of UNIX expertise is the ability to compose long command lines that chain together many commands.

Learning to issue simple command lines is the first step to ruling the UNIX world.

When the shell receives a command line, it goes through a series of steps to process it. The rules of this processing define what is called the Shell Grammar.

Commands can be run interactively vs non-interactively (scripts)



Source: https://jappavoo.github.io/UndertheCovers/textbook/unix/terminal.html

ILRI
INTERNATIONAL LIVESTOCK RESEARCH INSTITUTE
CGIAR

Help commands

man pages (manual pages)

Linux includes a built-in manual for nearly all commands, so these should be your go-to reference.

`man` format and display manual pages

1.NAME – a one-line description of what it does.

2.SYNOPSIS – basic syntax for the command line.

3.DESCRIPTION – describes the program's functionalities.

4.OPTIONS – lists command line options that are available for this program.

5.EXAMPLES – examples of some of the options available

6.SEE ALSO – list of related commands.

Simple commands

Listing files and directories

`ls`	lists the contents of your current working directory.
`ls -a`	lists all the contents of your current working directory (including directories with a dot)

1. What does the command `ls -lth` do?
2. To list all the files in long format and show complete time information, which command can you use?
3. To list all the files in a directory and output in a comma-separated format, which optional flag can you use?

Simple commands

Making directories

`mkdir` make a directory.

1. Make a directory in your **`home`** directory and name it **`lectures`**
2. In **`lectures`** directory make additional sub-directories, **`unix-stuff`** and **`shell-scripting`**
3. Make another directory in your **`home`** directory and name it **`assignments`**

Simple commands

Changing directory

cd change directory.

1. Change directory to **unix-stuff** and list all the contents of the directory

Current directory

cd .

Parent directory

cd ..

Typing **cd** with no argument always returns you to your **home** directory.

Simple commands

Copying files

`cp` copy files

Typical command

`cp file1.txt file2.txt` copy file1.txt in the current working directory and name it file2.txt

1. Copy the **`science.txt`** file to the **`assignments`** directory.
1. Assuming you have a file named **`file1.txt`** in the current working directory, what does the command **`cp file1.txt .`** do?
2. Copy the **`lectures`** directory into **`assignments`** directory
3. Copy only the contents of **`assignments`** directory into **`lectures`** directory

Simple commands

Moving files

`mv`	move or rename files

Typical command

`mv source target`	rename
`mv source directory`	move

Exercise

In your **`home`** directory, make a directory named **`backups`** and move the **`science.txt`** file from **`assignments`** to the **`backups`** directory.

Simple commands

Clearing the terminal screen

`clear`	clear the terminal screen

Display content of a file

`cat`	concatenate and print files
`less`	display contents of a text file on terminal screen one page at a time
`more`	display contents of a text file on terminal screen one page at a time
`head`	display first lines of a file
`tail`	display last lines of a file

Exercise

Display all the contents of the **`science.txt`** file
Display the last 5 lines in the **`science.txt`** file
Display the first 5 lines in the **`science.txt`** file

Simple commands

Searching the contents of a file

`less` display contents of a text file on terminal screen one page at a time

To search for a word in a text file, we can use the **`less`** command followed by the forward slash **`(/)`** and type the word to search. This will highlight all the occurrences of the word. Type [**n**] to search for the next occurrence of the word.

Exercise

Search for all the occurrences of the word **science** in the **`science.txt`** file

Simple commands

Searching the contents of a file with `grep` (global regular expression print)

`grep` searching and matching patterns in a file

`grep` is case sensitive. You can ignore this feature by using the flag `-i`, i.e `grep -i`

Exercise

```
grep science science.txt
```

1. Search for all occurrences of the word **science**, ignore case sensitivity.
2. Display lines that do not match the word **science.**
3. Print only the total count of the matched lines.
4. Display the line that match preceded by the line number.

Simple commands

Word and line counts

wc word, line, character and byte count

Exercise

```
wc science.txt
```

1. How many lines are in the `science.txt` file.
2. How many characters are in the `science.txt` file.
3. What is the size of the `science.txt` file in bytes?

Simple commands

I/O and Redirection

Most processes in UNIX:

- standard input (**`stdin`**): stream data going into a program. By default, this is input from the keyboard.
- standard output (**`stdout`**): output stream where data is written out by a program. By default, this output is sent to the screen.
- standard error (**`stderr`**), another output stream (independent of stdout) where programs output error messages. By default, error output is sent to the screen.

The output of a process or program can be saved using the redirection operator >.

