Open Source Software Development

Ung Văn Giàu
Email: giau.ung@eiu.edu.vn

Linux Operating System

Introduction

- The Linux OS is **everywhere**

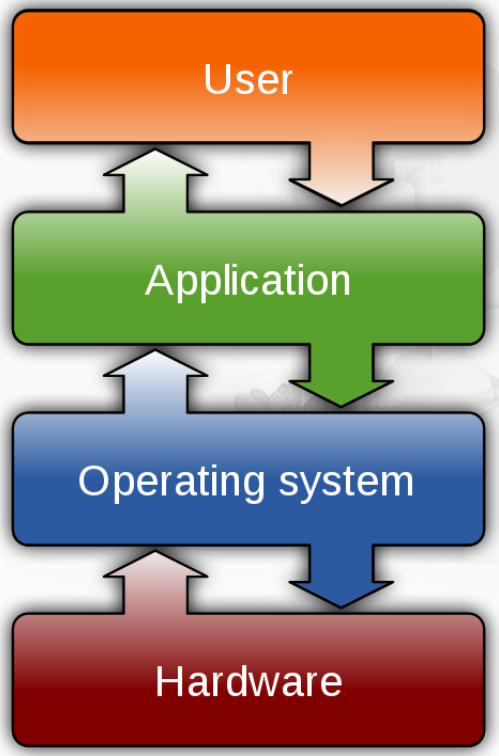
  Smartphones, cars, supercomputers, home appliances, home desktops, enterprise servers,…

- Linux is one of the most **reliable, secure and worry-free** OS available

Contents

LINUX

Review: What is an operating system?



- OS is **software** that **manages all of the hardware resources**
- To put it simply, OS manages **the communication between software and hardware**

What is Linux?

- Linux is an operating system (OS)

- One of the most popular platforms on the planet, **Android**, is powered by the Linux OS

What is Linux?

Linux OS comprises several different pieces:

- **Bootloader**

  The software **manages the boot process** of computer

- **Kernel**
  - The **core of the system** and manages the CPU, memory, and peripheral devices
  - The lowest level of the OS
- **Init system**
  - A sub-system that bootstraps the user space and is charged with controlling daemons
  - Manages the boot process
  - E.g., Systemd

What is Linux?

Linux OS comprises several different pieces:

- **Daemons**
  - Background services (printing, sound, scheduling,...)
  - Either start up during boot or after logging into the desktop
- **Graphical server**

  The sub-system that displays the graphics on the monitor

What is Linux?

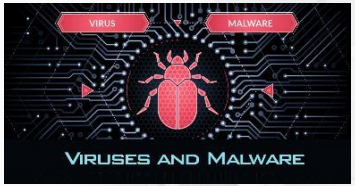
Linux OS comprises several different pieces:

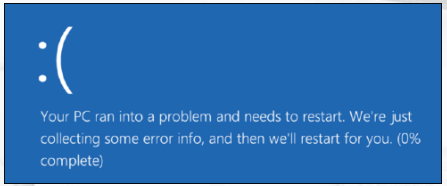
- **Desktop environment**
  - The piece that the users interact with
  - Desktop environments: GNOME (1999), Cinnamon, Mate, Pantheon, Enlightenment, KDE (1996), Xfce, Unity...
  - Each desktop environment includes built-in applications
- **Applications**
  - Most modern Linux distributions include App Store-like tools
  - E.g., Ubuntu Linux has the Ubuntu Software Center

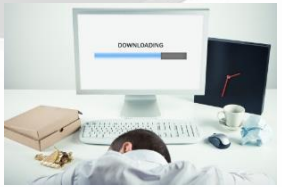
Why use Linux?

- Why bother learning a completely different computing environment?

  When the OS that ships with most desktops, laptops, and servers works just fine?







- Does that OS you're currently using really work?

  Do you find yourself battling obstacles like viruses, malware, slow downs, crashes, costly repairs, and licensing fees?

Why use Linux?

- If yes, Linux might be the perfect platform for you.

  Because

  - Linux has evolved into one of the **most reliable computer ecosystems**
  - **Zero cost of entry**
  - Linux is generally **far less vulnerable** to ransomware, malware, or virus attacks

  - Server **reboots** are only necessary if the kernel is updated

- If you follow the regular recommended updates, stability and dependability are practically assured

Why use Linux?

- Linux is also distributed under an **open source license**

→ Linux is an OS **by the people, for the people**

→ It is also a main factor in why many people choose Linux

freedom and freedom of use and freedom of choice

What is a “distribution”?

- Distribution (distros)

  a complete Linux system package

- Linux has many different versions to suit any type of user

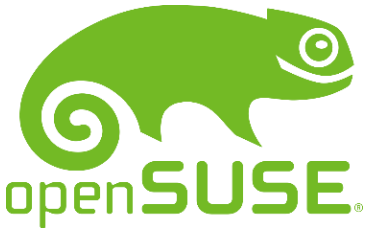
  new users to hard-core users

- Distribution can be downloaded for free, burned onto disk (or USB thumb drive), and installed

What is a “distribution”?

Popular Linux distributions include:

- UBUNTU
- FEDORA
- LINUX MINT
- DEBIAN
- OPENSUSE
- MANJARO
- ANTERGOS
- SOLUS
- ELEMENTARY OS



What is a “distribution”?

- The server distributions include:
  - Ubuntu Server
  - Centos
  - Red Hat Enterprise Linux
  - SUSE Enterprise Linux

Red Hat

CentOS

- Ubuntu Server and CentOS distributions are free
- Red Hat Enterprise Linux and SUSE Enterprise Linux have an associated price

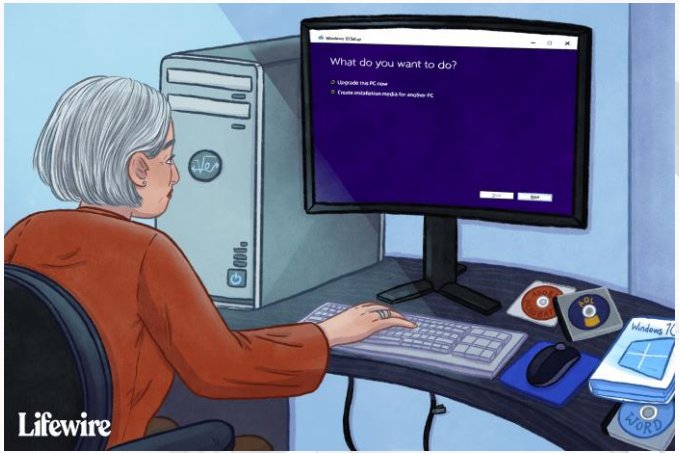
  Those with an associated price also include support

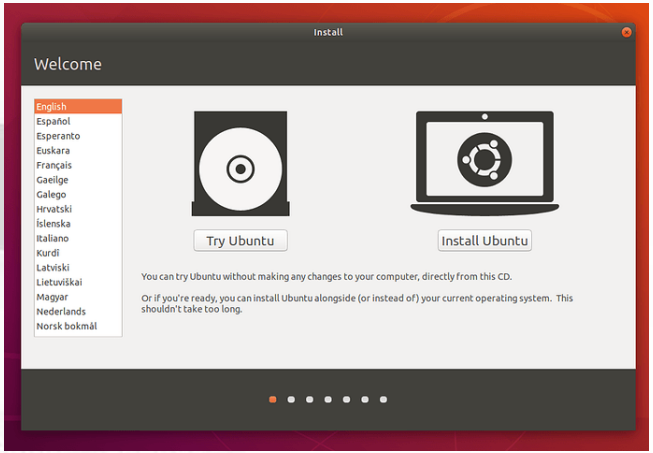
Which distribution is right for you?

