Final Notes

Lab 1 Exploring Linux Distributions

Requirements:

- **You will use Github to submit this Lab. Watch the 'What is Git and Github?' video before your start**
- Create a Github account before your start this lab.
- Bold your answers *(2 points)*
- Format your screenshots accordingly (not too big, not too small). If I can't read it, I cannot grade it.
- Video: https://youtu.be/8Sa52S527Qc

Working with Distrowatch

Question 1

During class, we explored the concept of Linux distribution. During this lab, you will research some Linux distributions using a website called Distrowatch. DistroWatch is a website that provides news, popularity rankings, and other general information about various Linux distributions and other Unix-like operating systems such as OpenSolaris, MINIX and BSD.

Go to Distrowatch. Explore the website to get familiar with the home page. On the top left corner, you have a form that allows you to submit queries to the website. In the **"Type Distribution Name"** box type **"Ubuntu".** This will return details about Ubuntu. Explore the Ubuntu Distrowatch page and answer the following questions:

1. What is the OS Type:
    - **Linux**
2. Which major distro is it based on?
    - **Debian**
3. Which processor architecture does it support?
    - **armhf, ppc64el, riscv, s390x, x86_64**
4. Is the distribution active or is it discontinued?
    - **Active**
5. What is the distro's home page?
    - **https://www.ubuntu.com/**

Question 2

On the top left corner, click on "Random Distribution" and answer the following questions from the distro you got.

1. What is the name of the distribution and the OS Type:
    - **Linux**

2. Which major distro is it based on?
    - **Debian, Ubuntu**
3. Which processor architecture does it support?
    - **x86_64**
4. Is the distribution active or is it discontinued?
    - **Active**
5. What is the distro's home page?
    - **http://luninuxos.com/**

Question 3

On the top of the page, right in the middle, you will find an option that allows you to search for distributions. Click on **"Search"** and after the page loads, fill in the following information in the **"Search Distribution by Criteria"** section and Click on Submit Query.

- OS Type: Linux
- Architecture: x86_64
- Status: Active
- Leave the rest as default.

From the query results, choose any distribution and answer the following question about the distro you chose.

1. What is the name of the distribution?
    - **Linux**
2. What is the country of Origin?
    - **France, Taiwan**
3. What major distribution is it based on?
    - **Debian, Ubuntu**
4. What is the distribution category?
    - **Live Medium, Netbooks**
5. Which processor architecture, aside from the one in the original query, does the OS support?
    - **i686, x86_64**

Question 4

Now that you know how to use Distrowatch. Find a Linux distribution for the following scenarios. For each distribution provide the website, name, and supported architecture.

1. A Linux distribution used for Data Rescue/Data recovery

- Distro Name: **Kali Linux (formerly BackTrack)**
- Website: **http://www.kali.org/**
- Desktop Environment: **Enlightenment, GNOME, KDE Plasma, LXDE, MATE, Xfce**

2. A Linux distribution used for Education that supports the ix86 processor architecture.

- Distro Name: **Puppy Linux**
- Website: **http://www.puppylinux.com/**
- Desktop Environment: **JWM, Openbox**

3. A Linux distribution that supports the OEM installation method

- Distro Name: **Kubuntu**
- Website: **http://www.kubuntu.org/**
- Desktop Environment: **KDE Plasma**

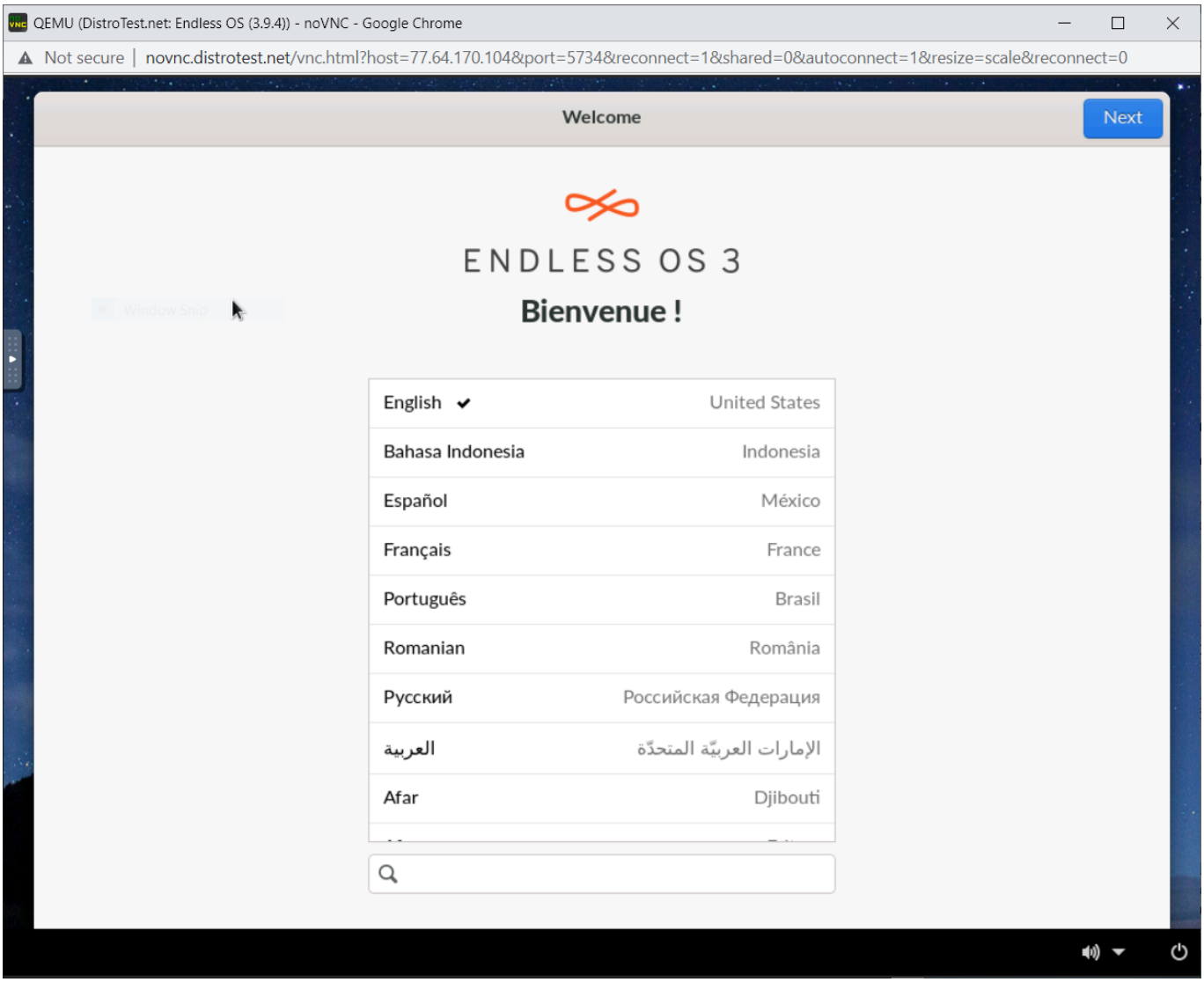
Question 5 (Extra credit 2 pts - Optional)

On the Distrowatch homepage in the menu located in the middle of the page, you will find an option called **“Submit Distribution”.** This option lists all the Linux distros that are pending evaluation, on development or that are experiencing some sort of legal constraint. Select one of these distributions and in a paragraph, share your thoughts. (keep it simple 5 to 8 sentences). **Type your paragraph here**

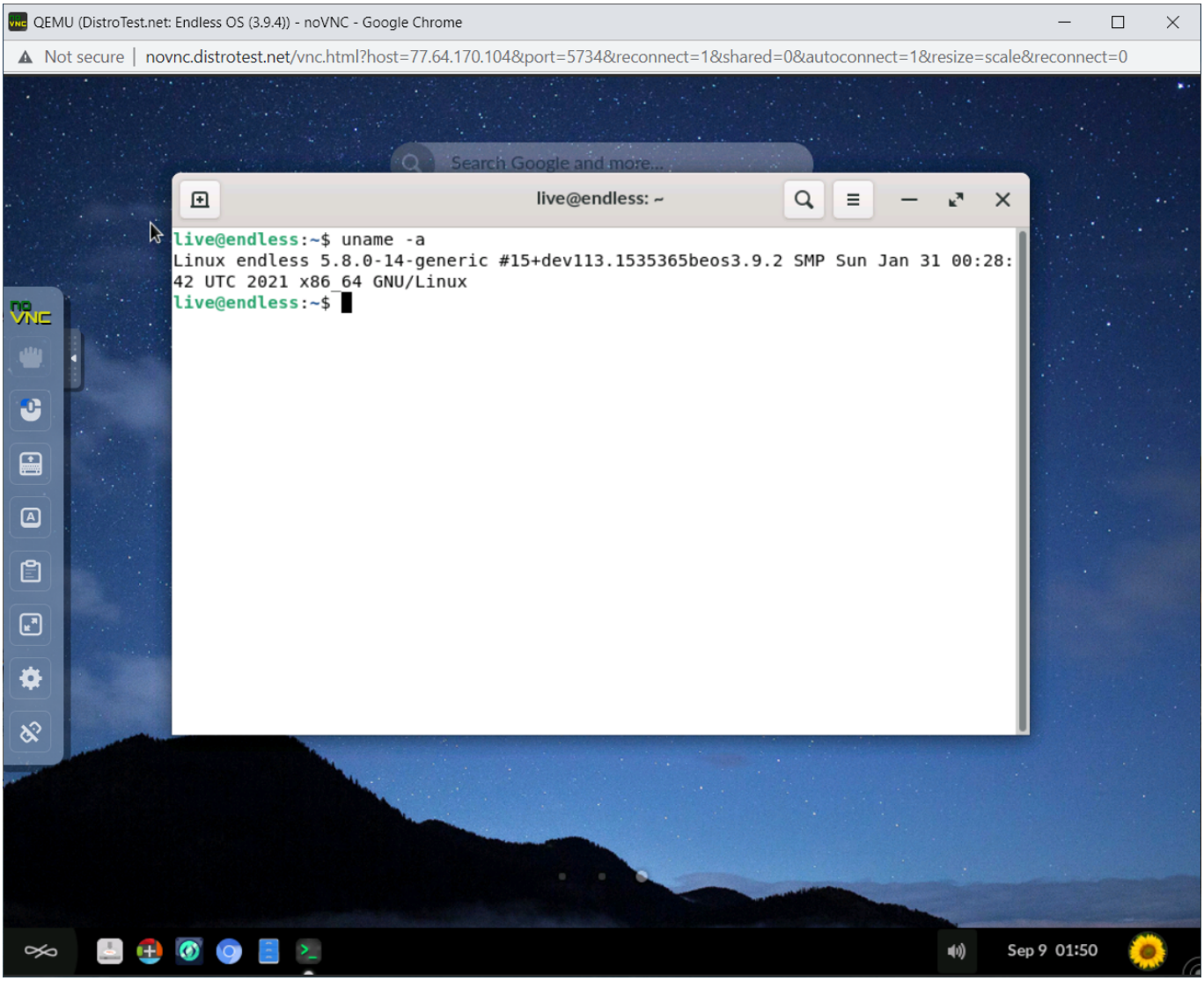
Working with DistroTest.net

Question 6

DistroTest.net is a project that allows you to test Linux/BSD distributions on your web browser. This website is great for trying out distributions before you even download the ISO file. Go to Distrotest.net and click on any of the distributions. Start the distribution and take a screenshot of the browser window that just popped up.



Locate the terminal application in the distribution you started and type the following command: `uname -a`
Take a screenshot of the browser window showing the terminal application open.



Stop the machine and take a screenshot of the browser window showing that the machine has been stopped.

The basic of Virtualization

Definition

Replication of hardware o simulate a virtual machine inside a physical machine.

Two type of virtualization:

- Server-side virtualization
- Client-side virtualization

Server-side

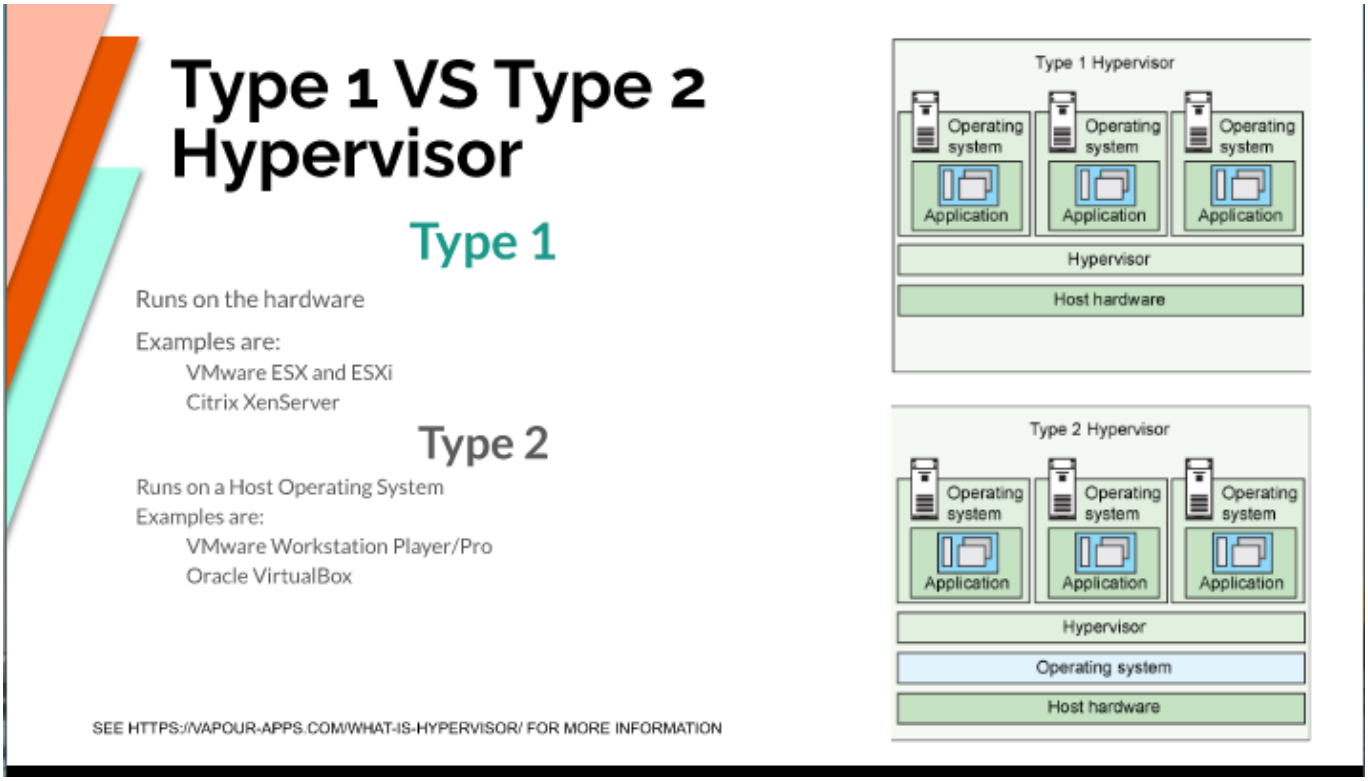
Virtual Desktop Infrastructure (VDI).