For example, you can redirect the output of the command **`ls`** to a file by:

```
ls -lth > list_of_files.txt
```

This will create a file called **`list_of_files.txt or overwrite`** it if it does not exist.

If you know that the file exists, you can **append** more content by using the operator **>>**

```
ls -lth >> list_of_files.txt
```

Simple commands

I/O and Redirection

Redirect the standard input into a file using `>` operator.

```
cat > list1
```

Type the following in an interactive manner

```
pear
banana
Apple
```

Exit the terminal by hitting `Ctrl ^ [d]`

Simple commands

I/O and Redirection

Redirect the standard input into a file using `>` operator.

```
cat > list1
```

Type the following in an interactive manner

```
pear
banana
apple
```

Hit `Ctrl ^ [d]` to stop the input stream

Using the above method, create another file called `list2` containing the following fruits: `orange, plum, mango, grapefruit`. Read the contents of `list2`

Concatenate the two lists, `list1` and `list2` and name the resulting list as `biglist`.

Simple commands

Sort items

Type **`sort`** and enter the following

dog
cat
bird
ape

Hit **`Ctrl ^ [d]`** to stop the input stream

Simple commands

Input Redirection

Input can also be given to a command from a file instead of typing it in the shell by using the redirection operator <.

```
cat < science.txt
```

Error Redirection

```
rmdir no-dir 2> nodir_error.txt
```

Merge stdout with stderr

```
cat science > combined_output.txt 2>&1
```

Use **`biglist`** as input stream, **`sort`** the list and redirect to a standard output list called **`slist`**

Simple commands

Pipes

who Display who is logged in

One method to get a sorted list of names is to type,

```
who > names.txt
sort < names.txt
```

This is a bit slow, and you must remember to remove the temporary file called **`names.txt`** when you have finished. What you really want to do is connect the output of the **who** command directly to the input of the **sort** command. This is exactly what pipes do. The symbol for a pipe is the vertical bar |

who | sort

Print the count of sorted list of all users logged in

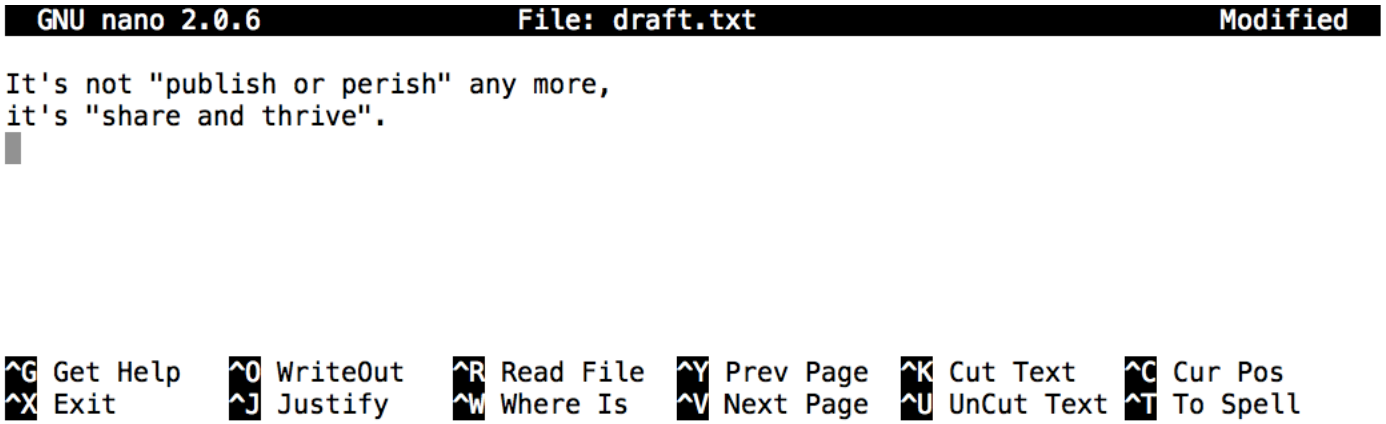
Using pipes, display all lines of **`list1`** and **`list2`** containing the letter '**p**', and **`sort`** the result.

Simple commands

Creating files

Files containing texts can be created using **text** editors such as **`nano, vi, vim`** on command line

Create a text file named **`draft.txt`** in a directory called **`thesis`** using **`nano`** and type the texts as shown on the screenshot below.