It will depend on the answer to three simple questions:

- How skilled of a computer user are you?
  - Basic: Linux Mint, Ubuntu, Elementary OS or Deepin
  - Professional: Debian or Fedora
  - Master: Gentoo
- Do you prefer a modern or a standard desktop interface?
- Server or desktop?

Installing Linux





Linux offers one of the easiest installations of all OSs

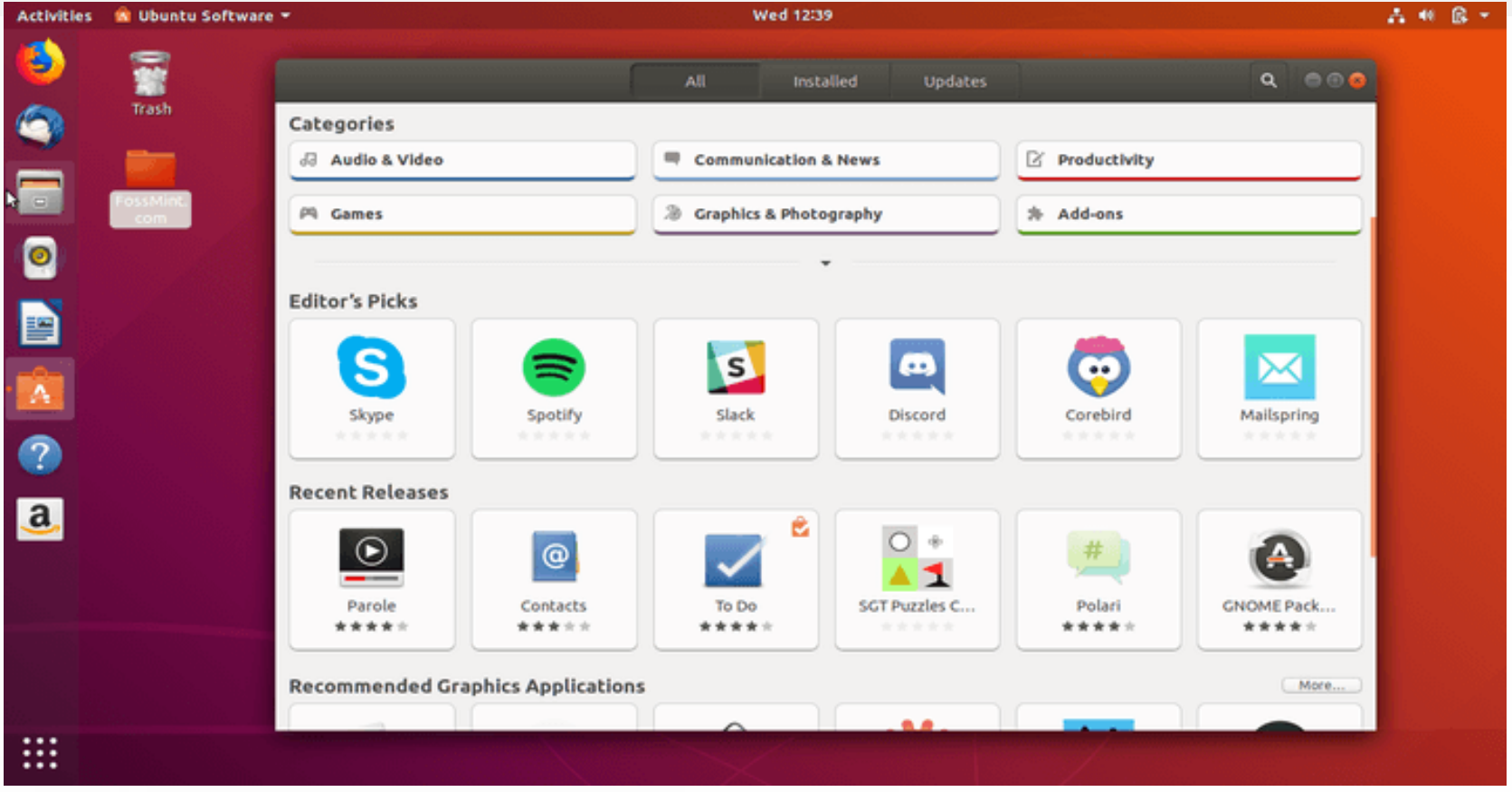
- **Live distribution:** run the operating system from either a CD/DVD or USB flash drive
- Once you've tried it out, and if you want, you simply double-click the "Install" icon

Installing software on Linux

- Most modern Linux distributions include what most would consider an app store
  - a centralized location where software can be searched and installed
  - E.g.:
    - ✓Ubuntu Linux (and many other distributions) rely on GNOME Software, Elementary OS has the AppCenter,
    - ✓openSUSE has AppStore

- For GUI-less servers, depend upon the command-line interface for installation

Installing software on Linux



Installing software on Linux

- The Debian-based distros will use the apt-get tool for installing software

sudo apt-get install wget

- Fedora-based distros will require the use of the yum tool

su yum install wget

What is Difference Between Linux and Unix

Linux	Unix
Open source	Mixed
Linux refers to the **kernel**	Unix refers to the original **OS**
Original code developed by Linus and the GNU Foundation	Original code developed by AT & T
Designed as a general-purpose scalable platform for a broad set of applications	Typically designed for a narrow audience with a defined set of target platforms and applications
Free community support. Paid support available from a number of service providers.	Paid commercial support
Interfaces often evolve	Interfaces often stable
Debian, Ubuntu, Fedora, Red Hat, Android,…	IBM AIX, Solaris, HP-UX, Darwin, macOS X,…

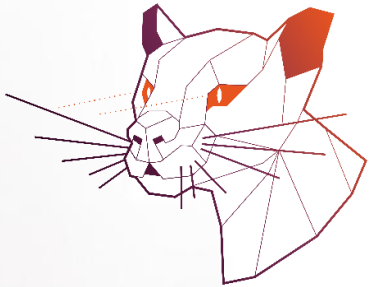
Summary

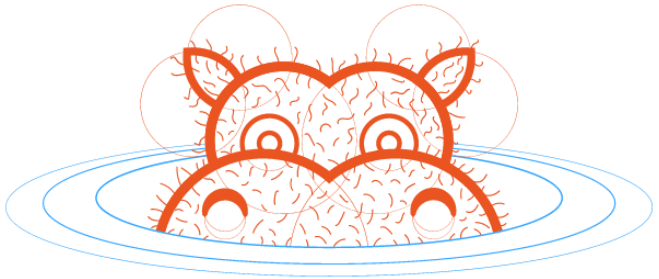
- Linux is not Unix, but it is a Unix-like OS

- **Linux is a UNIX clone** developed from scratch by Linus Torvalds and team.
- It is designed in such a way so that it acts like Unix, but it does not have the original Unix code in it.