- Thick client or fat client
- Thin client
- Zero client

Client-side

- Software installed on a computer to manage virtual machines
- Each VM has its own OS installed
- The computer needs:
    - A hypervisor
    - Hardware support
        - Capable CPU
        - Enough RAM
        - Enough STORAGE

Type 1 Vs Type 2 Hypervisor



Benefits of Virtualization

- Allows running multiple OSs on one machine.
- Reduces costs by decreasing the physical hardware that must be purchased for a network.

Example of VMs

- VirtualBox
    - It is **Type 2** virtualization product
    - Open Source software
    - supports a **Large** number of guest OSs
- VMWare Workstation Player
    - Type 2
    - Available for **Linux** and **Windows**
    - supports a **Large** number of guest OSs

Can my computer virtualize?

Your computer should meet the following minimal specifications:

- AMD V or INTEL V compatible processor
- DUAL core x64 processor with 1.3 GHz or faster 4GB of RAM
- Enough free HARD DRIVE space for installing guest OSs.

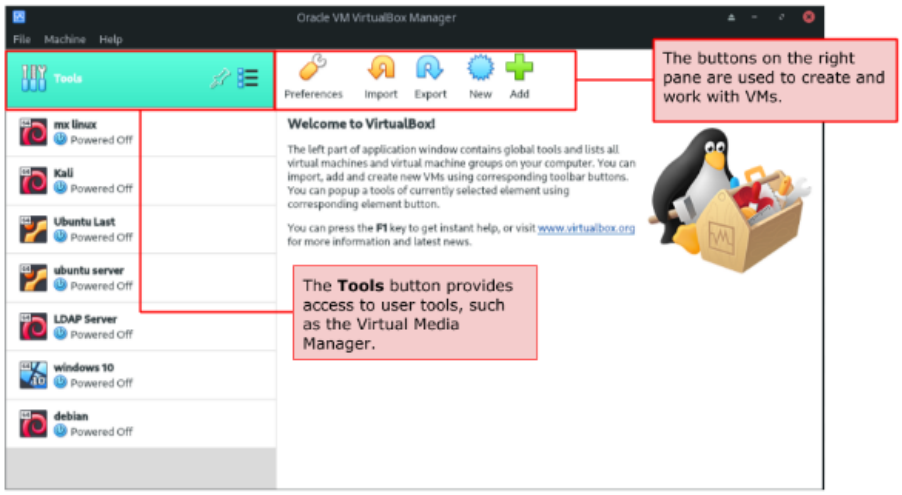
Using VirtualBox

VirtualBOx Extension Pack

- Base Package
- Extension pack
    1. The virtual USB 2.0 (EHCI) device.
    2. The virtual USB 3.0 (xHCI) device.
    3. VirtualBox Remote Desktop Protocol (VRDP) support.
    4. Host webcam passthrough.
    5. Intel PXE boot ROM.
    6. Disk image encryption with AES algorithm.
- To install VirtualBox Extension Pack just double click on the file.

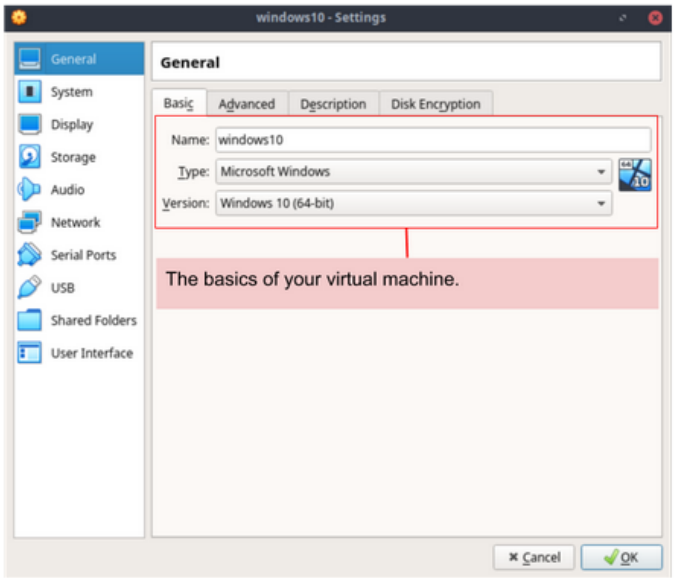
Exploring VirtualBox

Exploring VirtualBox



This window is called the **VirtualBox Manager**.

Exploring VirtualBox | VirtualBox Settings



Creation a Virtual Machine with VirtualBox

> Click on this link: https://www.youtube.com/watch?v=tjHplNis2kE

Installing Ubuntu 20.04 in VirtualBox

> Click on this link: https://www.youtube.com/watch?v=2MEN_IX8gJ8

Raspberry PI

Download the free ebook: https://magpi.raspberrypi.org/books/beginners-guide-4th-ed

What's a Raspberry PI?

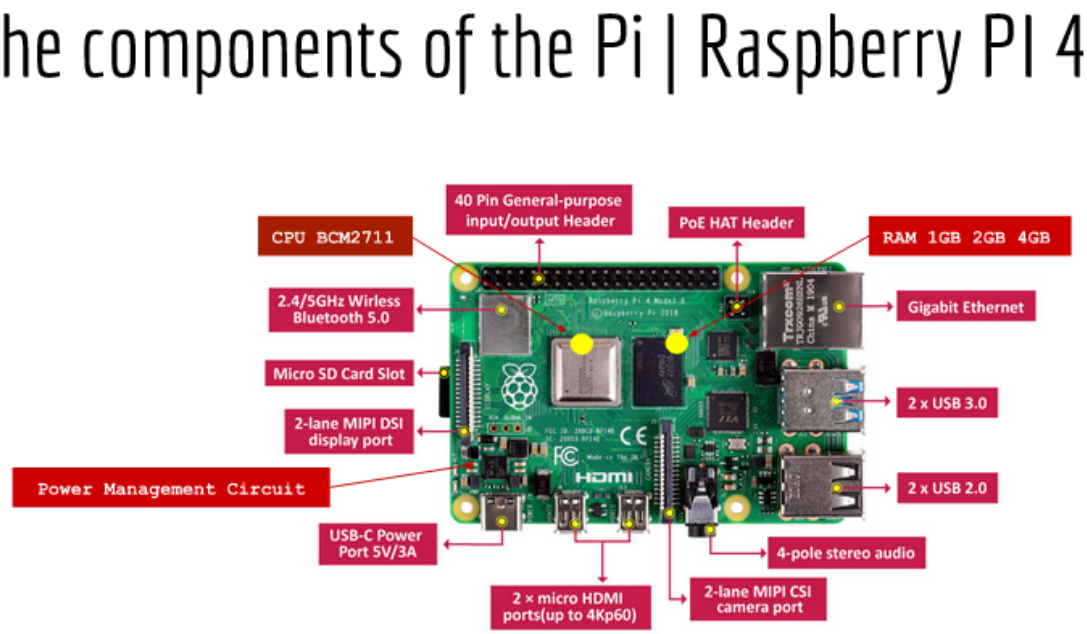
- The Raspberry pi is a low cost
- Credit-card sized computer that plugs into a computer monitor or TV
- It's capable of doing everything you'd expect a desktop computer to do.

Raspberry pi Foundation

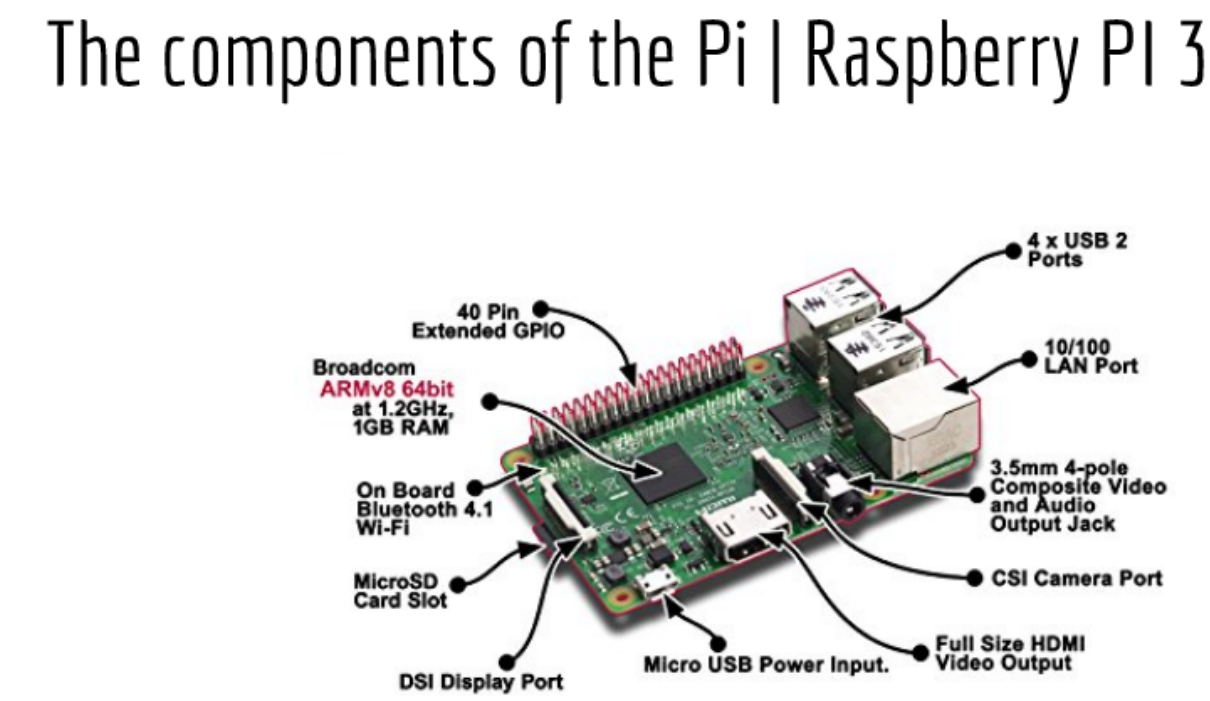
- The Raspberry pi Foundation is registered educational charity based in the UK
- Goal is to advance the education of adults and children, particularly in the field of computer science and related subjects.

PI COMPONENTS

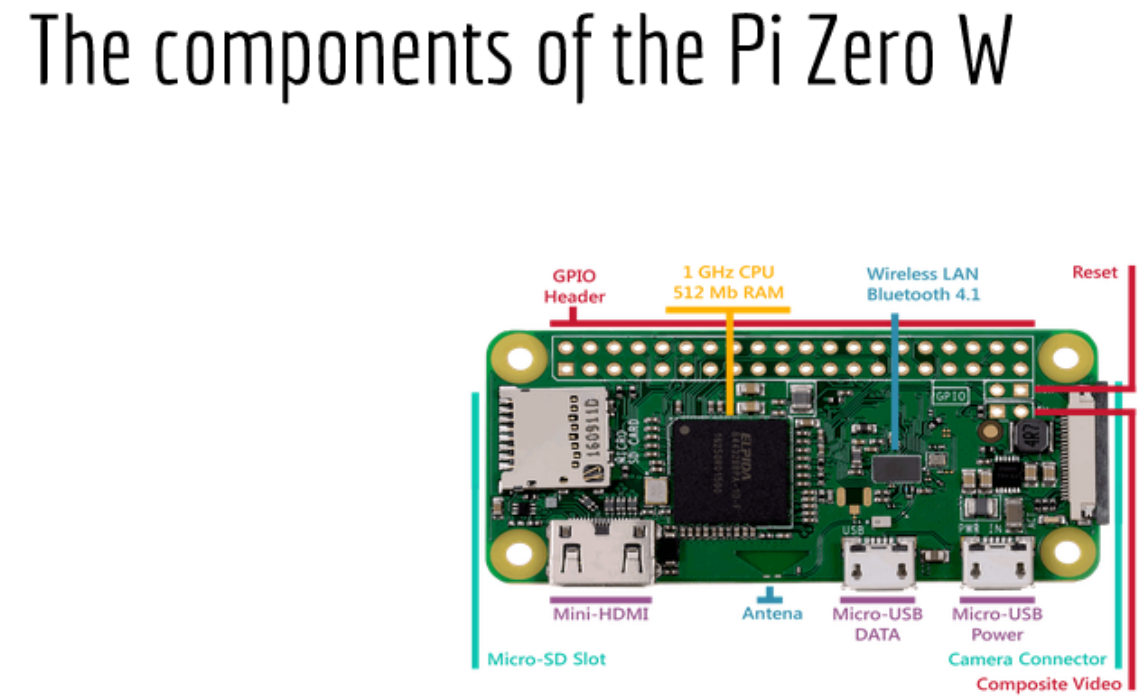
The components of the Pi | Raspberry PI 4



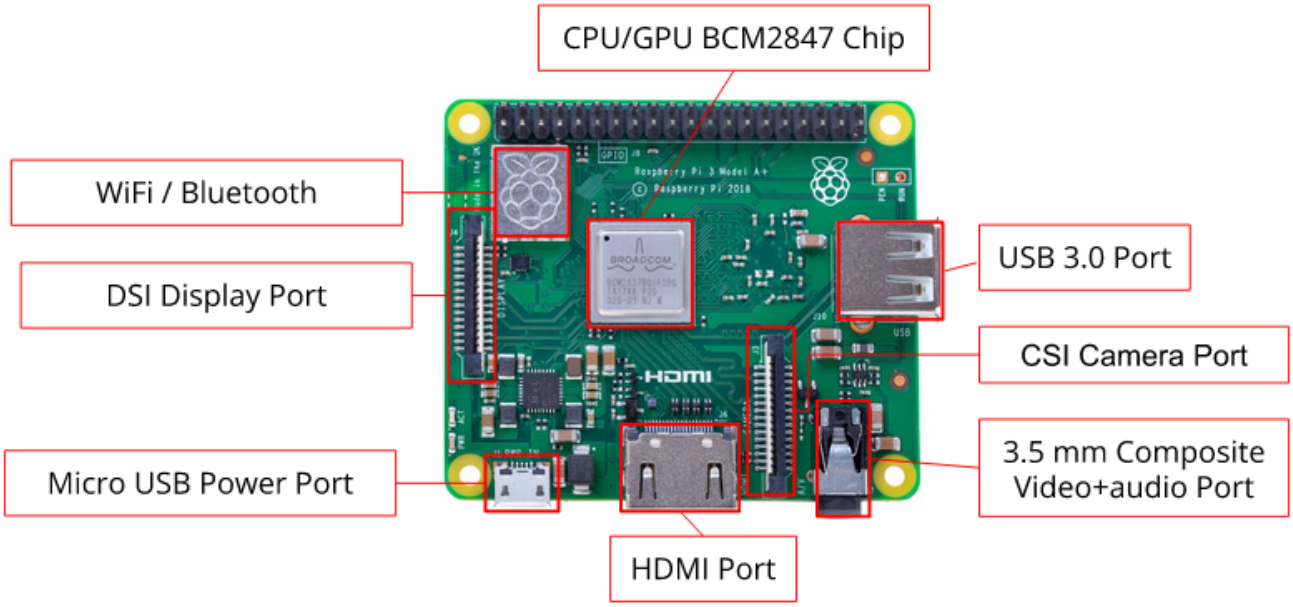
The components of the Pi | Raspberry PI 3