Once we're happy with our text, we can press **Ctrl+O** (press the **Ctrl** or **Control** key and, while holding it down, press the **O** key) to write our data to disk (we'll be asked what file we want to save this to: press **Return** to accept the suggested default of **`draft.txt`**).

https://swcarpentry.github.io/shell-novice/03-create/index.html

ILRI INTERNATIONAL LIVESTOCK RESEARCH INSTITUTE CGIAR

Other commands

Redirect and save output

To have the output go to both a file and the screen simultaneously.

```
cat science.txt | tee out.stdout.txt
```

You can also use **`tee`** to catch **`stderr`** with:

```
rmdir no-dir | tee out.stderr.txt
```

Other commands

Wildcards

* is a **wildcard**, which matches zero or more characters. Let's consider the **`shell-lesson-data/exercise-data/proteins`** directory: **`*.pdb`** matches **`ethane.pdb, propane.pdb,`** and every file that ends with **`.pdb`**.

On the other hand, **`p*.pdb`** only matches **`pentane.pdb`** and **`propane.pdb`**, because the '**p**' at the front only matches filenames that begin with the letter '**p**'.

File naming conventions

A directory is merely a special type of file. So, the rules and conventions for naming files apply also to directories. In naming files, characters with special meanings such as **/ * & %** , should be avoided. Also, avoid using spaces within names. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with _ (underscore) and . (dot).

Examples

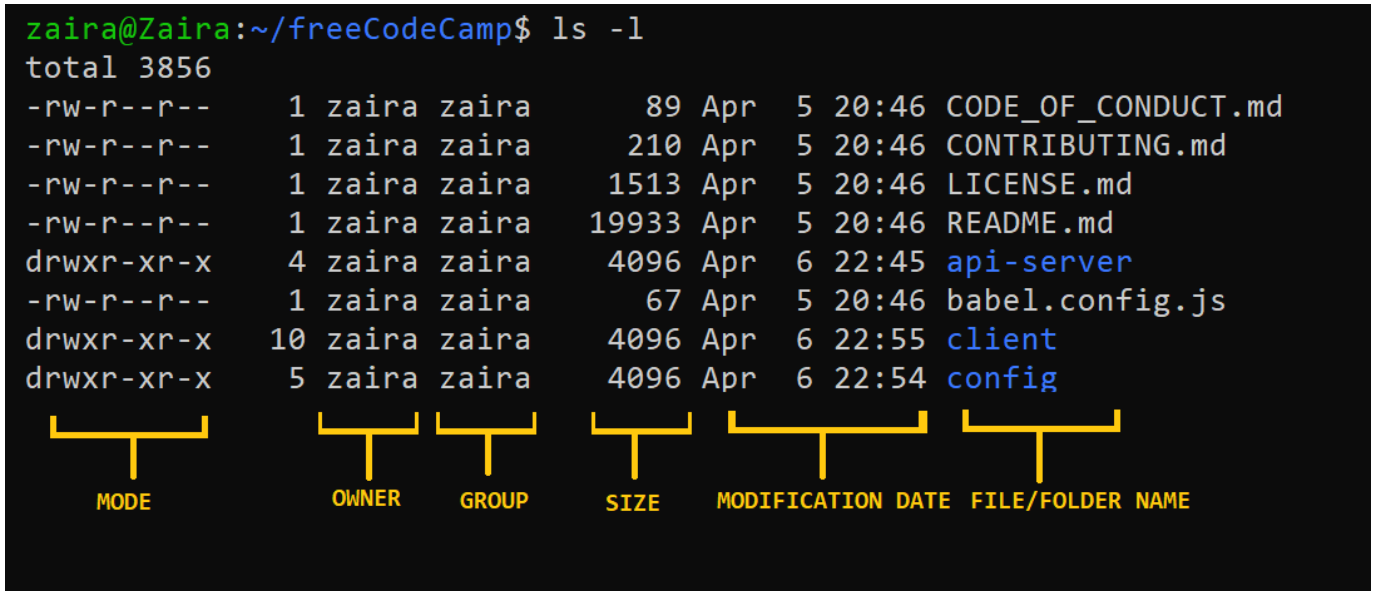
Good filename	**Bad filename**
assignment.txt	assignment
merged_files.txt	merged files.txt
dengue_and_chikungunya.fasta	dengue & chikungunya.fasta
metadata.csv	meta data.csv
project-dir	project dir

File permissions

In UNIX, there are three types of owners: **`user`**, **`group`**, and **`others`**.

File permissions fall in three categories: **`read`**, **`write`**, and **`execute`**.