- **Linux is just the kernel and not the complete OS**

Ubuntu





Introduction to Ubuntu

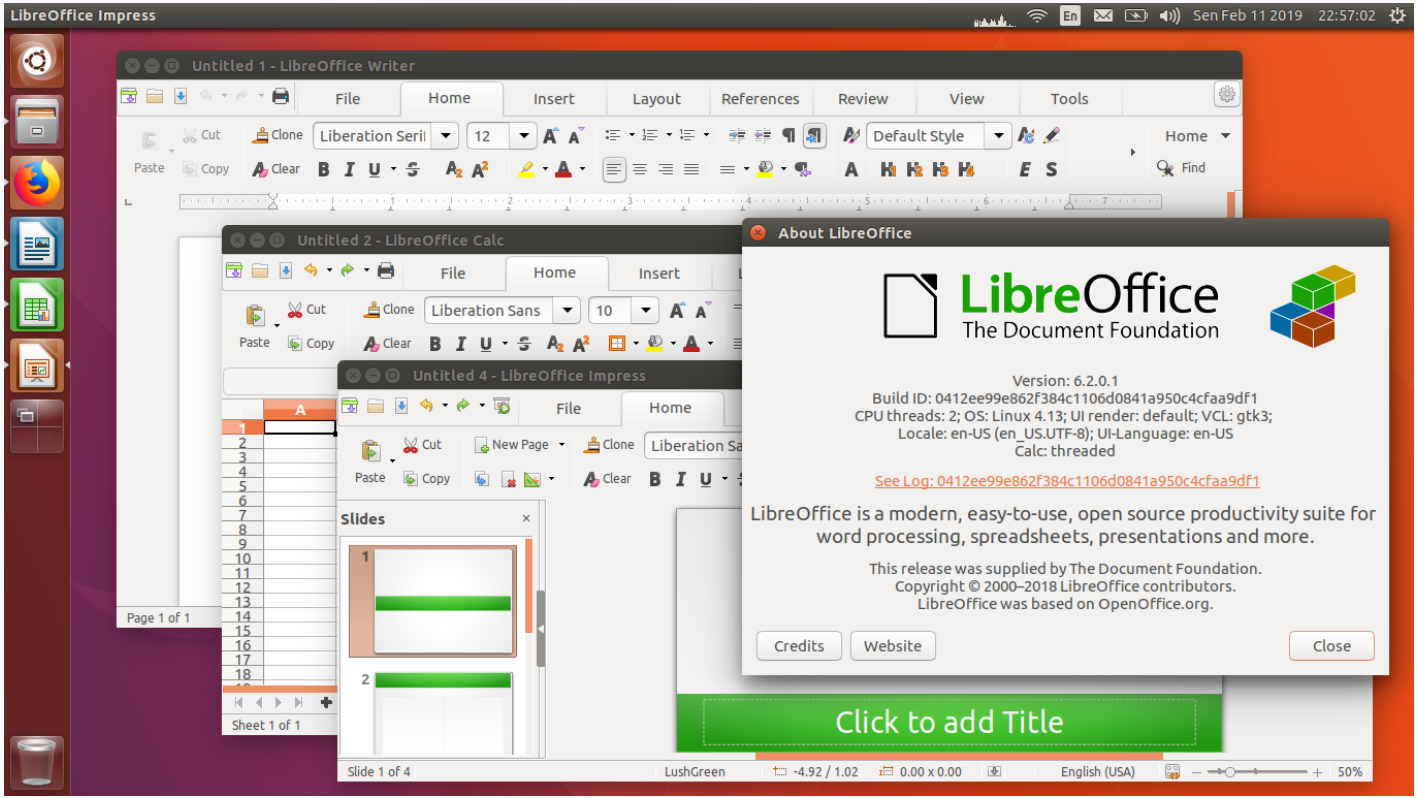
- Ubuntu is a Linux-based OS.
- It is designed for computers, smartphones, and network servers.
- All the principles used to develop the Ubuntu software are based on the principles of OSS development

Features of Ubuntu

- Ubuntu Desktop **supports all the normal software** on Windows: Firefox, Chrome, VLC,…
- It supports the office suite called **LibreOffice**
- Ubuntu has an in-built email software (Thunderbird)
- It is easy to find content on Ubuntu with the smart searching facility

- The **best feature** is, it is a **free OS** and is backed by a huge open source community

Features of Ubuntu



Release Cycle of Ubuntu

- Every year there are 2 releases of Ubuntu, one in **April** and one in **October**, from Canonical

- The version number normally denotes the year in which the software was released
  - Version 22.04, 23.04, 23.10, 24.04…

- Ubuntu official site: https://www.ubuntu.com/

Ubuntu - Flavors

- **Ubuntu Desktop**

  pre-built with software that help the regular users perform usual basic activities

- **Ubuntu Server**
  - The server version is used for hosting applications such as web servers and databases
  - Each server version is supported by Ubuntu for 5 years

- **One major difference** is that the graphical environment used for the Desktop Edition is not installed for the Server

Ubuntu Server 24.04

System requirements

- Ubuntu Server Edition provides a common, minimalist base for a variety of server applications, such as file/print services, web hosting, email hosting, etc.

- The recommended system requirements are:
  - CPU: 1 GHz or better
  - RAM: 1 GB or more
  - Disk: a minimum of 5 GB

Ubuntu - Flavours

- Ubuntu flavours offer a unique way to experience Ubuntu, each with their own choice of default applications and settings.

- Ubuntu flavours are owned and developed by members of our global community and backed by the full Ubuntu archive for packages and updates.

Ubuntu - Flavors

Kubuntu

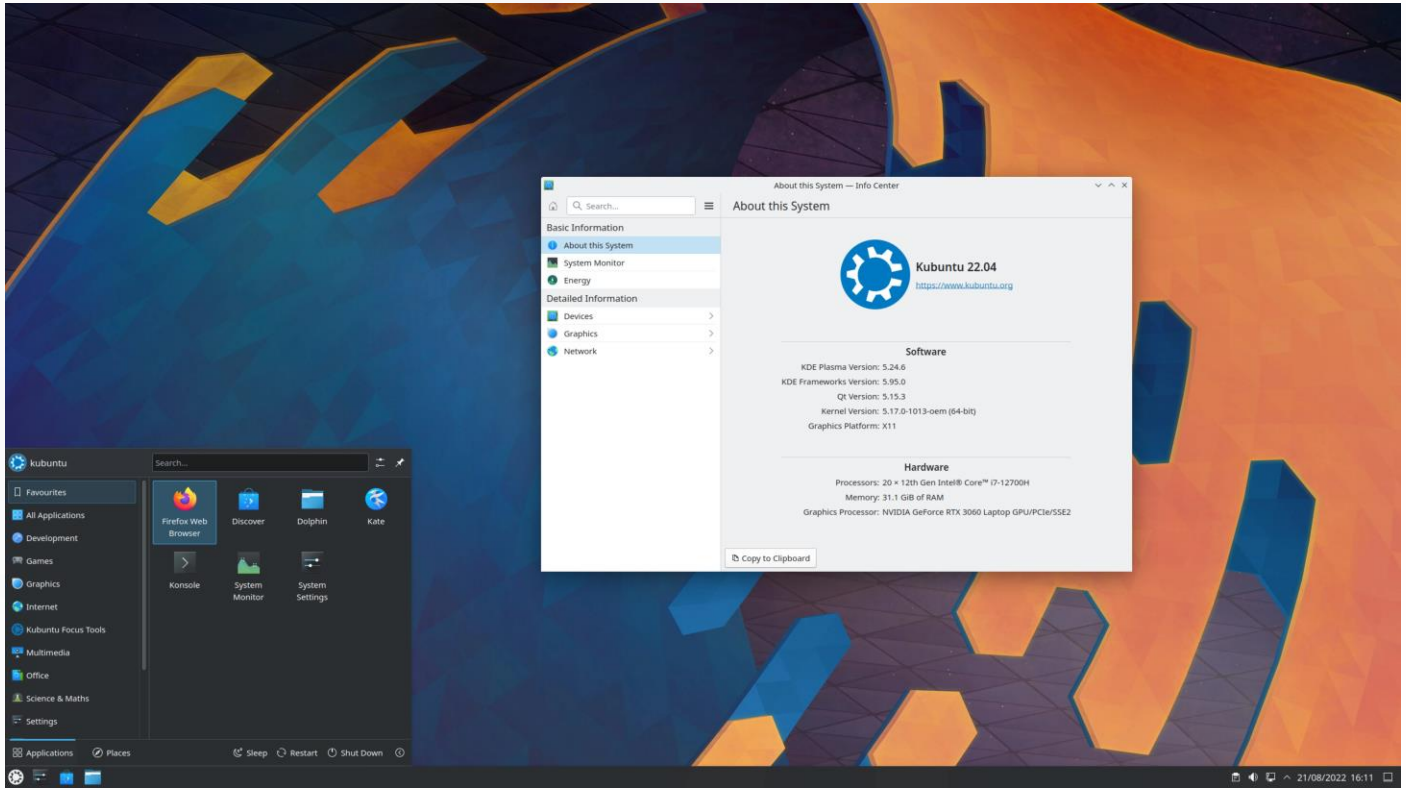
- Kubuntu unites Ubuntu with **KDE and the Plasma desktop**, bringing you a full set of applications including productivity, office, email, graphics, photography, and music applications ready to use at startup with extensive additional software installed from not one, but two desktop package managers.