The components of the Pi Zero W



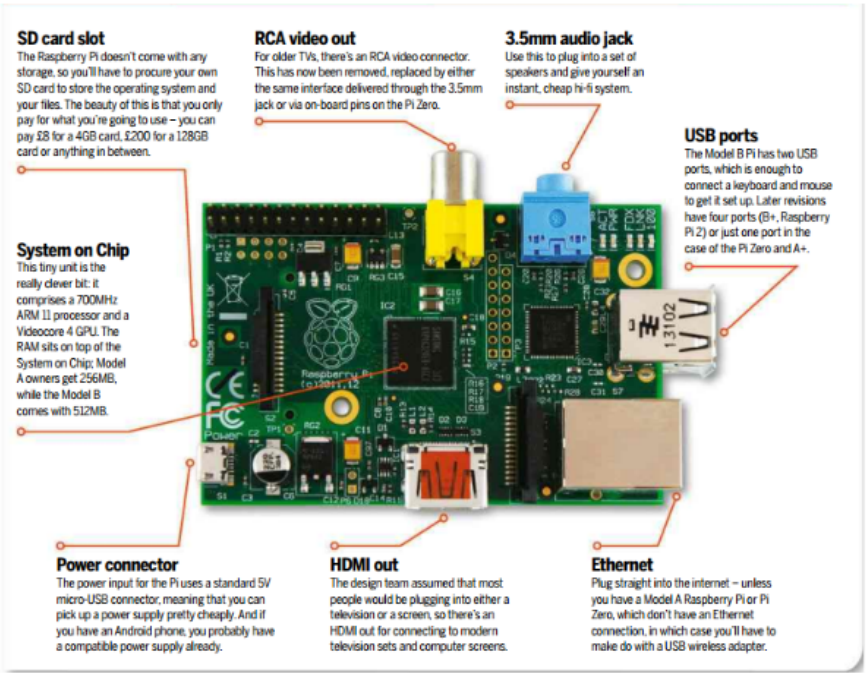
The components of the Pi 3 A+



The Raspberry Pi 400

A look at the first model of the Raspberry Pi

A look at the first model of the Raspberry Pi



Rasberry Pi SD-Card compatibility

> https://elinux.org/RPi_SD_cards

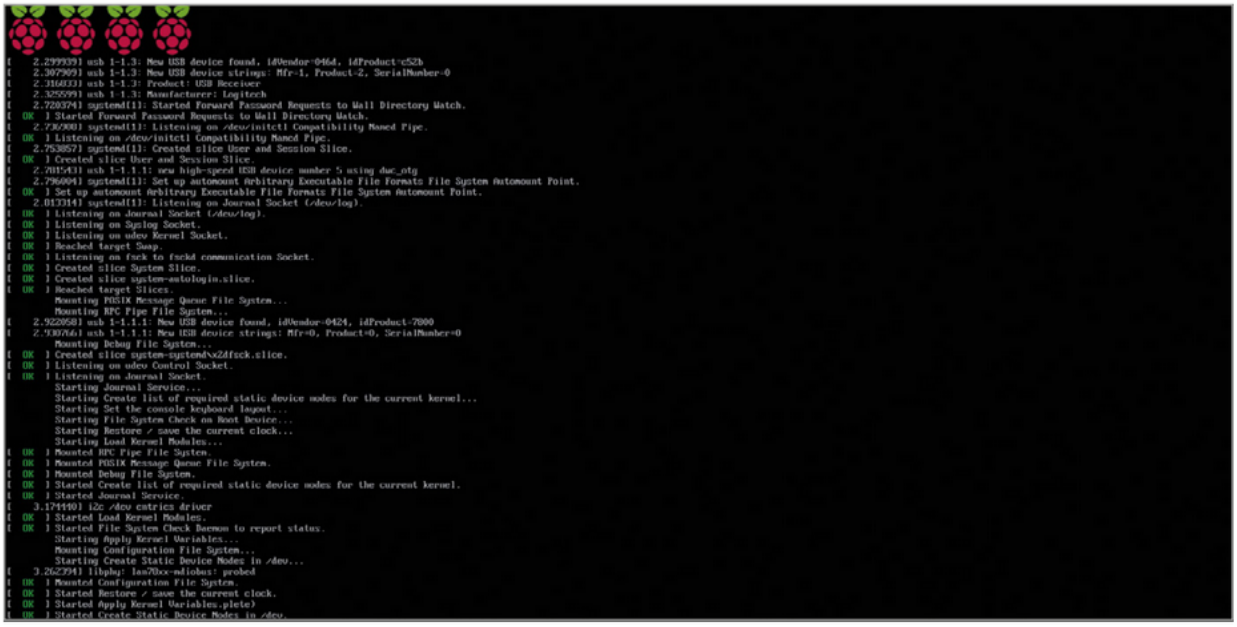
What Hardware Do I need?

What Software do I need:

- Download **Raspberry Pi Imager**.

Working with Raspberry Pi Os

Raspberry Pi OS boot screen



Different Operating Systems for the Raspberry Pi

- Ubuntu
- Kali Linux
- Diet pi
- Arch Linux
- Elementary OS
- Manjaro
- Windows 10

- Android

Where to buy?

- Amazon
- Microcenter

Thank You

Desktop Environments

- Before DE there was the CLI

GNOME	KDE	XFE
MATE	BUDGIE	LXDE
Cinnamon	Openbox	LXQT
Pantheon	Deeping DE	Fluxbox

Desktop Environments (cont)

- GUI: A Graphical User Interface
- DE: A Desktop Environment

- Different Linux Distributions ship with different desktop environments. The most common being GNOME and KDE.
    - Desktop Settings
    - Displays Manager
    - File Manager
    - Icons
    - Favorites Bar
    - Launcher
    - Menus
    - Panels
    - System Tray
    - Widgets
    - Window Manager

The GNOME Desktop Environment

The GNOME DE

- The default desktop in Ubuntu is GNOME 3.
    - Debian
    - Fedora
    - Red Hat Enterprise Linux
    - Oracle Linux
- GNOME was an acronym for **GNU Network Object Model Environment** .
- GNOME was started on **August 15, 1997, by Miguel de Icaza and Federico Mena**.

Authors

 authors

Other Desktop Environment

The KDE Desktop Environment

 KDE

The XFCE Desktop Environment

- XFCE is a lightweight desktop environment that aims to be fast and low on system resources, while still being appealing and user friendly.
- The XFCE project was started by Olivier Fourdan in 1996.  XFCE

The Mate Desktop Environment

- The Mate is the continuation of GNOME 2.  Mate

The Cinnamon Desktop Environment

 cinnamon

The LXQT Desktop Environment

The Deepin Desktop Environment

The Pantheon Desktop Environment

The Raspberry Pi OS Desktop Environment

Sources

Useful Resources

The Bash Shell

The History of the Bash Shell

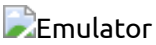
- Shells make large-scale IT possible.
- They're a necessary component to modern computing.

The Linux Terminal

- CLI- A Command-Line Interface
  - They are **Two ways** to access the CLI:
    - *Terminal Emulator*
    - *Linux Console*

Console Terminals

Terminal Emulator



The Bash Shell

- The GNU bash shell is program that provides interactive access to the Linux system

Bash shortcuts | Command Editing Shortcuts

 bash1  bash2  bash3

Shell Prompt

 shellPrompt

Let's Try some basic commands

- **date** *displays the current time and date*
- **cal** *displays a calendar of the current month*
- **df** *displays the current amount of free space on our disk drives*
- **free** *displays the amount of free memory*
- **uname** *displays information about your system*
- **clear** *clears the screen*

Command history

 command history

Sources

 sources

Hoe to navigate the filesystem

The Linux Directory Structure

 linuxDir

The Nemo file manager

 nemo

Note!

> There are a bunch of file mangers options for Linux.

Navigating the FS in the CLI

The Linux Directory Structure

 dirStruc  dirStruc2

Commands to move around the filesystem

 comdMoveAroun

The pwd command

> Displays the current working directory

The Cd Command

> Changes the current working directory. cdCmd cdCmd2

Bash Features

- **Tab Completion** - autocompletes a command by pressing the tab key
- **Arrow Keys** - allows you to move, edit, and repeat commands
- **Ctrl + a** - go to the start of the command line
- **Ctrl + e** - go to the end of command line

Listing Files and Directories

lsFiles

Examples of LS command

- **ls** - List the content of the present working directory
- **ls -a** - List all the files inside the current working directory including hidden files.
- **ls -a ~/Pictures** - List all the files inside a given directory.
- **ls -lr ~/Pictures** - Long list all the files inside a given directory recursively.

examiningls

Special Note

specialNote

Managing Files and Directories

General Knowledge

Options and Arguments:

- Commands are often followed by options that modify/enhance their behavior.
- Commands are also followed by arguments which are the items open which the command acts on.

```
command -option argument   ls -l ~/Downloads
```

Creating files and Directories

Creating directories

The *mkdir* command

- *mkdir* is used for creating a single directory or multiple directories.

> mkdir + the name of the directory.

- To create multiple directories, separate each directory name with a space.
- If you try to create a directory that already exists, you will get an error notifying you that the file already exists.

Examples of the *mkdir* command

- Create a directory in the present working directory
  - `mkdir wallpapers`
- Create a directory in a different directory using relative path
  - `mkdir wallpapers/ocean`
- Create a directory in a different directory using absolute path
  - `mkdir ~/wallpapers/forest`
- Create a directory with a space in the name
  - `mkdir wallpapers/new\ cars`
  - `mkdir wallpapers/'cities usa'`
- Create a directory with a single quote in the name
  - `mkdir wallpapers/"majora's mask"`
- Create multiple directories
  - `mkdir wallpapers/cars wallpapers/cities wallpapers/forest`
- Create a directory with a parent directory at the same time.
  - `mkdir -p wallpapers_others/movies`

Creating Files

- The touch command
  - touch is used for creating files
    - Examples:
      - To create a file called list
        - **touch** list
      - To create several files:
        - **touch** list_of_cars.txt Script.py names.csv
      - To create a file using absolute path:
        - **touch** ~/Downloads/games.txt

> **Note: Creating files is not the designed purpose of the touch command. The touch command updates any given file's timestamp. But, if the file does not exists, it creates it.

Deleting files and directories

- The **rm** command
  - **rm** removes files. Does not removes directories.

- To remove a non-empty directory use **rm** with the **-r** option.
- To remove empty directories use the **rmdir** command.

Examples of the *rm* command

- Remove a file
  - `rm list`
- Remove a file and prompt confirmation before removal
  - `rm -i list`
- Remove all the files inside a directory and ask before removing more than than 3 files
  - `rm -I Downloads/games/*`
- Remove an empty directory
  - `rmdir Downloads/games`
- Remove an non-empty directory
  - `rm -r Downloads/games`

Moving and copying files and directories

Moving files and directoires

The *mv* command

- mv moves and renames directores.
  - **mv** + source + destination
  - **nv** + file/directory to rename + new name
- Both source and destination can be an **absolute or a relative path**.

Examples of moving files and directories

- To move a file from a directory to another using relative path
  - `mv Downloads/homework.pdf Documents/`
- To move a directory from one directory to another using absolute path
  - `sudo mv ~/Downloads/theme /usr/share/themes`
    - *Notice that in this command I am using sudo since the destination is owned by root.*
- To move a file from one directory to another combining absolute path and relative path
  - `mv Downloads/english_homework.docx /media/student/flashdrive/`
    - *Notice that in this command I am moving the file "english_homework.docx" to the directory where the flash drive is mounted.*
- To move multiple directories/files to a different directory
  - `mv games/ wallpapers/ rockmusic/ /media/student/flashdrive/`

Examples of renaming files and directories

- To rename a file
  - **mv** homework.docs cis106homework.docx
- To rename a file using absolute path
  - **mv** ~/Downloads/homework.docs ~/Downloads/cis106homework.docx
- To move and rename a file in the same command
  - **mv** Downloads/cis106homework.docs Documents/new_cis106homework.docx

Copying files and directories

The *cp* command

- **cp** copies files/directories from a source o a destination.
- the **cp** command used the same structure as the **mv**
  - > **cp** + files to copy + destination
- To copy directories you must use the **-r** option
  - > **cp -r** + directories to copy + destination

Examples of coying files and directories

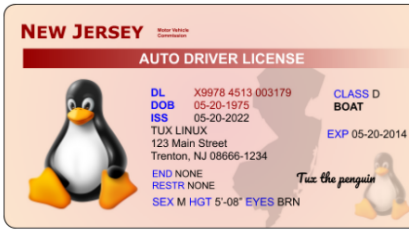
- To copy a file
  - `cp Downloads/wallpapers.zip Pictures/`
- To copy a directory with absolute path
  - `cp -r ~/Downloads/wallpapers ~/Pictures/`
- To copy the content of a directory to another directory
  - `cp Downloads/wallpapers/* ~/Pictures/`
- To copy multiple files in a single command
  - `sudo cp -r script.sh program.py home.html assets/ /var/www/html/`

Working with links

Inodes (index files)

What is an inode?

- A data structure that contains all the information about a file except the file name and its content
- Every file in the file system has an inode
- Each inode is identified by an inode number (index number)
- An inode number references an entry in the inode table
- The inode table is a database of the location of the data on a Linux partition
- To view a file's inode number, use the ls -i command
- To display the inode data on a file or directory use the stat command
- Examples:
  - `stat script.sh`
- Each partition on a hard drive has its own inode table and every file has a unique inode number.



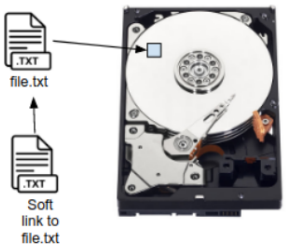
Think of an inode as an ID for each file in the file system that does not have the file name or content.

Hard Links

Soft Links

- **Symbolic links (soft links)** are a special type of file that point to other files instead of data in the hard drive
- Soft links do not share the same inode number as hard link do
- If you modify a soft link, the target file is modified too
- The advantage of soft links is that they can point to files that are store on different partitions
- To create a symbolic link: `ln -s file fileSL`

IMPORTANT: *The file system must support symbolic links in order for you to create links in it. FAT32 does not support links!*



If the concept of soft links is not clicking, think of them as shortcuts in Windows

Getting Help

- Man (manual) pages are documentation files that describe Linux shell commands, executable programs, system calls, special files, and so forth.
- To view the manual of a command type: **man** + command.
    - > Example: **man ls**
- To exit the **man** page press letter "**q**".

Section	Description	Examples
1	Executable programs or shell commands	`man ls`, `man pwd`
2	System calls, which are system requests that programs make to the kernel	`man kill`, `man read`
3	Library calls (to access functions in program libraries)	`man xcrypt`, `man stdin`
4	Special files, such as the floppy disk, that are usually found in `/dev`	`man fd`, `man tty`
5	File formats and conventions	`man passwd`, `man hosts`
6	Games	`man tetravex`, `man AisleRiot`
7	Macro packages and conventions	`man man (7)`, `man gruff (7)`
8	System administration commands	`man yast`, `man suseconfig`

Other ways of getting help

- Most command have a help option built in. Normally that option is accessible by using -h or --h or --help
- Read error messages carefully. Most of the time the answer is in the error itself.
- Some commands may not have a man page but an info page. To read the info page of a command use the info command in a similar way of how you use the man command.
- You can use the apropos command to see the manual page name and description of a particular command
- You can use the whatis command to display a simple description of what a command does.
- There is also a nice program called cheat that you can install in your system to see examples of what a particular command does.
- To learn more about cheat visit the github page: https://github.com/cheat/cheat
- To instal cheat use the command: **`sudo snap install cheat`**
- To use cheat just type cheat + the command. If there is a cheat page available, it will be displayed
- And as it is the case with almost everything, GOOGLE IT! In this case google the error message.