	Read	**Write**	**Execute**
Files	Open, read	Edit, Delete, Save (Modify)	Run an executable file (script)
Directories	Listing without modification	Create, Delete, Rename contents	Access



`user (u)`	`group (g)`	`others (o)`
`rwx`	`rwx`	`rwx`

https://www.freecodecamp.org/news/linux-chmod-chown-change-file-permissions/

ILRI INTERNATIONAL LIVESTOCK RESEARCH INSTITUTE CGIAR

File permissions – changing permissions

chmod change file modes

Syntax

```
chmod permissions filename
```

Where,

- **permissions** can be read, write, execute or a combination of them.
- **filename** is the name of the file for which the permissions need to change. This parameter can also be a list if files to change permissions in bulk.

We can change permissions using two modes:

- **Symbolic mode**: this method uses symbols like `u, g, o` to represent users, groups, and others. Permissions are represented as `r, w, x` for read write and execute, respectively. You can modify permissions using **+, -** and **=**.

- **Absolute mode**: this method represents permissions as 3-digit octal numbers ranging from **0-7**.

File permissions – changing permissions

Suppose I have a script and want to make it executable for the owner, **jjuma**.

```
-rw-r--r--  1 jjuma  1995788633   948B Feb  4 22:07 fastqc-run.sh
```

I can change the file permission using the command

```
chmod u+x fastqc-run.sh
```

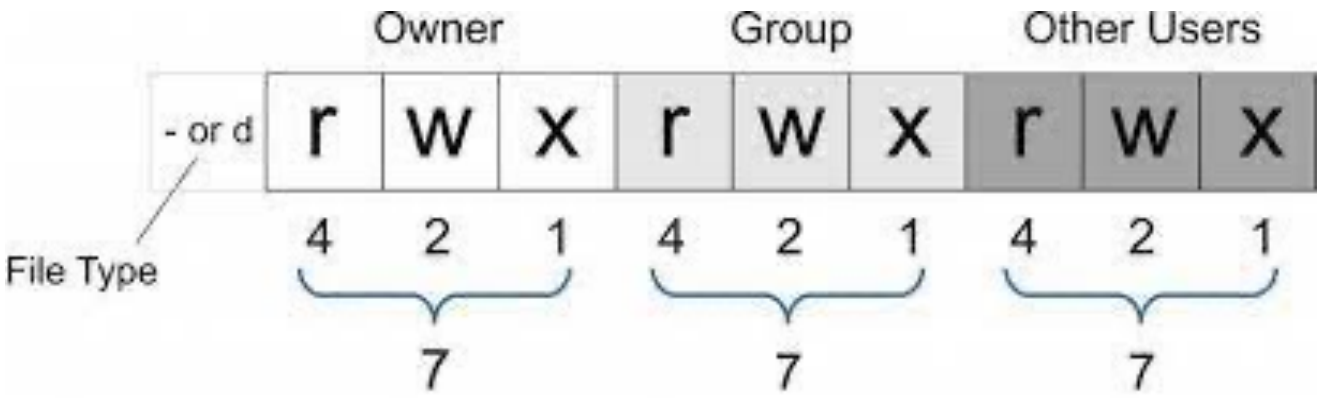
```
-rwxr--r--  1 jjuma  1995788633   948B Feb  4 22:07 fastqc-run.sh
```

What would the commands below do to the script?

```
chmod go+x fastqc-run.sh
chmod o-r fastqc-run.sh
chmod g=w fastqc-run.sh
```

File permissions – changing permissions using absolute mode

Absolute mode uses numbers to represent permissions and mathematical operators to modify them.



Permission	Assign permission	Revove permission
Read	Add 4	Subtract 4
Write	Add 2	Subtract 2
Execute	Add 1	Subtract 1

File permissions – changing permissions using absolute mode

Set read for user, read and execute for group, and only execute for others.

	Read	Write	Execute	Sum
user	4	0	0	4
group	4	0	1	5
others	0	0	1	1

```
chmod 451 filename
```

Remove execution rights from others and group.

	Read	Write	Execute	Sum
user	4	0	0	4
group	4	0	1 - 1	4
others	0	0	1 - 1	0

```
chmod 440 filename
```

File permissions – changing permissions using absolute mode

If the resulting mode of **`chmod 451`** is equivalent to `r--r-x–x`?