- Built using the Qt toolkit, Kubuntu is fast, slick and beautiful.

- Kubuntu is mobile-ready, enabling easy integration between your PC desktop and phone or tablet with KDE Connect.

Ubuntu - Flavors

Kubuntu



Ubuntu - Flavors

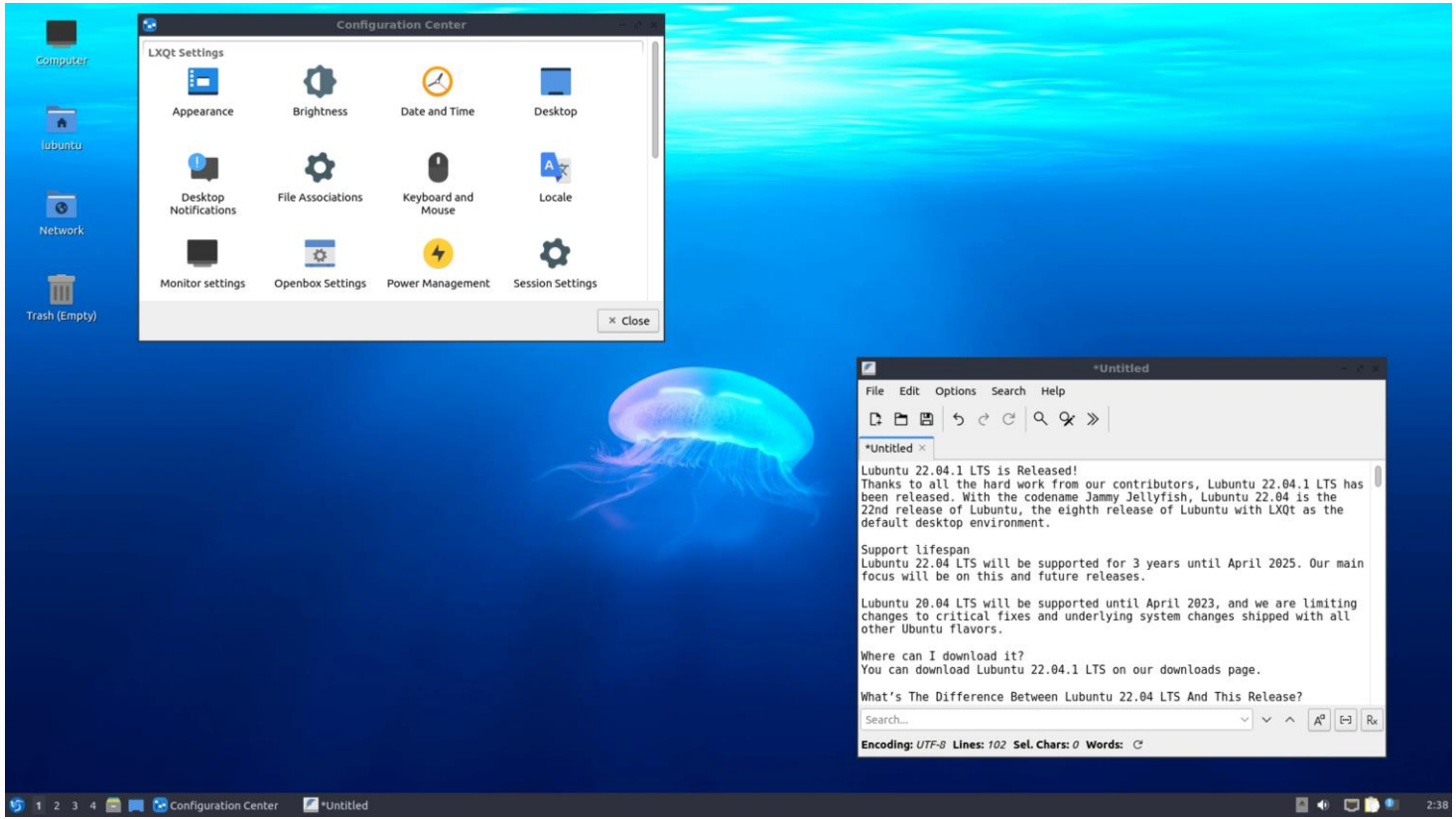
Lubuntu

- Lubuntu is designed to be a simple, easy to use system that is light, fast and modern. Lubuntu provides the **LXQt desktop** environment which is focused on Qt technologies.

- Lubuntu comes with the essential applications and services needed to browse the Internet, chat, play and be productive.