Using *wildcards* (file globbing)

Using Wildcards / File Globbing

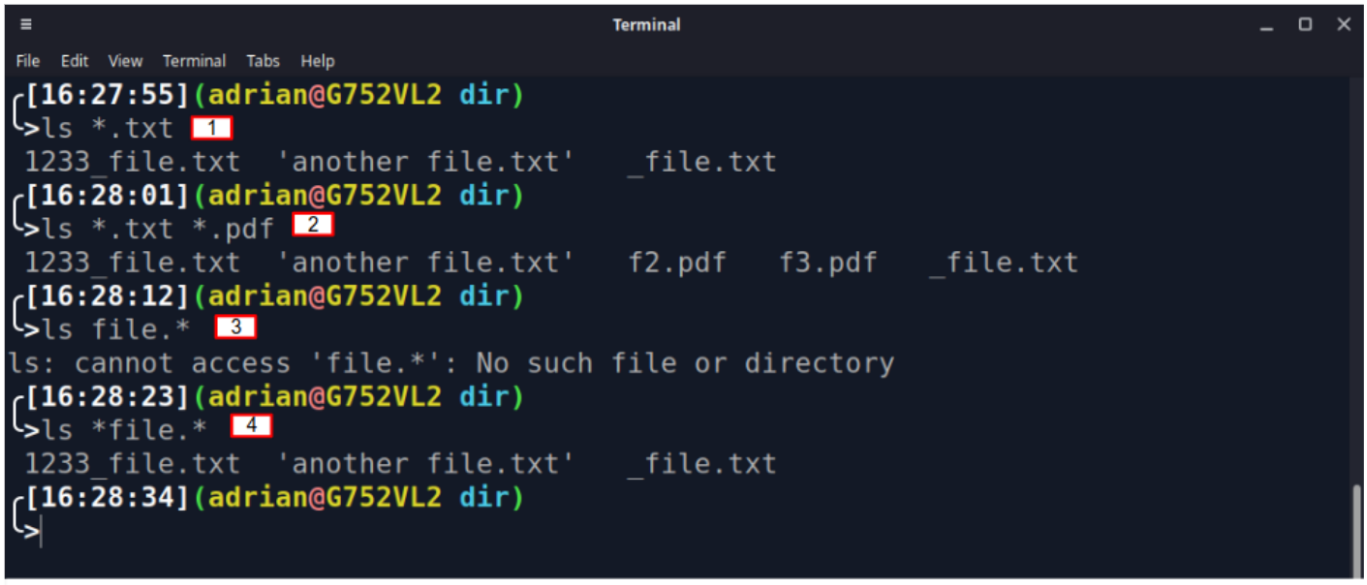
- Wildcard represents letters and characters used to specify a filename for searches.
- File globbing is the processing of pattern matching using wildcards.
- The wildcards are officially called metacharacter wildcards.
- For Example:
  - You can use a wildcard to get a long list of all files in the current directory starting with "new."
  - Use wildcards to manage to directories faster
  - Move or delete a group of files
  - Locate files based on a portion of their filenames
  - Create files and directories quicker

Important note:
Wildcards are not regular expressions. A regular expression is a sequence of characters that define a search pattern.

The *Wildcard

- The main wildcard is a star, or asterisk (*) character.
- The **star alone** matches anything and nothing and matches any number of characters.
  - For example, **ls *.txt**will match all files that end in **.txt**.
  - Examples of when to use **the wildcard:*
    - to list all files with a particular file extension.
    - Do not remember the complete name of a file but you remember a prtion of the name.
    - Want to copy, move, or remove all files that match a particular naming convention.

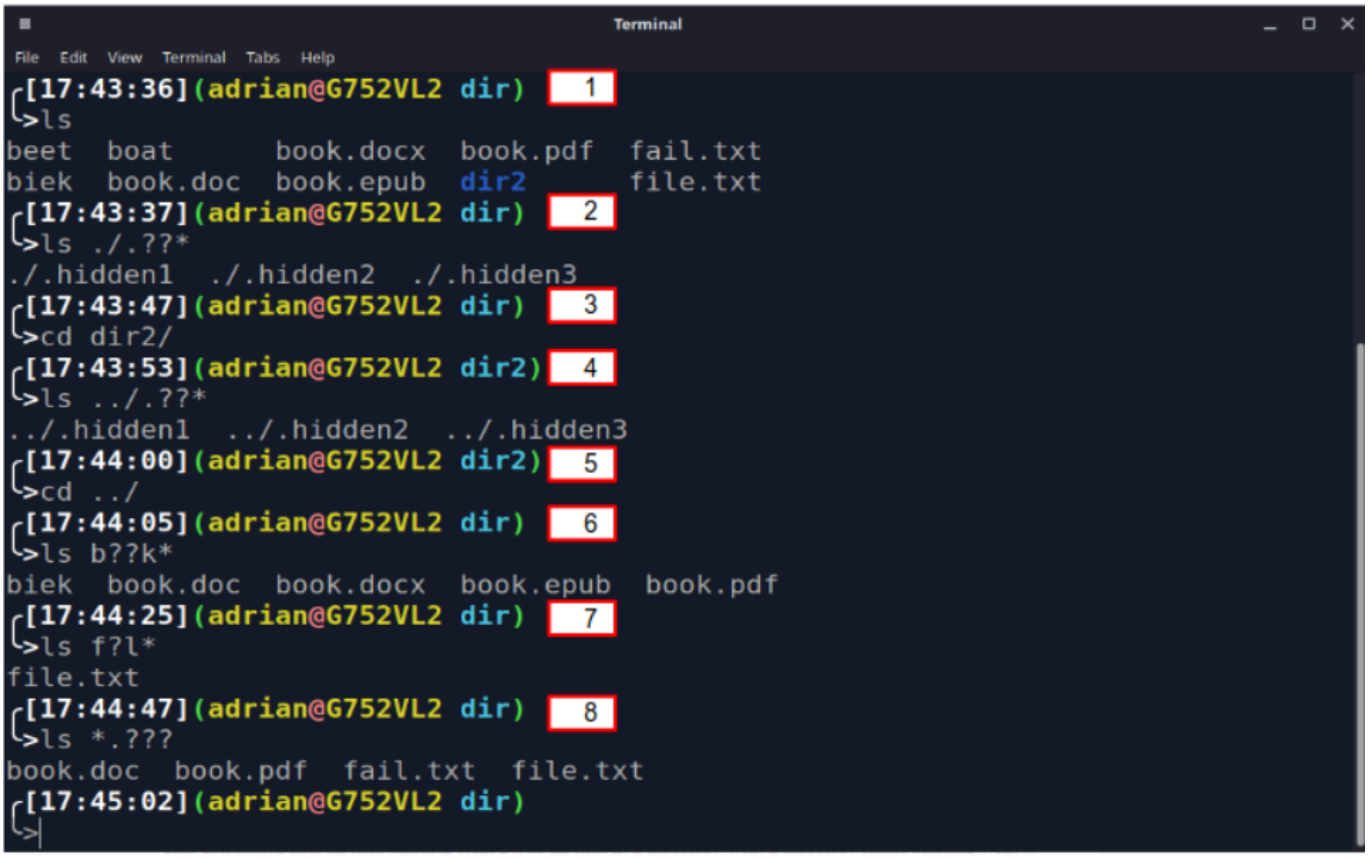
Examples of *Wildcard



1. ls *.txt lists all the files that end in .txt
2. ls *.txt *.pdf list all the files that end in .txt and .pdf
3. ls file.* lists all the files that start with the string "file." regardless of their file extension. In this example, there were no files in the directory that matched this criteria.
4. ls *file.* list all the files that have any letter before the string "file." and after as well.

The *? Wildcard*

- The ? wildcard metacharacter matches **precisely one character**. You might need the question mark to minimize a long list of files names down to a few.
- In addition, the question mark proves very useful when working with hidden files (hidden files are also called **dot files**).
- If you want to list all hidden files you can use: `ls .??*` which will match all files that start with a . or .. and have any character after it.
- Isn't this the same as using the * character on its own? **NO!**
  - The problem with dot files and wildcard expressions is that the **current directory** and the **parent directory** have a name.
  - The current directory is called/represented with a single dot (.) and the parent directory is called/represented with two dots (..)
    - *Remember cd ../../../ that's what I am talking about!*
  - If you use a wildcard expression such as `.*` to list all files that start with a dot. The shell will also match . and ..
  - To go around this problem you can use `./.*` to match all the files in the current directory with a file name starting with a dot.
  - You can also match all the files that start with a . in the parent directory using `../.*`



1. List all the files in the current directory (excluding hidden files)
2. List all the hidden files in the current directory
3. Changes the current working directory to dir2
4. List all the hidden files in the parent directory
5. Changes the current working directory to the previous directory (dir)
6. List all the files that have a two character between letter b and k.
7. List all the files that have a single character between letter f and l.
8. List all the files that have a 3 letter file extension.

The *[] Wildcard*

- The brackets wildcard match a single character in a range.
- The brackets wildcard use the exclamation mark to reverse the match. For example, match everything except vowels [!aeiou] or any character except numbers [!0-9]
- **Examples:**
  - To match all files that have a vowel after letter f:
    - `ls f[aeiou]*`
  - To match all files that do not have a vowel after letter f:
    - `ls f[!aeiou]*`
  - To match all files that have a range of letters after f:
    - `ls f[a-z]*`
  - To match all files whose name has at least one number:
    - `ls *[0-9]*`
  - To match all the files whose name does not have a number in their file name:
    - `ls *[!0-9].*`
  - To match all files whose name begins with a letter from a-p or start with letters s or c:
    - `ls [a-psc]*`
  - To match all files whose name begins with any of these two sets of characters: letters from a-f or p-z:
    - `ls [a-fp-z]*`
  - To match all files whose name begins with any 3 combination of numbers and the current user's username:
    - `ls [0-9][0-9][0-9]$USER`

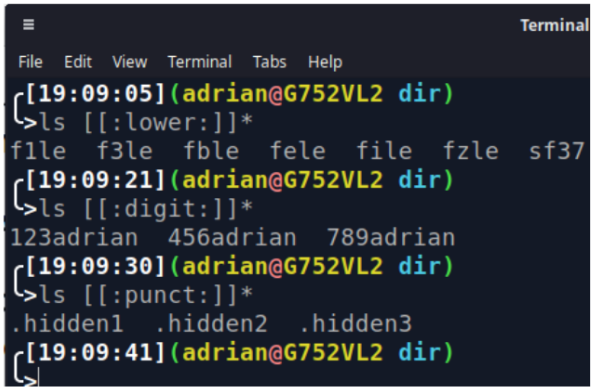
31

![exampleBrak](../notes/imgs4/exampleBrak.png

The [] wildcard with Regex Character Classes

You can use POSIX or Character Classes with the [] wildcard

POSIX class	Represents	Means
[:upper:]	[A-Z]	Upper case letters
[:lower:]	[a-z]	Lower case letters
[:alpha:]	[A-Za-z]	Upper and Lower case letters
[:alnum:]	[A-Za-z0-9]	Lower case, upper case, and digits
[:digit:]	[0-9]	digits
[:xdigit:]	[0-9A-Fa-f]	hexadecimal digits
[:punct:]	[.,!?:...]	puctuation
[:blank:]	[\t]	space and tabs
[:cntrl:]	n/a	control characters
[:graph:]	[^\t\n\r\g\v]	printed characters without spaces
[:print:]	[^\t\n\r\g\v]	printed characters including spaces
[:space:]	[\t\n\r\g\v]	whitespace characters



Using Wildcards / File Globbing (quick reference)

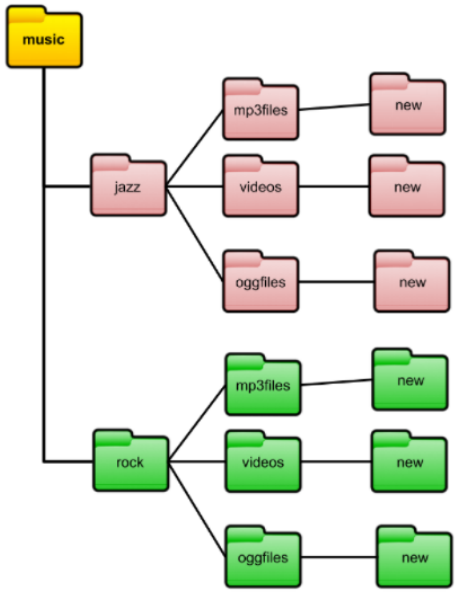
Wildcard	Description
*	Matches zero or more characters in a filename
?	Matches any one character in a filename
[acf]	Matches one of multiple characters in a filename; in this example, a, c, or f
[a-f]	Matches one of a range of characters in a filename; in this example, any character from a through f
[!a-f]	Matches filenames that don't contain a specified range of characters; in this example, filenames that don't contain a through f

Using Brace Expansion

- Brace expansion {} is not a wildcard but another feature of bash that allows you to generate arbitrary strings to use with commands.
- For example,
  - To create a whole directory structure in a single command:
    - `mkdir -p music/{jazz,rock}/{mp3files,vidoes,oggfiles}/new{1..3}`
  - To create a N number of files use:
    - `touch website{1..5}.html`
    - `touch file{A..Z}.txt`
    - `touch file{001..10}.py`
    - `touch file{{a..z},{0..10}}.js`
  - Remove multiple files in a single directory
    - `rm -r {dir1,dir2,dir3,file.txt,file.py}`

Here is a fun example!