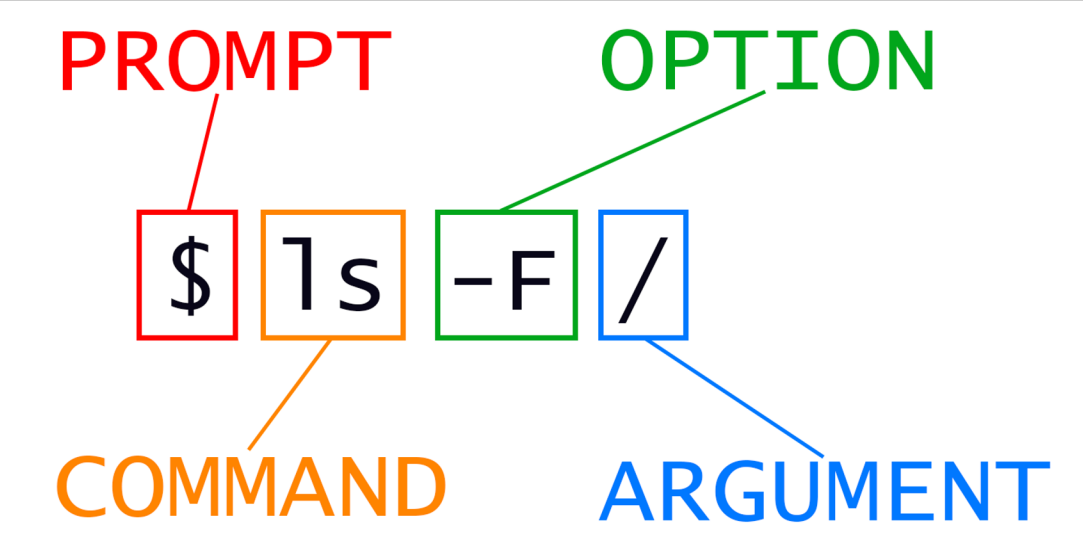
Write the resulting file modes for the following

```
chmod 777 filename
chmod 754 filename
chmod 700 filename
```

Other useful commands and syntax of shell command

```
wget
history
zcat
find
gzip
df
file
diff
```

Syntax of shell command



Setup

Download the data from https://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip

Other useful commands and syntax of shell command

`cut` cut out selected portions of each line of a file

In the **`animal-counts`** sub-directory within **`exercise-data`** directory, run the command

```
cut -d , -f 2 animals.csv
```

We can further obtain uniq animals by using the command

```
cut -d , -f 2 animals.csv | sort | uniq
```

Challenge

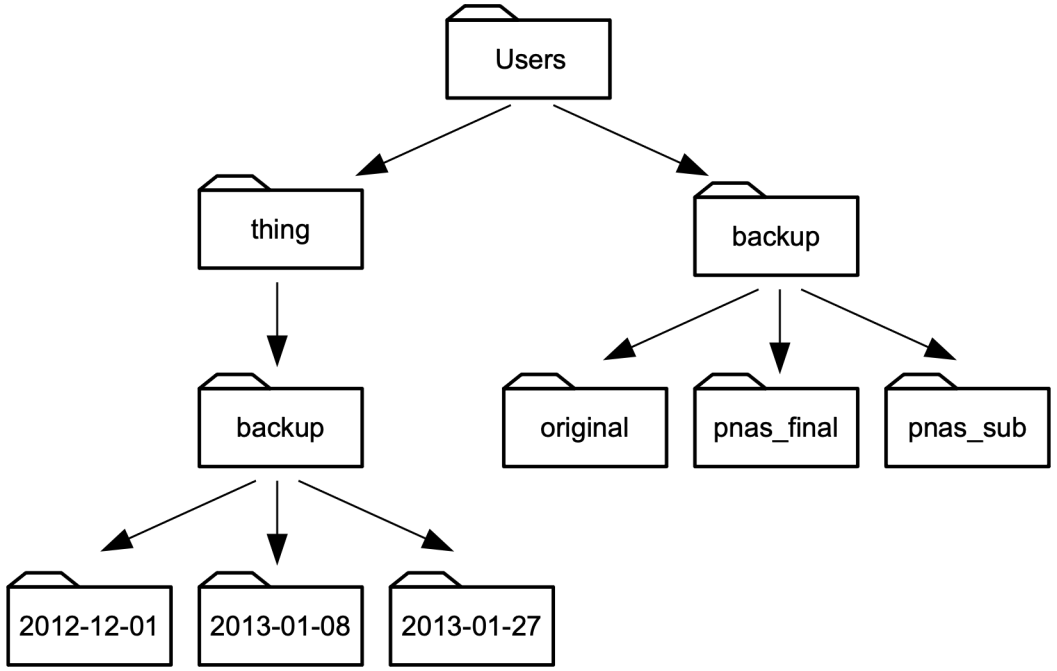
Starting from **`/Users/amanda/data`**, which of the following commands could Amanda use to navigate to her home directory, which is **`/Users/amanda`**?

```
cd .
cd /
cd /home/amanda
cd ../..
cd ~
cd home
cd ~/data/..
cd
cd ..
```

Challenge

Using the filesystem diagram below, if **pwd** displays **/Users/thing**, what will **ls -F ../backup** display?