Ubuntu - Flavors

Lubuntu



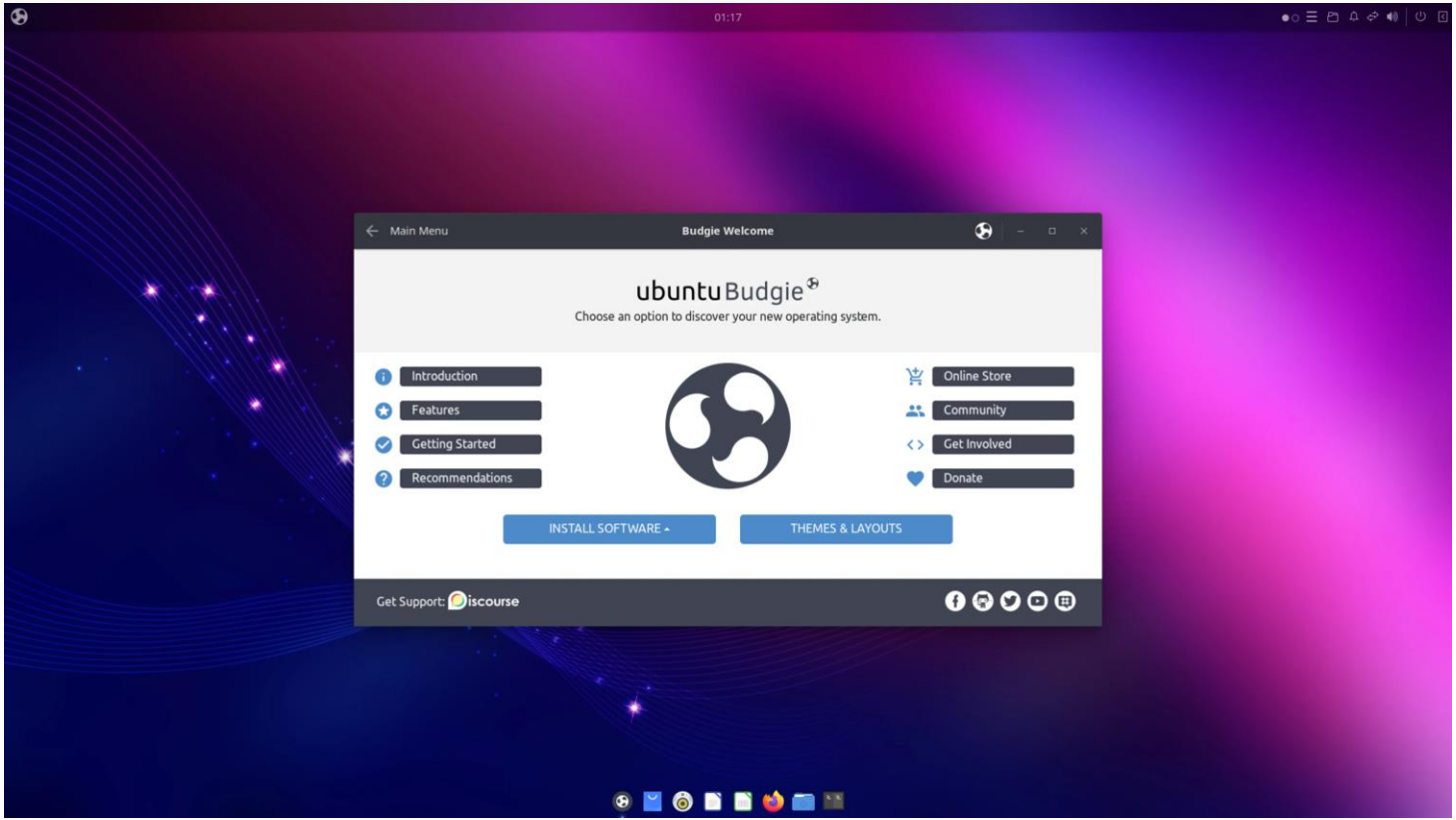
Ubuntu - Flavors

Ubuntu Budgie

- Ubuntu Budgie is a proud official Ubuntu flavour.
- They combine the simplicity and elegance of the **Budgie desktop** environment with the power and familiarity of an Ubuntu based operative system.
- The result is a modern and fast desktop distribution with great defaults, yet fully customizable.

Ubuntu - Flavors

Ubuntu Budgie



Ubuntu - Flavors

Ubuntu MATE

- Ubuntu MATE is a stable, easy-to-use operating system with **a configurable desktop environment**. It is ideal for those who want the most out of their computers and prefer a traditional desktop metaphor.

- With modest hardware requirements it is suitable for modern workstations, single board computers and older hardware alike. Ubuntu MATE makes modern computers fast and old computers usable.

Ubuntu - Flavors

Ubuntu MATE

Ubuntu - Flavors

Ubuntu Studio

- Ubuntu Studio is pre-configured **for content creation** of all kinds.

- Whether you’re an audio engineer, musician, graphic designer, photographer, video producer, or streamer, this is a full-fledged desktop computing system that will fit your needs. If you can dream it, you can create it with Ubuntu Studio.

Ubuntu - Flavors

Ubuntu Studio

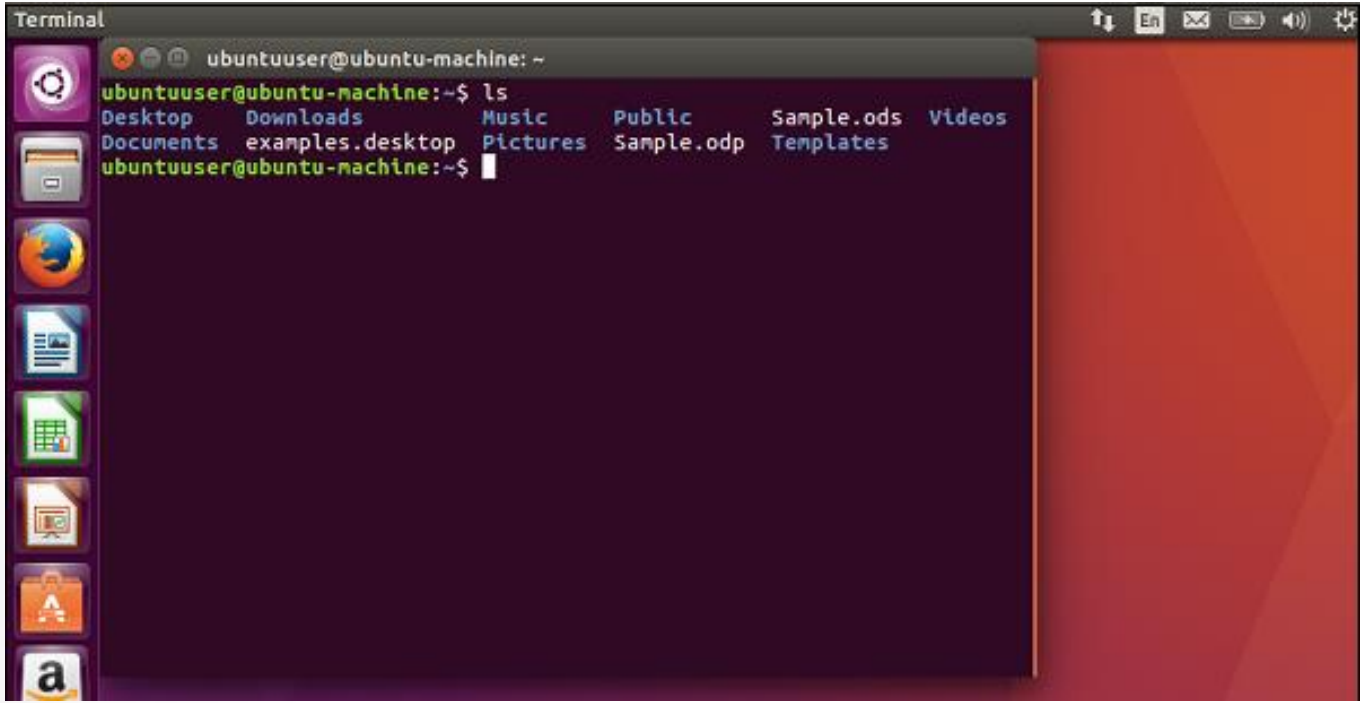
Ubuntu - Flavors

- **Ubuntu Kylin**

  The Ubuntu Kylin project is tuned to the needs of **Chinese users**, providing a thoughtful and elegant experience out-of-the-box.