Try to replicate this file structure in a single command



Bonus

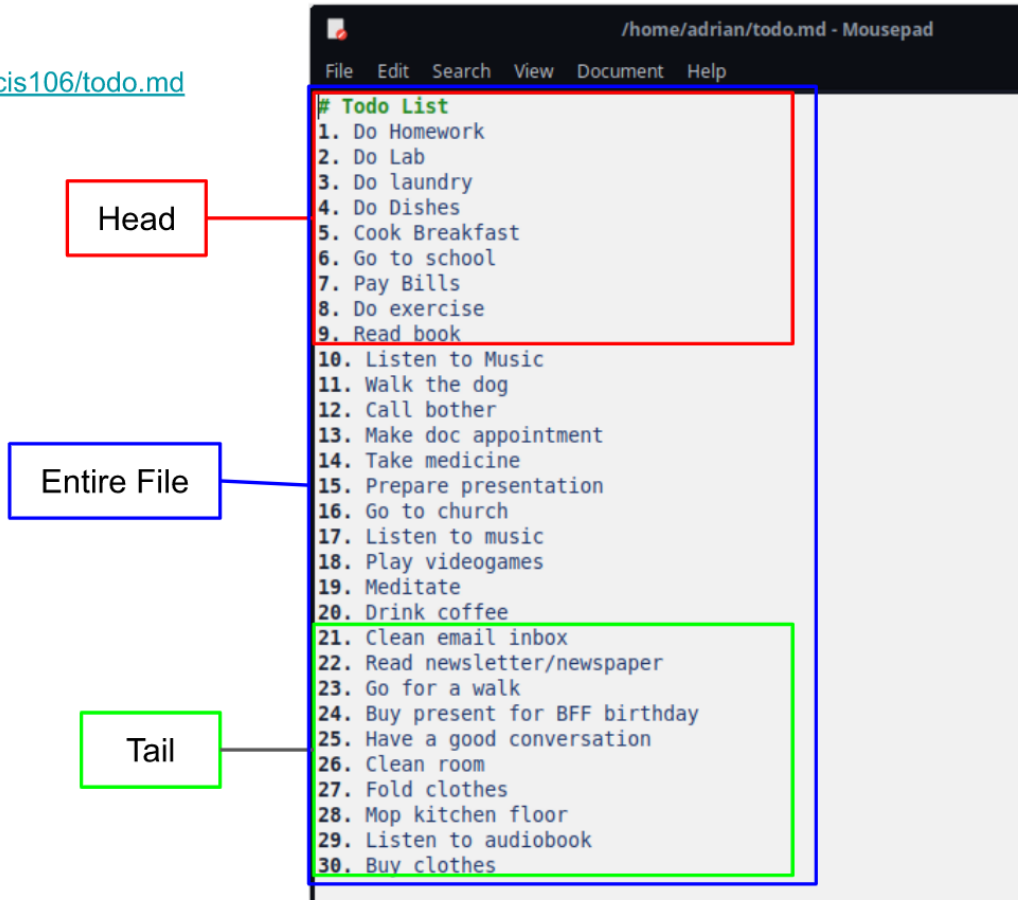
- **ranger** - visual file manger
  - **Installation:** *sudo apt install ranger*
  - **Usage: ranger**
- **Nnn**
  - **Installation:** *sudo apt install ranger*
  - **Usage: ranger#** Handling Text Files

Handling text files

- Linux offers a lot of command line tools for handling text
  - cat

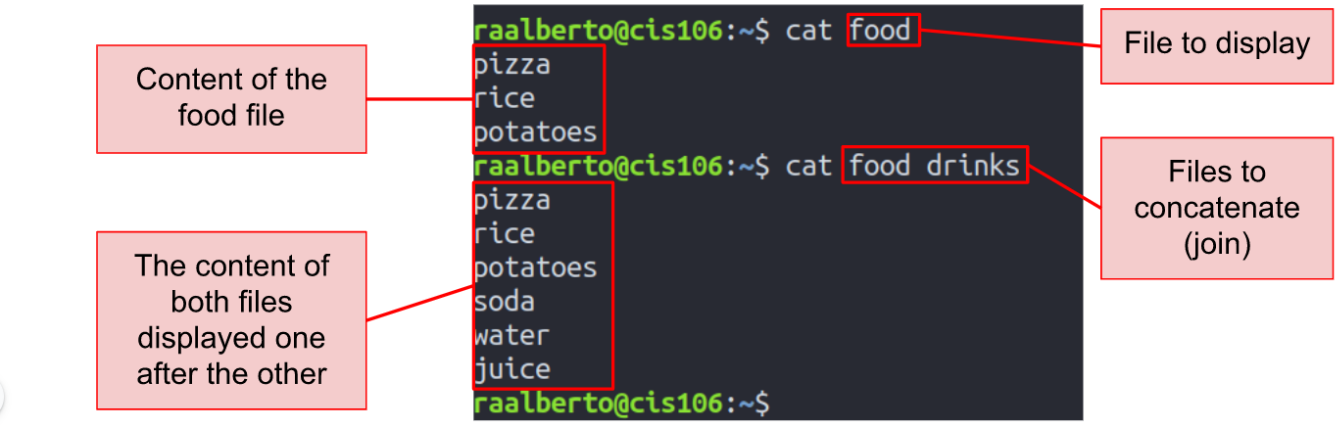
- tac
- more
- less
- head
- diff
- tail
- cut
- past
- sort
- wc
- tr
- grep

To download this file:
https://robertalberto.com/cis106/todo.md



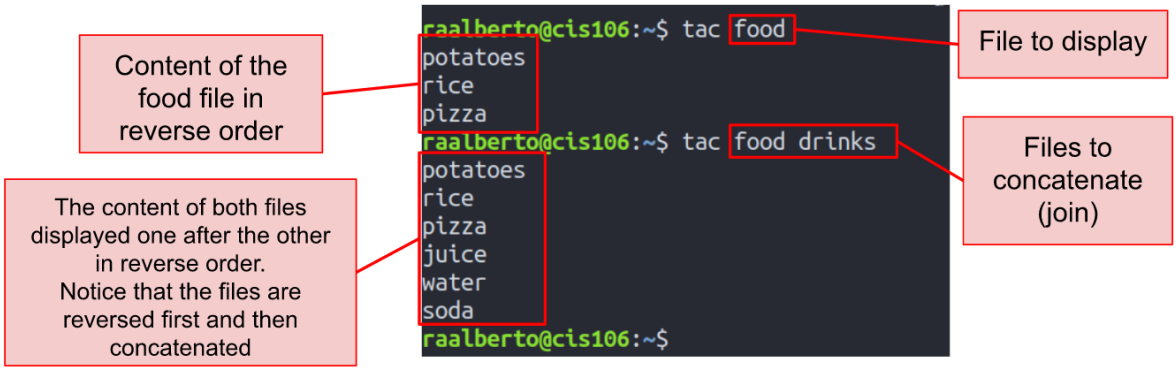
Cat

- The cat command is used for **displaying the content of a file**.
- Cat is short for **concatenate** which is the the command intended use.
- Concatenation means joining two strings together.
- Usage:
  - **cat + file to display**
  - **cat + file 1 + file 2**



Tac

- The tac command is used for displaying the content of a file in **reverse order** in a line by line basis
- The tac command can also **concatenate files** in reverse order
- Usage:
  - **tac + file to display**
  - **tac + file 1 + file 2**



More

- The **more** command is a pager program used for displaying the content of text file one page at a time.
- Usage:
  - > **more** + file to view.
  - > *For more information, read the man page of the more command.*

Less

- The **less** command is another pager program that displays the content of file 1 page at time.
- **Less** is faster than more when dealing with large files since it loads 1 page at time.
    - Usage:
        - **less** + file to view
        - *For more information, read the man page of the more command.*

Head

- The **head** command displays the top **N** number of lines of a given file.
- By default, it prints the first 10 lines.
    - Usage:
        - **head** + **option** + **file**
        - *For more information, read the man page of the more command.*

Tail

- The **tail** command displays the las **N** number of lines a given file.
- By default, it prints the last 10 lines.
    - Usage:
        - **tail** + **option** + **file**
        - *For more information, read the man page of the more command.*

cut

- The cut command is used to extract a specific section of each line of a file and display it to the screen.
- Usage:
    - `cut + option + file`
- Examples of the cut command:

Displays the first field of each line, using tab as the field separator.

```
cut -f1 hostnames.txt
```

Displays the first field of each line, using : as the field separator.

```
cut -d : -f1 /etc/passwd
```

This command is a nice way of displaying a list of users in your linux system.

See more examples here: https://robertalberto.com/linuxcommands/commands/cut.html

Paste

- The paste command is used to join files horizontally in columns.
- Usage:
  - `paste + option + files`
- Examples of the paste command:

Merge two files

```
paste users.txt ips.txt
```

Merge two files using a different delimiter

```
paste -d ":" users.txt ips.txt
```

Merge files sequentially instead of horizontally

```
paste -s users.txt ips.txt
```

Sort

- The sort command is used for sorting files.
- Sorting means arranging the content of the file in a particular order.
- The sort command sorts the contents of a text file, line by line.
- The sort command supports sorting:
  - Alphabetically
  - in reverse order
  - by number
  - by month
- You can use sort for sorting by column number.
- Sort can be used ignoring case sensitivity
- Sort can return whether a file is sorted or not (this is really useful in scripts)
- By default, the sort command interprets a blank space as the default field separator.
- **The sort command follows this order unless specified otherwise:**
  - Lines starting with a number will appear before lines starting with a letter.
  - Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
  - Lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.

Assuming you have a file with a list of users. Sort the file.

```
sort users.txt
```

wc

- The **wc** command is used for printing the number of lines, and bytes in a file.
- Usage:
  - **wc + option + file**

Tr

- The **tr** command is used for translating or deleting characters from standard output.
  - **standard output | tr + option + set + set**

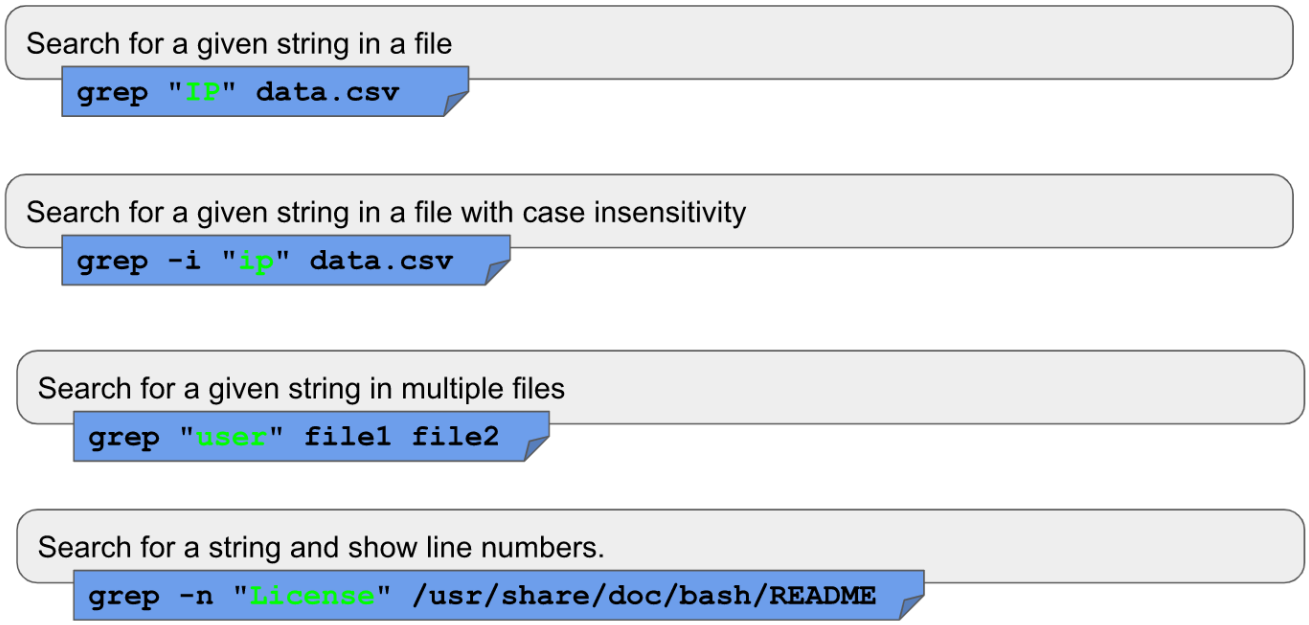
Diff

- The **diff** command compares files and displays the differences between them
    - diff + option + file1 + files2

Grep

- The **grep** command is used to match a string pattern from a file or standard output when using the pipe
    - grep + option + pattern to match + file
    - standard output + pipe | + grep + pattern to match

Examples of the grep



Rev

- The **rev** command is used for reversing the characters position in a given text.
    - **rev** + file

Working with I/O Redirection.

- In Inux, we can redirect the **input and output** of commands to and from files, as well as connect multiple commands together into powerful command piperlines.

Redirecting STDOUT and STDERR

- To redirect standard output, we use: >
  - **Example:**
    - `ls -lax > list_of_files.txt`
- To redirect standard error, we use: 2>
  - **Example:**
    - `cat badFile.txt 2> error_cat_command.txt`
- To redirect standard output and append the output to a file, we use: >>
  - **Example:**
    - `ls -1alh >> list_of_files.txt`
- We can use the output redirection to create an empty file:
  - **Example:**
    - `> newfile`
    - `: > newfile` (in older versions of bash)
- We can also get rid of output that we do not want:
  - **Example:**
    - `ls -l ~/Downloads ~/documents 2> /dev/null`

Something interesting about the cat command

What happens when you use cat without any file?

- In the absence of filename arguments, cat copies standard input to standard output, which means that cat will take any text that you type and display it in the terminal. Looking like this:

- To exit press CTRL + d (hold down the Ctrl key and press “d”).
- You can redirect the output of cat and essentially use cat to create a file with any given text you want.

The pipe (|)

The pipe (|)

- The pipe allows you to redirect the standard output of a command to the standard input of another.
- Usage:
  - `command 1 + | (pipe) + command 2 + | (pipe) +command #`
- Examples of the pipe:

Use grep to look for a string in a particular man page

```
man ls | grep "human-readable"
```

Display only the options of the of any command from its man page

```
man ls | grep "^[[:space:]]*[[:punct:]]"
```

Display all IP addresses from the output of the ip command

```
ip addr | grep -Eo '[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}\.[[:digit:]]{1,3}'
```

Alias

Creating you own commands with alias

- **What is alias?** - A shorthand for a more complicated command.
- How to create an alias?
  - `alias name_of_alias="command here"`
- **`Examples:`**
  - `An alias to upgrade a linux (debian system):`
    - `alias update="sudo apt update; sudo apt upgrade -y; sudo apt full-upgrade -y"`
  - `An alias to obtain a computers public ip:`
    - `alias publicip="curl ifconfig.me && echo ''"`
  - `Some useful git aliases:`
    - `alias add="git add ."`
    - `alias push="git push"`
    - `alias merge="git merge"`
    - `alias pull="git pull"`
    - `alias checkout="git checkout -b"`
    - `alias commit="git commit -m"`
    - `alias qpush="git add .; git commit -m 'quick push'; git push"`

- Before creating an alias, always check to see if the word you will use is reserved. Use the type command to figure it out.
- For example:
  - type random - random is a reserved word because there already is a command using it.
  - type randomXYZ - is a good choice because there isn't a command using it.
- If you make an alias to a reserved word, you may break your system because the original command will no longer be used when using the alias.
- To make your aliases permanent, place them either at the end of your .bashrc file or in Ubuntu, place it in the file .bash_aliases located in your home directory.
- Aliases are a good way of remembering tough commands.
- To remove an alias, use the unalias command
  - unalias quoteanime