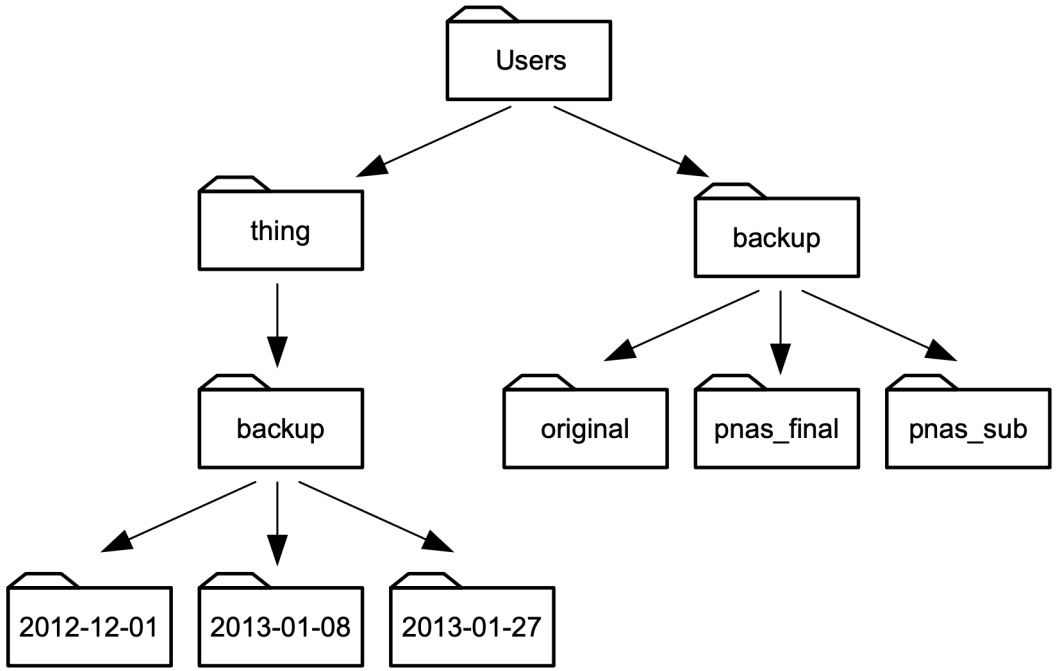
```
../backup: No such file or directory
2012-12-01 2013-01-08 2013-01-27
2012-12-01/ 2013-01-08/ 2013-01-27/
original/ pnas_final/ pnas_sub/
```



Challenge

Using the filesystem diagram below, if **pwd** displays `/Users/backup`, and `-r` tells ls to display things in reverse order, what command(s) will result in the following output:

```
pnas_sub/ pnas_final/ original/
```



Challenge

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 filename! You want to correct the mistake, which of the following commands could you use to do so?

```
cp statstics.txt statistics.txt
mv statstics.txt statistics.txt
mv statstics.txt .
cp statstics.txt .
```

Challenge

Count the number of lines in all the files ending with **`.pdb`** extension in the **`proteins`** directory, sort by the number of lines and and redirect the output to a file called **`sorted_lengths.txt`** in a new directory called **`results`**.

In the **`proteins`** directory, we want to find the 3 files which have the least number of lines. Which command listed below would work?

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

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 **`.txt`**. Which of the following would remove all the processed data files, and *only* the processed data files?

```
rm ?.txt
rm *.txt
rm * .txt
rm *.*
```

Challenge

The **`uniq`** command has a **`-c`** option which gives a count of the number of times a line occurs in its input. Assuming your current directory is **`shell-lesson-data/exercise-data/animal-counts`**, what command would you use to produce a table that shows the total count of each type of animal in the file?

```
sort animals.csv | uniq -c
sort -t, -k2,2 animals.csv | uniq -c
cut -d, -f 2 animals.csv | uniq -c
cut -d, -f 2 animals.csv | sort | uniq -c
cut -d, -f 2 animals.csv | sort | uniq -c | wc -l
```