- **Xubuntu**

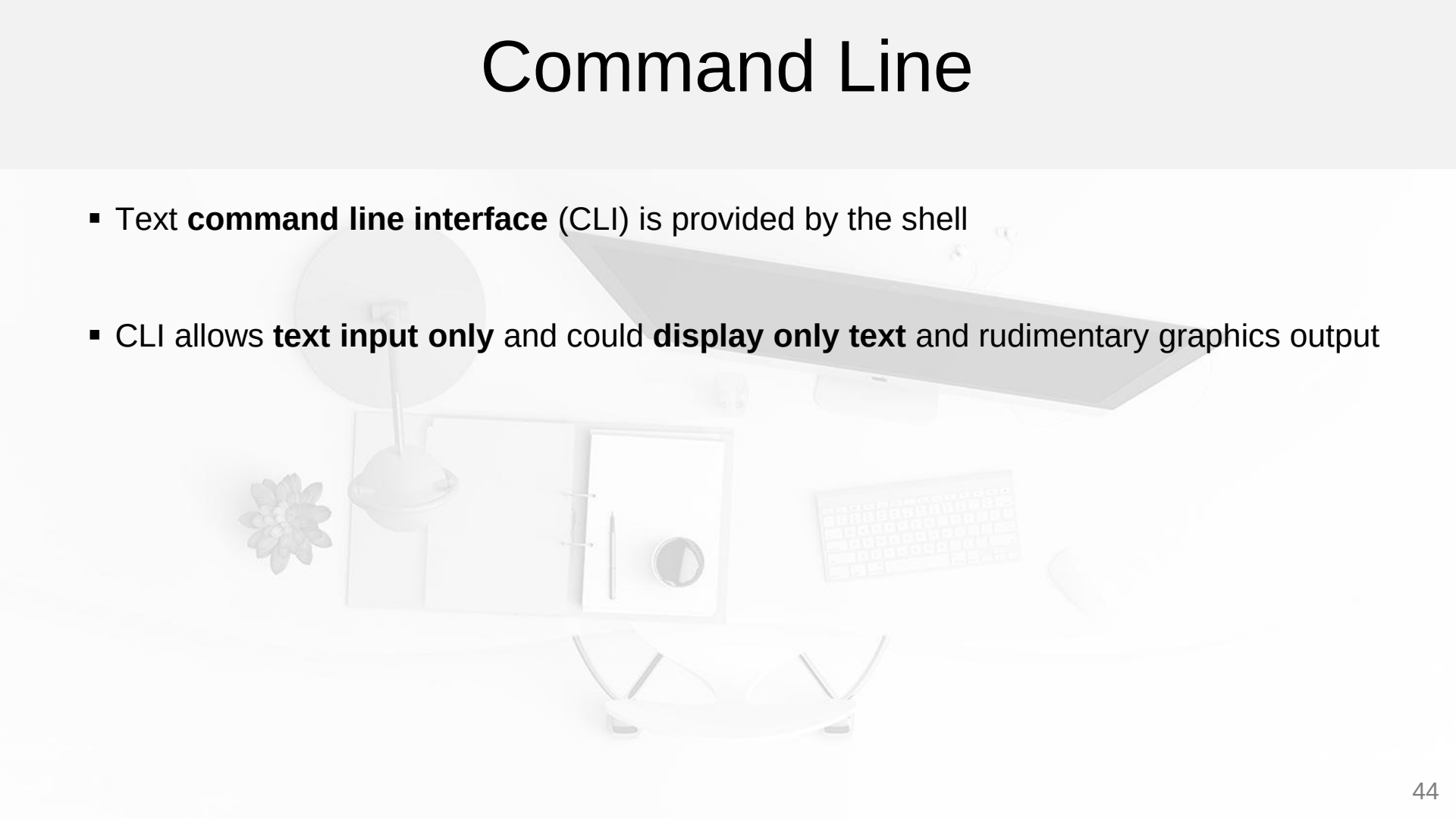
  Xubuntu comes with **Xfce**, which is a stable, light and configurable desktop environment with a lot of consideration for usability.



Command Line

Command Line

- Text **command line interface** (CLI) is provided by the shell
- CLI allows **text input only** and could **display only text** and rudimentary graphics output



Interacting with the bash Manual

- The **man** command provides access to the manual pages stored on the Linux system
- Entering the man command is **followed by a specific command name**
  - to view through the man pages by pressing the **spacebar**, or go line by line using the **Enter key**
  - press the **q** key to **quit**
  - a command has man pages in multiple section content areas
    - ✓**man** section# topic
    - ✓**man** 7 hostname
- Most commands accept the **--help** option

Navigating the Filesystem

Looking at the Linux filesystem

How Linux System references files and directories:

- Does not use drive letters in pathnames
- Linux stores files within a single directory structure, called a **virtual directory**
- The Linux virtual directory structure contains a single base directory, called the **root**
- The first hard drive installed in a Linux system is called the **root drive**
- On the root drive, Linux can use special directories as **mount points**. Mount points are directories in the virtual directory where you can **assign additional storage devices**
- In a Linux filesystem, **common directory names** are used for **common functions**

Navigating the Filesystem

Looking at the Linux filesystem

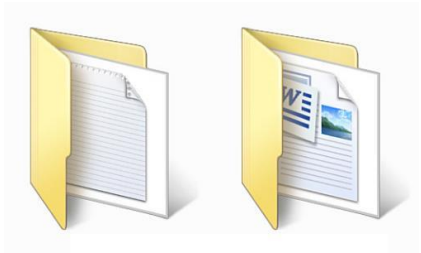
TABLE 3-3 Common Linux Directory Names

Directory	Usage
/	root of the virtual directory, where normally, no files are placed
/bin	binary directory, where many GNU user-level utilities are stored
/boot	boot directory, where boot files are stored
/dev	device directory, where Linux creates device nodes
/etc	system configuration files directory
/home	home directory, where Linux creates user directories
/lib	library directory, where system and application library files are stored
/media	media directory, a common place for mount points used for removable media
/mnt	mount directory, another common place for mount points used for removable media
/opt	optional directory, often used to store third-party software packages and data files

Navigating the Filesystem

Looking at the Linux filesystem

/proc	process directory, where current hardware and process information is stored
/root	root home directory
/sbin	system binary directory, where many GNU admin-level utilities are stored
/run	run directory, where runtime data is held during system operation
/srv	service directory, where local services store their files
/sys	system directory, where system hardware information files are stored
/tmp	temporary directory, where temporary work files can be created and destroyed
/usr	user binary directory, where the bulk of GNU user-level utilities and data files are stored
/var	variable directory, for files that change frequently, such as log files



Directory and File Management

Directory and File Management

- **Print working directory**
  - Displays the shell session's current directory location
  - Syntax: **pwd**

- **List contents**
  - List the files and folders within a directory
  - Syntax: **ls [Options] [directory]**
    - ✓ **-a:** all
    - ✓ **-l:** long list format
  - **Note:** ls -l -a = ls -la

Directory and File Management

- **Change directory**
  - Move the shell session to another directory
  - Syntax: **cd [directory]**
  - E.g.,
    - ✓cd ~: navigate to your home directory
    - ✓cd /home
    - ✓cd ..