Sources

Sources

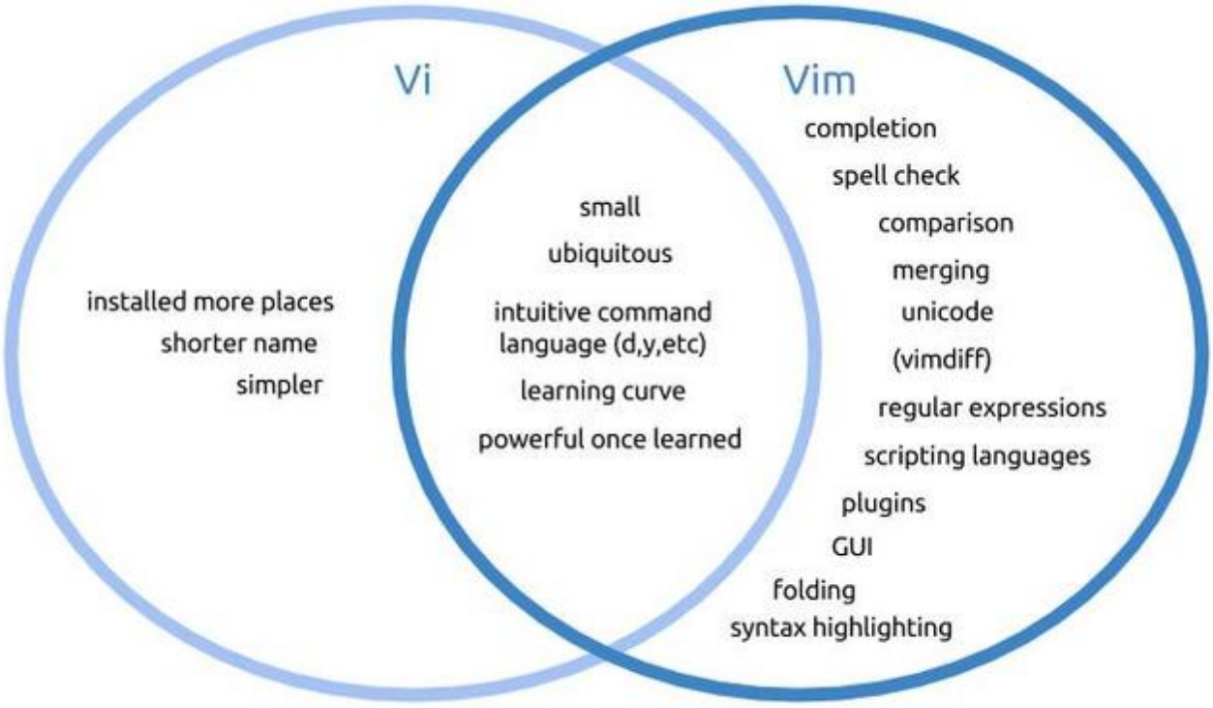
- Blum, Richard, and Christine Bresnahan. *Linux Command Line and Shell Scripting Bible*. John Wiley & Sons, 2021.
- Shotts, William E. *The Linux Command Line: a Complete Introduction*. No Starch Press, 2019.

Vim

What is WIM

- The **vi command-line** text editor is included in all POSIX compliant OS.
- The **vi command** is now linked to the **vim command**.
- Even when you issue the **vi command**, you're actually starting the **vim editor**.
  - To install vim
    - `sudo apt install vim`

- If **vi** is installed in all Linux distros, Why am i learning **VIM**?
  - vim had more features
  - vim is also light weight

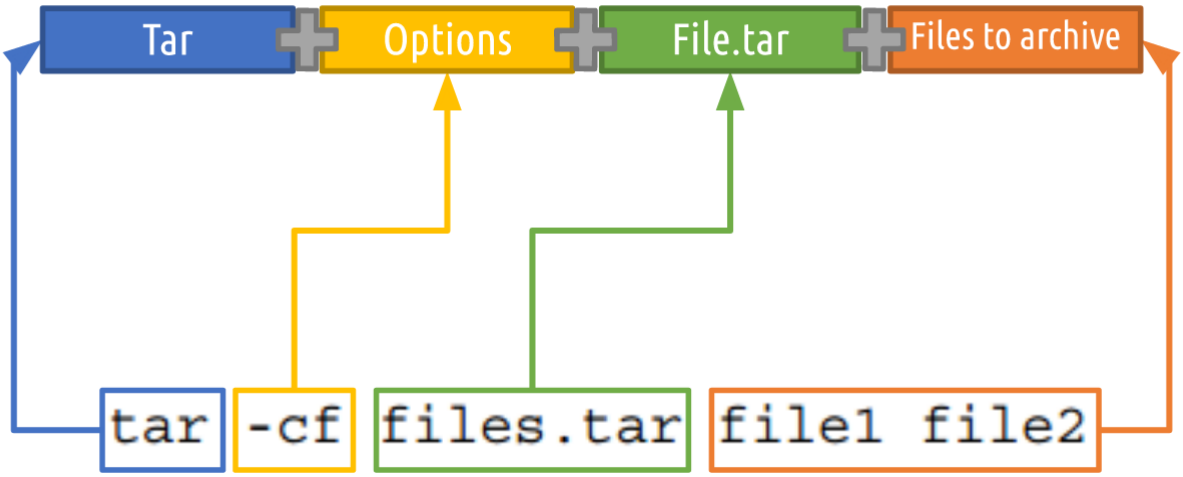
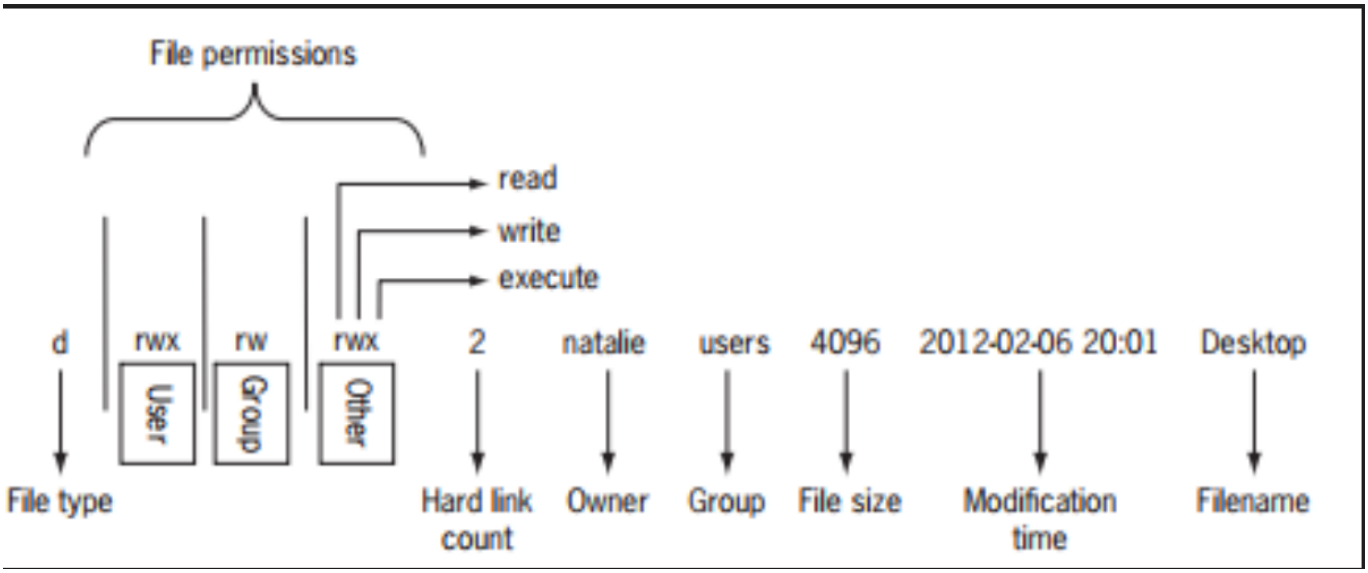
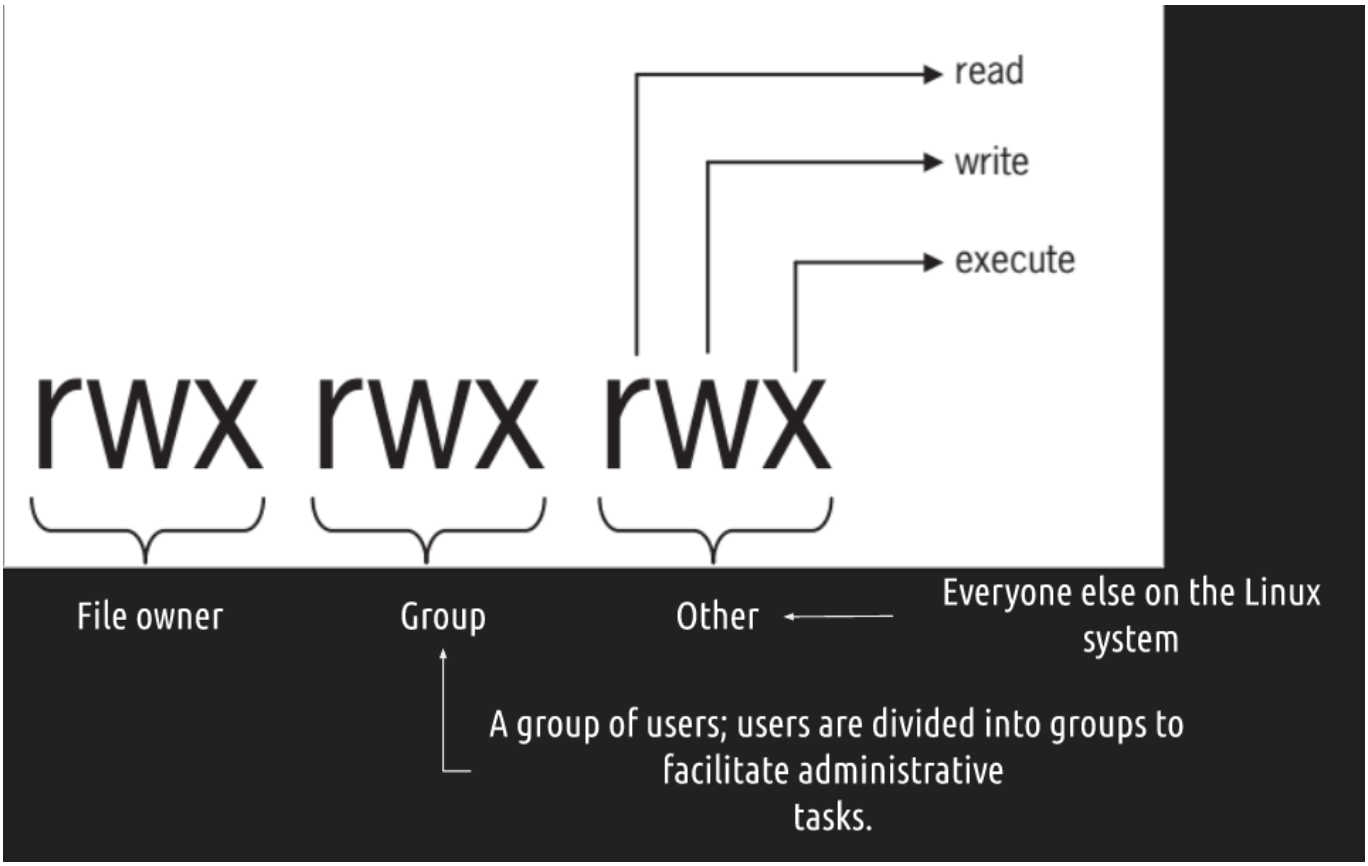
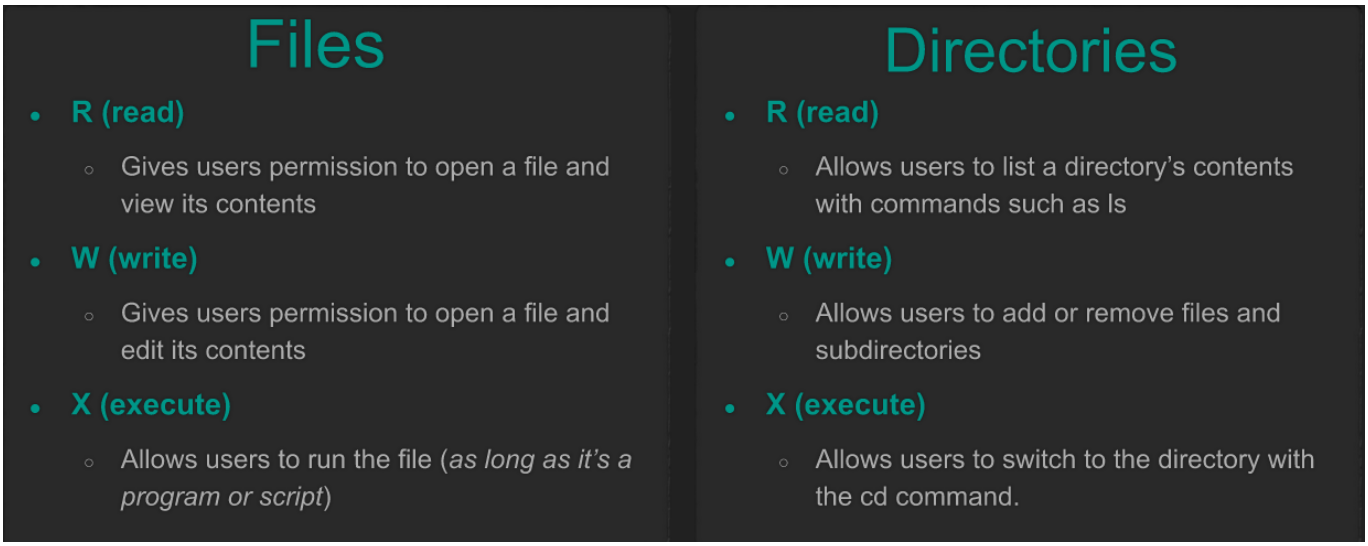
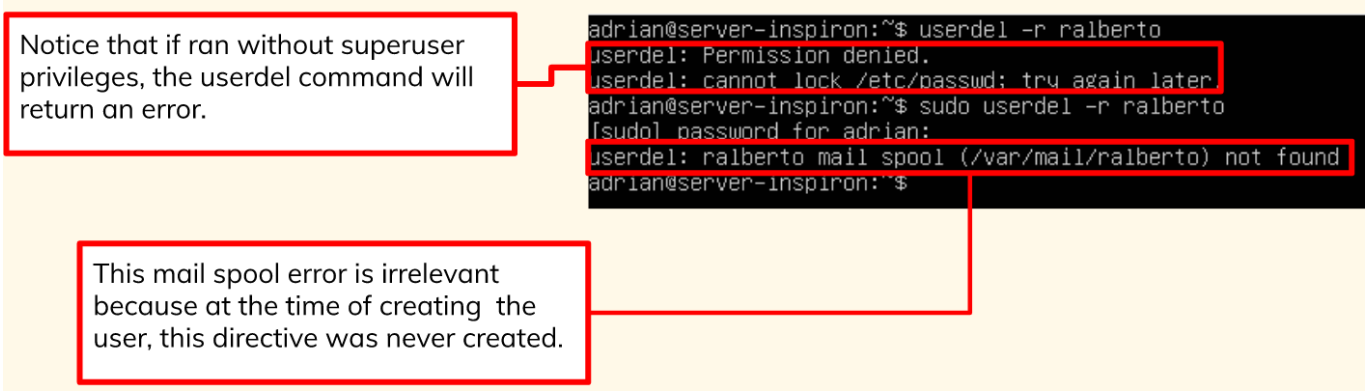


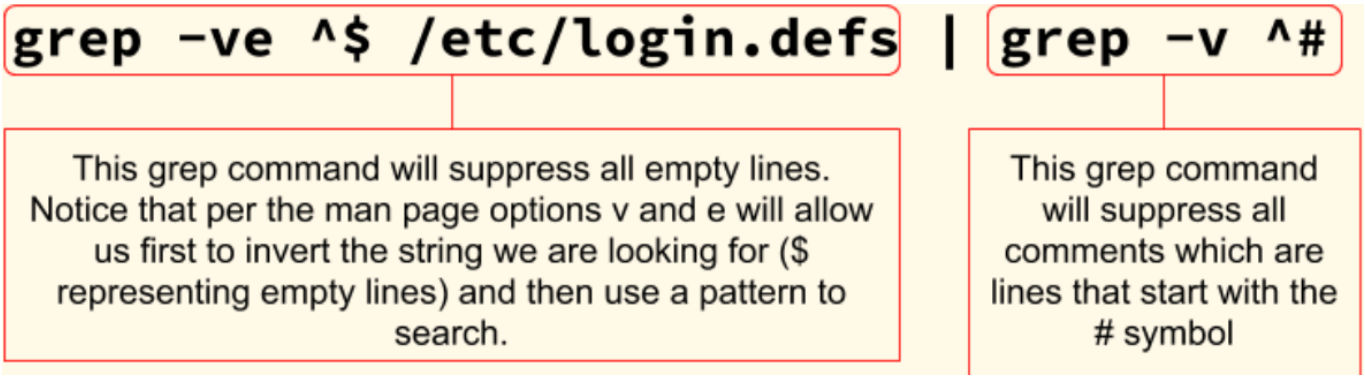
Vim modes:

- **Insert mode:** used for writing text
- **Normal mode:** used for manipulating text
- **Command mode:** used for entering vim commands
- **Ex-mode:** Similar to the command-line mode but optimized for batch processing.
  - vim starts in normal mode
  - From normal mode **press i** to enter insert mode. The world **--INSERT--** will appear.
  - To switch back to normal mode **press esc**.
  - In the lack of the **esc key press ctrl + c**.

Insert text:

- you can create a file and open **vim** at the same time by typing **vim** and a file name.
  - Example:
    - ```
      vim notes.txt
      ```
  - In insert mode, you can use:
    - **The arrow keys** to move around,
    - **Enter Key** to continue in the next line,
    - **Backspace** for deleting.

Saving and quitting vim
      ```

- To save a text file you need to enter **normal mode** using: and the use the **w** key.
  - **:w** will save the file
  - **:w new.txt** will save the file as new.txt
  - **:wq** will save the file and quit
  - **:wqa!** will save the file and close all files open in the buffer

Editing a file with wim

- You can tell vim that you want to edit another file by using the e command
- `:e new.txt` -> will open `new.txt` and allow you to edit
- You can use auto completion here
- Ctrl + g will show the file that you are currently editing in the status line
- You can also use :f in command mode to see the file that you are currently working on.

Navigating a file

- In normal mode use the keys:
  - H = left
  - J = down
  - K = up
  - L = right
- You can prefix the number of times by adding the number after the letter 10H will move 10 character to the left.

Moving around Words, sentences, and paragraphs

- To move between words use w e
  - `w -> moves word by word to the beginning of each word`
  - `e -> moves word by word to the end of each word`
- You can prefix the number of words you want to move
- 10e will move 10 words
- To move between sentences use ()
  - (-> previous sentences
  -) -> next sentence
- To move between paragraphs use {}
  - { -> previous paragraph
  - } -> next paragraph
- *A paragraph beginnings after an empty line and ends in an empty line.*

Searching words in wim

- Use / and the word you are looking for to search forward
  - `/hello`
- letter n will repeat the search for the next word
- ? To search backward
  - `?hello`
- * will search for the next occurrence of the word under the cursor
- # will search backward for the previous occurrence of the word under the cursor

Screen movement

- G **uppercase g** Moves to end of the file
- gg **2 lowercase g** moves to the beginning of a file
- **ctrl + f** moves a page forward at a time
- **ctrl + b** moves a page backward at a time

Moving to Lines

- To move to a specific line use : plus the line number
  - :8 will move you to line 8
  - Additionally use 8G
- $ will move to the end of the line
- 0 will move to the beginning of the line
- `vim sample.txt +100` will open sample.txt and move to line 100
- + executes any vim command from the shell prompt

Delete text and copy and paste

- dw = delete current word
- u = undo
- dd = delete line under the cursor
- d + /word = delete until the word given
- yw = copy the current word
- p = for paste after the cursor
- P = for paste before the cursor
- yy = copies a whole line
- x = for cut

Useful to Know

- Read files
  - Shift o enters a new empty line
  - :r file name = insert the text of the file given into the file being edited
- Create a vim custom file
  - In your home directory create a file named .vimrc and add the commands to that file
  - http://learnvimscriptthehardway.stevelosh.com/
- Run an external command
  - :!+command
- To run a command and paste in a file
  - :r !+command

Managing Data

Basic Terminology

- **Backup:** Copies files and directories to an archive
- **System Backup**
- **Archive**
- List of important directories to include in system backup:
  - /etc
  - /home
  - /opt
  - /root
  - /var

Archiving utilities

- **Tar (tape archive):** create archives by combining files and directories into a single file.
- **CPIO:** Create an archive, restores files from an archive, copies a directory hierarchy.
  - Create (copy-out) mode places multiple files into a single archive file
  - Extract (copy-in) mode restores files from an archive
  - Pass-through (copy-pass) mode copies a directory hierarchy.
- **Ar:** creates, modifies, and extracts form archives.

The tar program

- To create an archive
  - `tar + option + archive name + files to add to archive`
    - The **option-f** is always required.
    - Files inside an archive are called members.
- To extract an archive:
  - `tar + options + file to extract`

Tar command example explained



The CPIO program

The CPIO program

Cpio requires a list of files to archive. The option to create an archive is -o

- `ls | cpio -ov > archive.cpio`

To extract an a archive to cpio use the -i option with <

- `cpio -iv < archive.cpio`

Archive specific files

- `find . -iname *.sh | cpio -ov > scriptsArchive.cpio`

Create a tar archive with cpio

- `ls | cpio -ov -H tar -F sample.tar`

Extract *.tar Archive File using cpio

- `cpio -idv -F sample.tar`

View the content of *.tar Archive File

- `cpio -it -F sample.tar`

The ar utility

The GNU **ar** program creates, modifies, and extracts from archives.

- Archive files with **ar**
  - ```
    ar r test.a *.txt
    ```
- List content of an archive
  - ```
    ar t test.a
    ```

- Add a new member to an archive
    - ```
      ar r test.a test3.txt
      ```
- Delete a member from archive
    - ```
      ar d test.a test3.txt
      ```

File Compression

- Gzip (GNU Zip) has better compression ratios than compress does and can uncompress files that were compressed using the compress command.
- The **`gzip`**, **`bzip2`**, and **`xz`** commands are used for compression.
- When you compress a file with any of these tools the result is a file with a similar name but with the correspondent file extension.
- **Example:**
    - `file.txt ----> file.txt.gz`
    - `file.txt ----> file.txt.bz2`
    - `file.txt ----> file.txt.xz`

File Compression | GZIP, BZIP2, XZ

- Gzip, bzip2, and xz compress files in place meaning the original file is deleted after compression.
- **`bzip2`** offers better compression ratios in comparison to gzip.
- **`xz`** produces better compression ratios than gzip and bzip2.

Important:

Do not confuse gzip with zip. Zip is used to pack and unpack zip archives containing several files compressed into a single file that has been imported from or is being exported to a Windows system.

File Compression | zip, 7zip, and rar

- Zip is an archiving and compression utility.
- To use zip: **`zip + archiveName.zip + files to include in archive`**
- Example: **`zip allmyfiles.zip file1 file2 file3`**
- To unarchive use: **`unzip archive.zip`**

- 7-Zip is an open source, cross-platform, and fully-featured file archiver with a high compression ratio.
- To use 7zip on linux you need the package: **p7zip-full**
- The general formula to use 7z is: **`7z + option + fileName.7z + file(s) to archive`**
- See next slide for examples

- RAR is a proprietary archive file format developed by Eugene Roshal. The command unrar allows Linux users to extract rar archives. The command rar allows you to create rar archives
- To use unrar: **`unrar + option + filename.rar`**
  - Example: **`unrar x games.rar`**
- To use rar: **`rar + option + archivename.rar + files to archive`**
  - Example: `rar a archivename.rar file2 file2`

Linux File Permissions

Linux File Permissions | File Ownership

- A file can be owned only by one user and one group.
- The **/etc/passwd** file contains a list of all the users in Linux.

Ls -l output review



Linux File Permissions



Linux File Permissions | Files vs Directories



Linux File Permissions | The chmod command

- the **chmod (change mode)** command is used to change permissions on files and directories.
- It has this Syntax: **chmod permissions file/directory**
- You can use it in **two** ways to change file permissions:
  - Symbolic notation
  - Numeric notation

Linux File Permissions | Symbolic Notation

Table 5-2 Symbolic notation

Category	Operator	Permission
u (user)	+ (add to existing permissions)	r (read)
g (group)	- (remove from existing permissions)	w (write)
o (other)	= (assign absolute permissions)	x (execute)
a (all)	One of the preceding operators	One or more of the preceding permissions

Examples:

- `chmod u+x script.sh`
- `chmod o-x script.sh`
- `chmod u=rwx,g=rw,o=r script.sh`

Linux File Permission | Numeric Notation

Table 5-3 Numeric notation

Permission	Numeric value
---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rwx	7

Permission	Value
Read	4
Write	2
Execute	1

Example:

```
chmod 766 script.sh
chmod 700 script.sh
chmod 555 script.sh
```

Managing Users and Groups

Managing User Accounts

- Managing user accounts involves adding, modifying and deleting user accounts and account's information
- To add user accounts we use the `useradd` or `adduser` command.

  In Ubuntu, the `adduser` program is recommended over `useradd` due to useradd being a low-level utility.
- To modify user's information we use the `usermod` program.
- To delete a users we use the `userdel` program.
- The following files are involved in the user creation process:
  - `/etc/login.defs` `/etc/default/useradd` `/etc/skel/`
  - `/etc/passwd` `/etc/shadow` `/etc/group`

- How do I add a user in Ubuntu?
  - Run the command **sudo adduser** followed by the **username**.
- How do I delete a user in Ubuntu?
  - To delete a user use the **userdel -r** command followed by the **username**.
    - The **-r** option for the command to delete the user and its home directory.



- Understanding the purpose of these files is canonical to the understanding of how users and groups work on in Linux.
  - /etc/default/useradd
  - /etc/passwd
  - /etc/group ...

The /etc/login.defs file

- It contains directives for use in various **shadow password suite commands**.
- **Shadow password suite** is an umbrella term for commands dealing with account.
  - ```
    grep -ve ^$ /etc/login.defs | grep -v ^#
    ```
    ```



Shell Scripting

Creating a basic script

- Start vim, enable line numbers, and enter insert mode.
- Type:

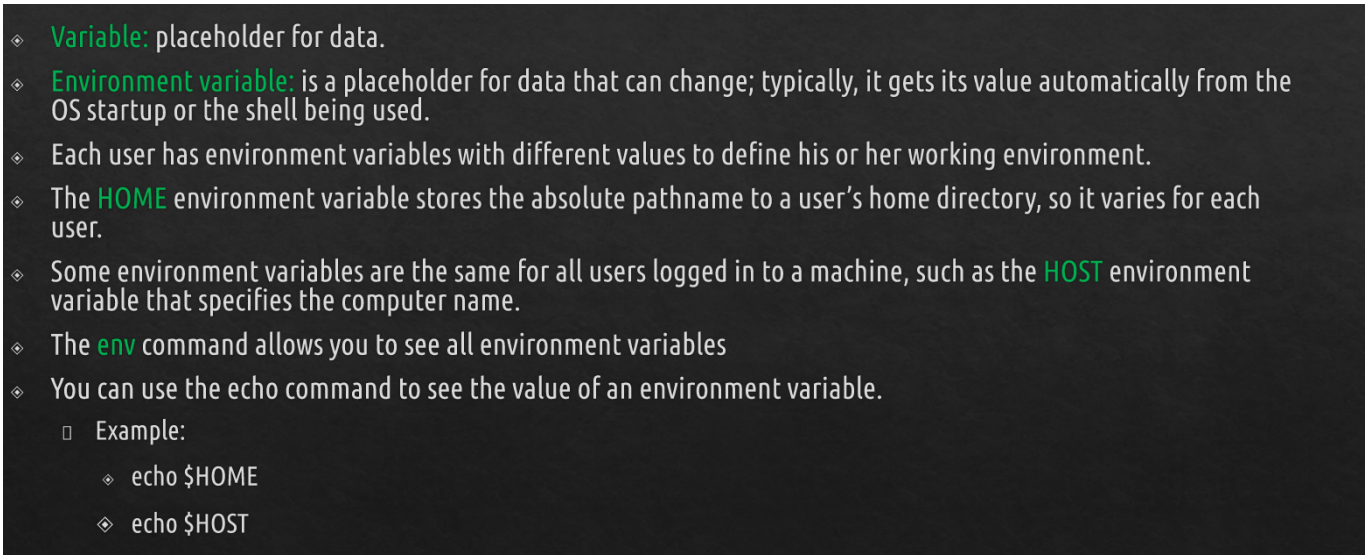
```
#!/bin/bash
echo " to display info about your Linux system"
uname -a
```

> echo -n **does not output a new line**.

- Save the file and name it "name.sh"
- Type: **chmod u+x name.sh** to make the file executable.
- To run the script type: **./name.sh**

Working with variables

Shell scripting | Variables



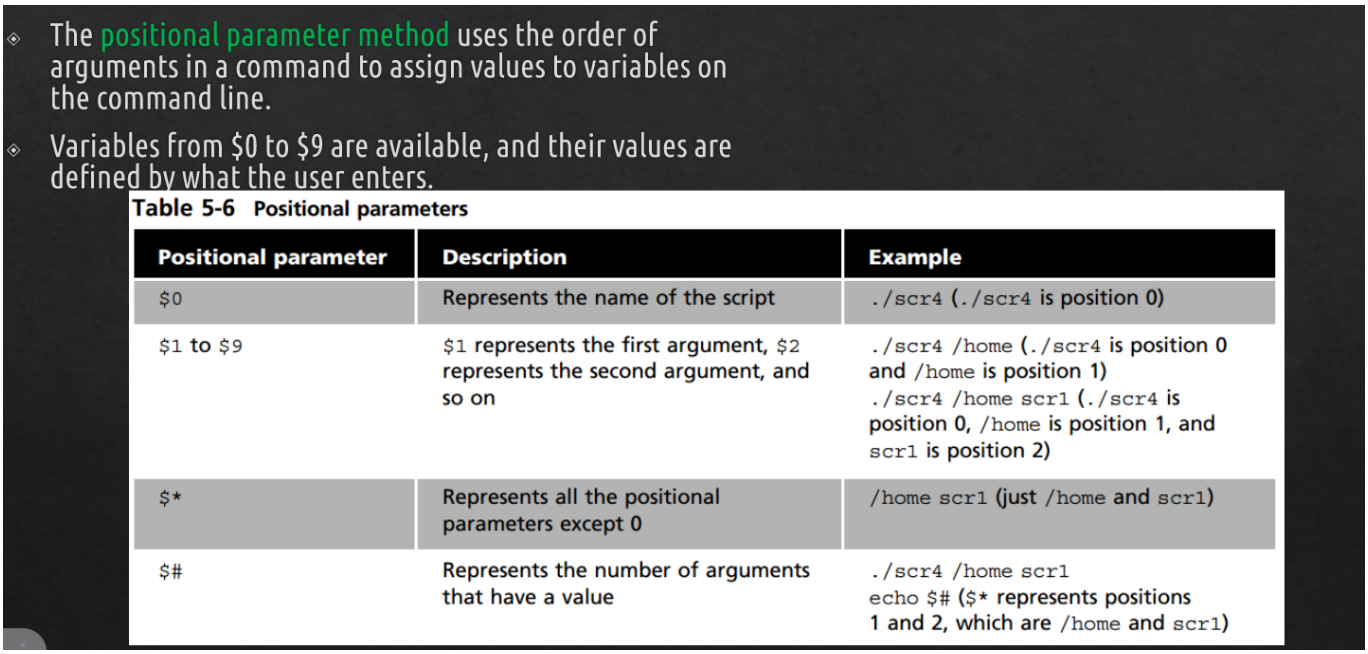
- A **shell variable** is similar to an environment variable, but its value is usually assigned in a shell script.
- Shell variables can be created in **two ways**:

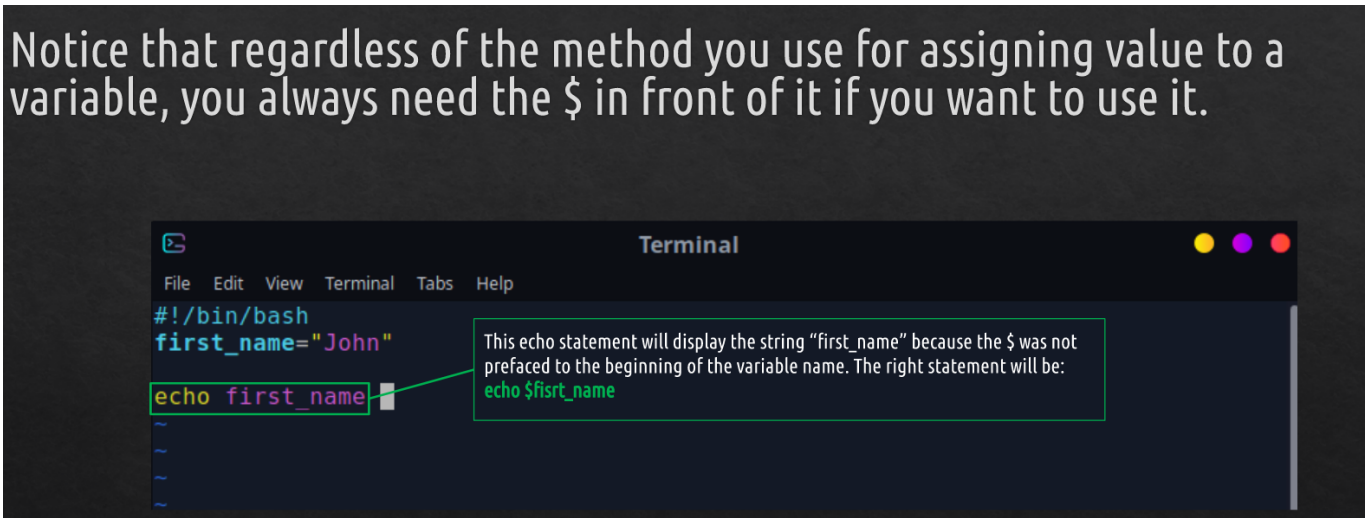
Direct Assignment:

```
color = blue
```

The Prompt MEthod:

```
echo "Enter a color:";read color
```





- You can use **curly braces** to reference a variable's value: **${variable_name}**
- It is useful if you want to **append a string to a variable**.
- **Command Substitution:** allows the output of a command to replace the command itself.

Can be done in **TWO WAYS:**

```
$(command)
`command`
```

Shell scripting | Exit Status Codes

Exit status code: a number sent to the shell when you run a command.

Type:

```
#!/bin/bash
cd baddir
echo $?
```

- Successful commands usually return the **code 0**, and failures return a value **greater than 0**.
- to see an exit status use the **$? variable**.

Using Structured Commands

if-then-Else statements, nesting if statements, test, compound testing, and case statements

Shell scripting | Conditions

- The if statement is used to carry out certain commands based on testing a condition and the exit status of the command.
- If statements are used to control how the script will execute.
- For instance, you might want a portion of the script to run if the user is in the Marketing Department and have another portion run if the user is in Human Resources.
- Or you may want to run another command if the a given command executes successfully.
- if statement—Starts the condition being tested
- then statement—Starts the portion of code specifying what to do if the condition evaluates to true
- else statement—Starts the portion of code specifying what to do if the condition evaluates to false

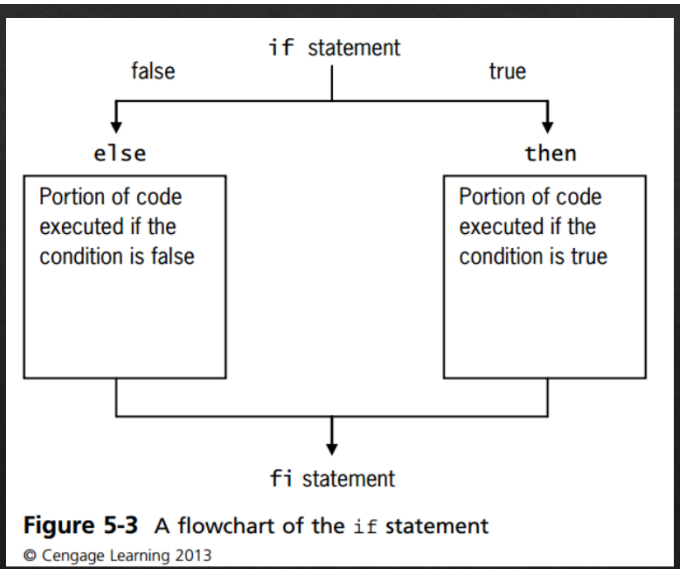


Figure 5-3 A flowchart of the `if` statement

How does if then works in bash?

IMPORTANT NOTE:

- if statements in bash are different than in if statements in other languages.
- In other programming languages, the object after the if statement is an equation that is evaluated for a TRUE or FALSE value.

The bash shell `if` statement runs the command defined on the if line.

If the exit status of the command is zero (the command completed successfully), the commands listed under the `then` section are executed. If the exit status of the command is anything else, the then commands aren't executed, and the bash shell moves on to the next command in the script.

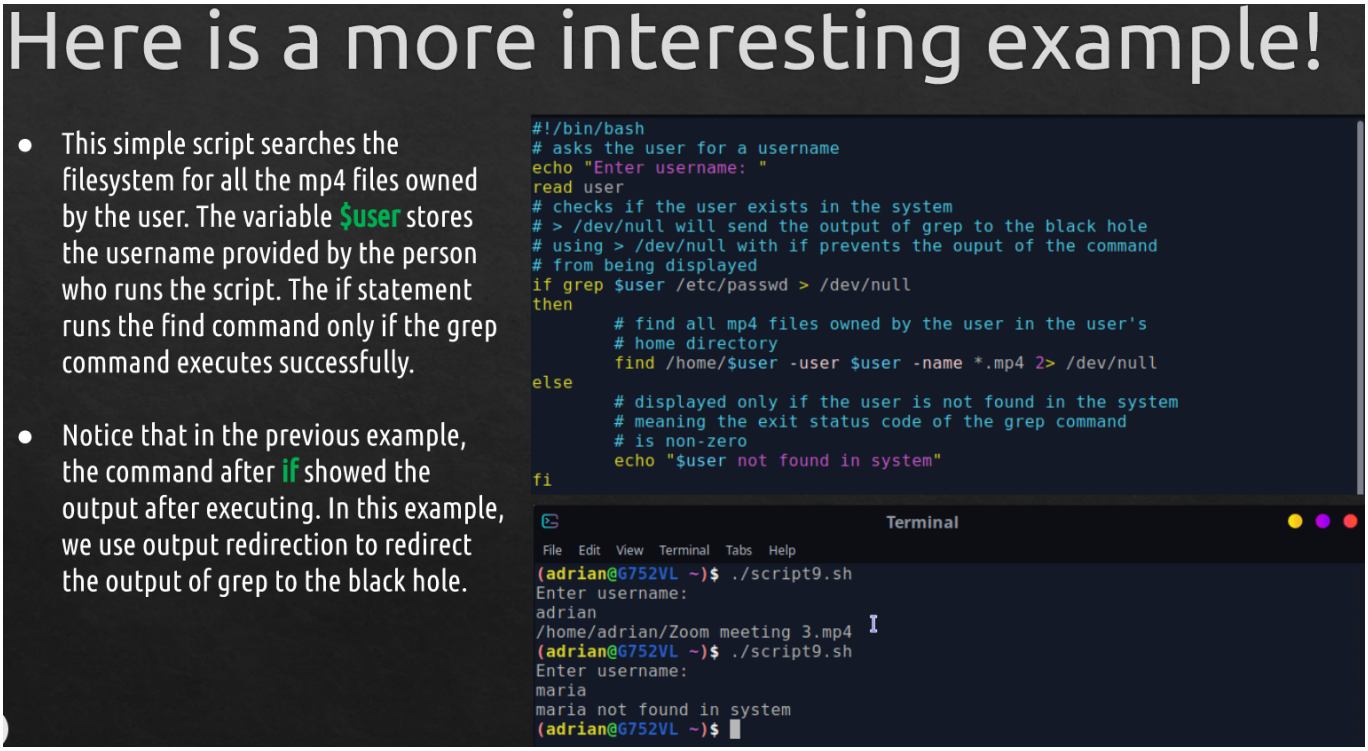
```
if command
then
    commands
fi
```

The `fi` statement indicates the end of the if statement

Example

```
#!/bin/bash
if pwd
then
        echo "it worked"
fi
```

> if the **pwd** command executes successfully, the string **it worked** will be displayed



Shell scripting | Comparison operators

Table 5-7 File attribute operators in the BASH shell

File attribute operator	Description
`-a`	Checks whether the file exists
`-d`	Checks whether the file is a directory
`-f`	Checks whether the file is a regular file
`-r`	Checks whether the user has read permission for the file
`-s`	Checks whether the file contains data
`-w`	Checks whether the user has write permission for the file
`-x`	Checks whether the user has execute permission for the file
`-O`	Checks whether the user is the owner of the file
`-G`	Checks whether the user belongs to the group owner of the file
`file1 -nt file2`	Checks whether `file1` is newer than `file2`
`file1 -ot file2`	Checks whether `file1` is older than `file2`

Numeric Comparison

Comparison	Description	Example
n1 -eq n2	Checks if n1 is equal to n2	If [$n1 -eq $n2]
n1 -ge n2	Checks if n1 is greater than or equal to n2	If [$n1 -ge $n2]
n1 -gt n2	Checks if n1 is greater than n2	If [$n1 -gt $n2]
n1 -le n2	Checks if n1 is less than or equal to n2	If [$n1 -le $n2]
n1 -lt n2	Checks if n1 is less than n2	If [$n1 -lt $n2]
n1 -ne n2	Checks if n1 is not equal to n2	If [$n1 -ne $n2]

String Comparison		
Comparison	Description	Example
str1 = str2	Checks if str1 is the same as string str2	If [$str1 = $str2]
str1 != str2	Checks if str1 is not the same as str2	If [$str1 != $str2]
str1 < str2	Checks if str1 is less than str2	If [$str1 < $str2]
str1 \> str2	Checks if str1 is greater than str2	If [$str1 > $str2]
-n str1	Checks if str1 has a length greater than zero	If [$str1 -n]
-z str1	Checks if str1 has a length of zero	If [$str1 -z]

A **case statement** uses one variable to specify multiple values and matches a portion of the script to each value.

```
#!/bin/bash
clear
echo "------------------------------------------------"
echo -e "\twhich command would you like to run?"
echo "1) ls"
echo "2) ls -a"
echo "3) ls -l"
echo "4) ls -la"
echo "5) ls -laR"
echo "------------------------------------------------"
read answer

case $answer in
        1)
                ls
                ;;
        2)
                ls -a
                ;;
        3)
                ls -l
                ;;
        4)
                ls -la
                ;;
        5)
                ls -laR
                ;;
        *)
                exit
                ;;
esac

                                                11,1          All
```

Shell scripting | Looping

- Looping is used to perform a set of commands repeatedly. In the menu script, the user is given a list of options to choose from, and after a selection is made, the script ends.
- Shell scripting support different types of loops:
  - while loop
  - until loop
  - for loop

```
while [ condition ]
do
        command1
        command2
        commandN
done
```

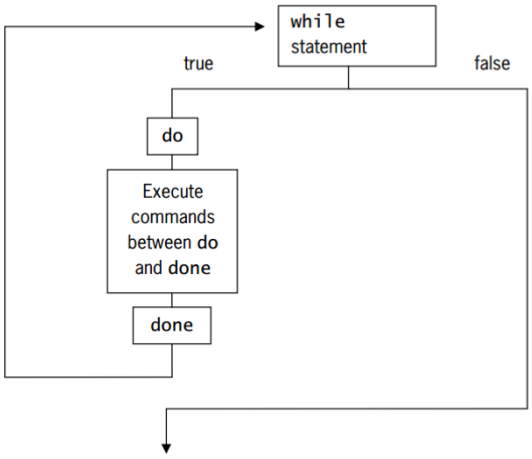


Figure 5-4 A `while` loop

- An until loop repeats the commands between do and done as long as the tested condition is false (exit status code is greater than 0)—in other words, until the condition is true.

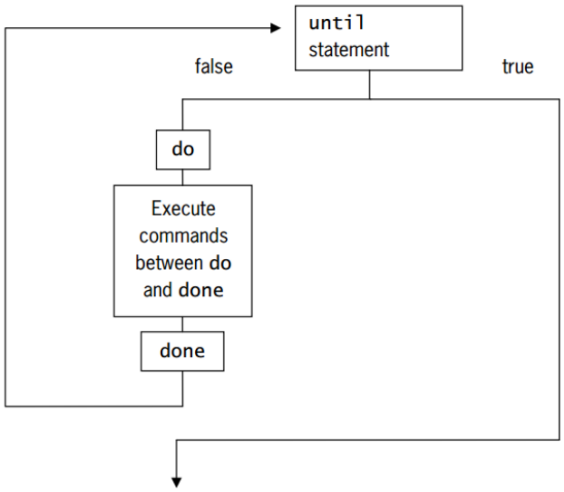


Figure 5-5 An `until` loop

- A for loop repeats the commands between do and done a specified number of times. Each time the script carries out the commands in the loop, a new value is given to a variable.

```
for VARIABLE in 1 2 3 4 5 .. N
do
        command1
        command2
        commandN
done
```

```
for VARIABLE in file1 file2 file3
do
        command1 on $VARIABLE
        command2
        commandN
done
```