Directory and File Management

- **Create directories**
  - Create new empty directories
  - Syntax: **mkdir [Options] folder_name**

    -p: make parent directories as needed
  - E.g.:

    ✓mkdir dir1

    ✓mkdir dir1 dir2 dir3

    ✓mkdir /tmp/tutorial

    ✓mkdir -p /home/giau/test_2/cse454

Directory and File Management

- **Create a File**
  - Syntax 1: **touch file_name**

    E.g.: touch file1
  - Syntax 2: **echo “Contens” > file_name**
    - ✓E.g.: echo “Lower case” > a.txt
    - ✓E.g.: echo “Upper case” >> a.txt
- **Copy files / directories**
  - Syntax: **cp [OPTIONs] Source[s] destination**
  - Option:
    - ✓**-i**: prompt before overwrite
    - ✓**-R or -r**: copy directories recursively

Directory and File Management

- **Create links**
  - A link is a placeholder in a directory that points to the real location of the file
  - Two types:
    - ✓**A symbolic link: ln -s Target [Directory]**

      simply a physical file that points to another file somewhere
    - ✓**A hard link: ln Target [Directory]**

      creates a separate virtual file that contains information about the original file and where to locate it

Directory and File Management

- **Rename or Move a file / directory**
  - Renaming file is called **moving file**
  - The **mv** command is available to move both files and directories to another location or a new name
  - Syntax: **mv [Options] Source[s] Dest**
  - E.g.:
    - ✓Move: mv combined.txt dir1
    - ✓Rename: mv backup_combined.txt combined_backup.txt

Directory and File Management

- **Delete directory**
  - The basic command for removing a directory is **rmdir** (only for removing **empty** directories)
  - Syntax: **rmdir [option] directory**
  - Option:

    -p: remove Directory and its ancestors

Directory and File Management

- **Delete files / directories**
  - Syntax: **rm [options] Dest**
  - The shell has no recycle bin or trashcan
  - Options:
    - ✓If removing lots of files and don't want to be bothered with the prompt, use the **-f** parameter to force the removal
    - ✓**-i:** prompt before every removal
    - ✓**-r, -R**: remove directories and their contents recursively
    - ✓**-d:** remove empty directories

Directory and File Management

- **View the whole file**
  - Syntax: **cat [options] file_name**
    - ✓Option:

      -n: numbers all the lines
    - ✓For large files, the text in the file just quickly scrolls off the display without stopping
  - Syntax: **more [option] file_name**
    - ✓Options:

      -<number>: the number of lines per screenful

      +<number>: display file beginning from line number

Directory and File Management

- **Edit file content**
  - Use Nano: **nano file**
  - Use Vi: **vi file**
  - Use gedit (Desktop): **gedit file**

Nano Editor

- To open an existing file or to create a new file, type:

  nano file_name
- At the bottom of the window, there is a list of the most basic command shortcuts
- All commands are prefixed with either **^** or **M** character
  - The caret symbol (**^**) represents the **Ctrl** key
  - The letter **M** represents the **Alt** key
- For example:
  - Ctrl + o: to save changes
  - Ctrl + x: to exit nano
  - Ctrl + g: to get a list of all commands

Vi Editor

- The **vi** (**vi**sual) editor has **two modes of operation**:
  - **Command mode**: enables you to perform administrative commands on the file
  - **Insert mode:** enables you to insert text into the file
- Switch to the Insert mode from the command mode by pressing '**i**'
- To return to the command mode and save the changes, press the **Esc** key

Vi Editor

Vi basic commands

- i – Insert text before the current cursor location (goes into insert mode)
- a – Insert text after the current cursor location (goes into insert mode)
- ESC – Terminate insert mode
- u – Undo last change
- o – Open a new line (goes into insert mode)
- dd – Delete line
- D – Delete from the cursor position to the end of the current line
- dw – Delete word
- x – Delete character at the cursor
- X – Delete the character before the cursor location

Vi Editor

- k – Move cursor up
- j – Move cursor down
- h – Move cursor left
- l – Move cursor right
- Shift + zz – Save the file and quit
- :w – Save the file but keep it open
- :q! – Quit without saving
- :wq – Save the file and quit
- :x or :exit – Save (but only write if there are changes) and quit

Package Management

Package Management

apt-get command

- APT package handling utility
- Retrieve packages and information about them from authenticated sources for installation, upgrade and removal of packages together with their dependencies
- Syntax: **apt-get command pkg1 [pkg2 ...]**

Package Management

apt-get command

- Most used commands:
  - **update** - Retrieve new lists of packages
  - upgrade - Perform an upgrade
  - **install** - Install new packages
  - remove - Remove packages
  - **purge** - Remove packages and config files
  - autoremove - Remove automatically all unused packages

Package Management

apt-get command

- Most used commands:
  - dist-upgrade - Distribution upgrade
  - clean - Erase downloaded archive files
  - autoclean - Erase old downloaded archive files
  - source - Download source archives
  - download - Download the binary package into the current directory

Basic commands

Basic commands

- **sudo command**
  - Execute a command as another user (default as superuser)
  - Syntax: **sudo command**

- **clear command**
  - Clear the terminal screen
  - Syntax: **clear**

Basic commands

- **Output redirection**
  - The most basic type of redirection is sending output from a command to a file

    command > outputfile
  - Use >> to append output from a command to an existing file

    data >> outputfile

- **Input redirection**

  Takes the content of a file and redirects it to a command
  - **command < inputfile**
  - E.g.: wc < test6

Basic commands

- **Pipes**
  - Send the output of one command to the input of another command

    ✓ls > list.txt

    ✓sort < list.txt
  - The pipe is put between the commands to redirect the output from one to the other

    ✓**command1 | command2**

    ✓E.g.: ls | sort
  - There’s no limit to the number of pipes you can use in a command

    E.g.: **ls | sort | more**

Basic commands

- **ping command**
  - Send ICMP ECHO_REQUEST to a network host
  - Syntax: **ping [Options] dest**
  - Options:
    - ✓-c count: stop after sending count ECHO_REQUEST packets
    - ✓-i <interval>: seconds between sending each packet
    - ✓-s <size>: use <size> as number of data bytes to be sent
  - E.g.: ping google.com

- Use **ifconfig –a** to display all interfaces which are currently available, even if down

Basic commands

- **grep command**
  - Search for PATTERN in each FILE
  - Syntax: **grep [OPTIONs] PATTERN [FILEs]**
  - Options:
    - ✓**-i:** ignore case distinctions
    - ✓**-w:** force PATTERN to match only whole words
    - ✓**-x:** force PATTERN to match only whole lines
    - ✓**-n:** print line number with output lines
  - E.g.: grep -i ‘hello world’ menu.h main.c

Basic commands

- **wget command**
  - Download a file from the web (HTTP, HTTPS, FTP)
  - Syntax: **wget [options] [url]**
  - Options:

    ✓**-O new_name:** save the downloaded file under a different name

    ✓**-P directory:** save the downloaded file to specific directory
  - E.g.:

    wget https://cdn.kernel.org/kernel/v4.x/linux-4.17.2.tar.xz

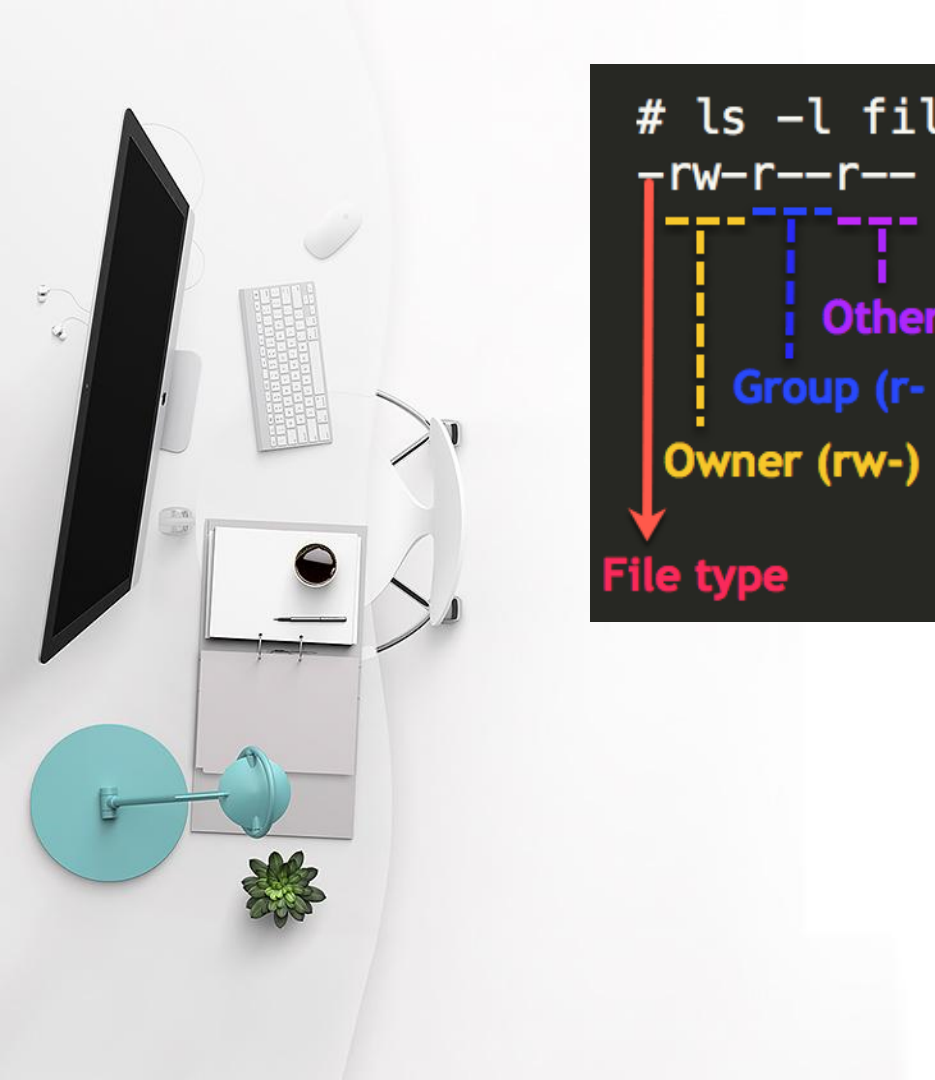
Compression and Decompression

Compression and Decompression

- **tar command**
  - **Compress** many **files or directories** together into a .tar/.tar.gz/.tar.bz2 archive, and extract the archive
  - Syntax: **tar [options] [files]**
  - Options:
    - ✓ **-x:** extract files from an archive
    - ✓ **-v:** verbosely list files processed
    - ✓ **-f:** use archive file
    - ✓ -**z:** filter the archive through gzip
    - ✓ **-j:** filter the archive through bzip2
    - ✓ **-c:** create a new archive
    - ✓ -p: preserve file permissions
  - E.g.:
    - ✓ tar xf archive.tar = tar –xf archive.tar: extract all files from archive.tar
    - ✓ tar cf archive.tar file_1 file_2: create archive.tar form files

Compression and Decompression

- **gzip command**
  - **Compress file** into a gz file, and uncompress it
  - Syntax: **gzip [options] [Files]**
    - ✓Compress: gzip –k FileName
    - ✓Decompress: gzip –d file.gz = gunzip file.gz
- **bzip command**
  - **Compress file** into a bz2 file, and uncompress it
  - Syntax: **bzip [options] [Files]**
    - ✓Compress: bzip2 –k FileName
    - ✓Decompress: bzip2 –d file.bz2 = bunzip2 file.bz2



```
# ls -l file
-rw-r--r-- 1 root root 0 Nov 19 23:49 file
```

Other (r - -)

Group (r- -)

Owner (rw-)

File type

r = Readable
w = Writeable
x = Executable
- = Denied

Ownership

Ownership

- **chown command**
  - Change the owner and/or group of each FILE to OWNER and/or GROUP
  - Syntax: **chown [OPTIONs] [OWNER][:[GROUP]] FILEs**
  - Option:

    -R: operate on files and directories recursively
  - E.g.:

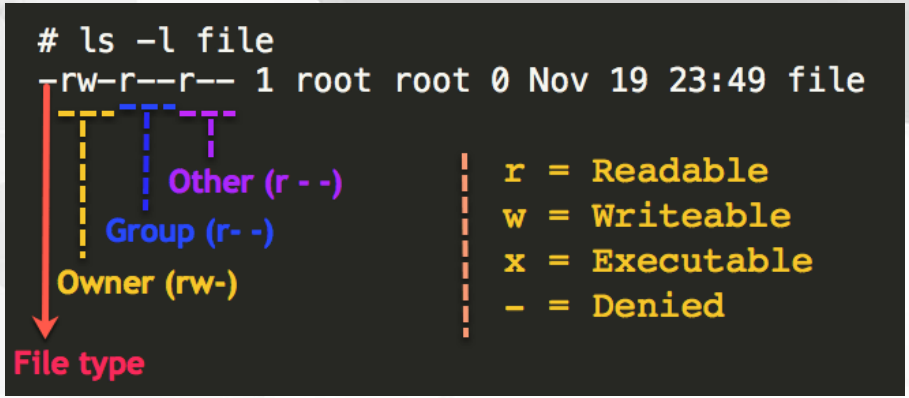
    ✓chown www-data:www-data /var/www/index.html

    ✓chown –R www-data:www-data /var/www/

File / Directory Permission

- View the permissions of files and directories:

ls –l or **ls –n**



- File type: a file (_) or a directory (d)
- The next fields represent the permission groups: **owner**, **group**, and **other**

File / Directory Permission

- **Permission Groups**

Permission	Description
Owner	Permissions used by the assigned owner of the file or directory
Group	Permissions used by members of the group that owns the file or directory
Other	Permissions used by all users other than the file owner, and members of the group that owns the file or the directory

File / Directory Permission

- **Permission Set**
  - **Each permission group has 3 permissions**, called a permission set
  - Each set consists of **read**, **write**, and **execute** permissions
  - **Each file or directory has 3 permission sets** for the three types of permission groups
  - The first permission set represents the **owner** permissions, the second set represents the **group** permissions, and the last set represents the **other** permissions

File / Directory Permission

- **Permission Set**
  - The **read, write, and execute** permissions are represented by the characters **r, w, and x**, respectively
  - The presence of any of these characters, such as r, indicates that the particular permission is granted.
  - A dash (–) symbol in place of a character in a permission set indicates that a particular permission is denied
  - Linux **assigns initial permissions automatically** when a new file or directory is created

File / Directory Permission

- **Permission Set**
  - Permissions can also be represented numerically:
    - ✓**r = 4**
    - ✓**w = 2**
    - ✓**x = 1**
    - ✓E.g.:
      - -(rw-)(rw-)(r--)
      - -(42-)(42-)(4--)
      - 664

File / Directory Permission

- **chmod command**
  - To alter the permissions of a file
  - Syntax:
    - ✓**chmod [Options] Mode[,Mode…] File/Dir**
    - ✓**chmod [Options] Octal-Mode File/Dir**
  - Option:

    -R: change files and directories recursively
  - E.g.,
    - ✓ chmod u+x file_name
    - ✓ chmod g+rw file_name
    - ✓ chmod o-w file_name
    - ✓ chmod u=rwx,g=rx,o= file_name
    - ✓ chmod 774 file_name

Basic Script Building

Using Multiple Commands

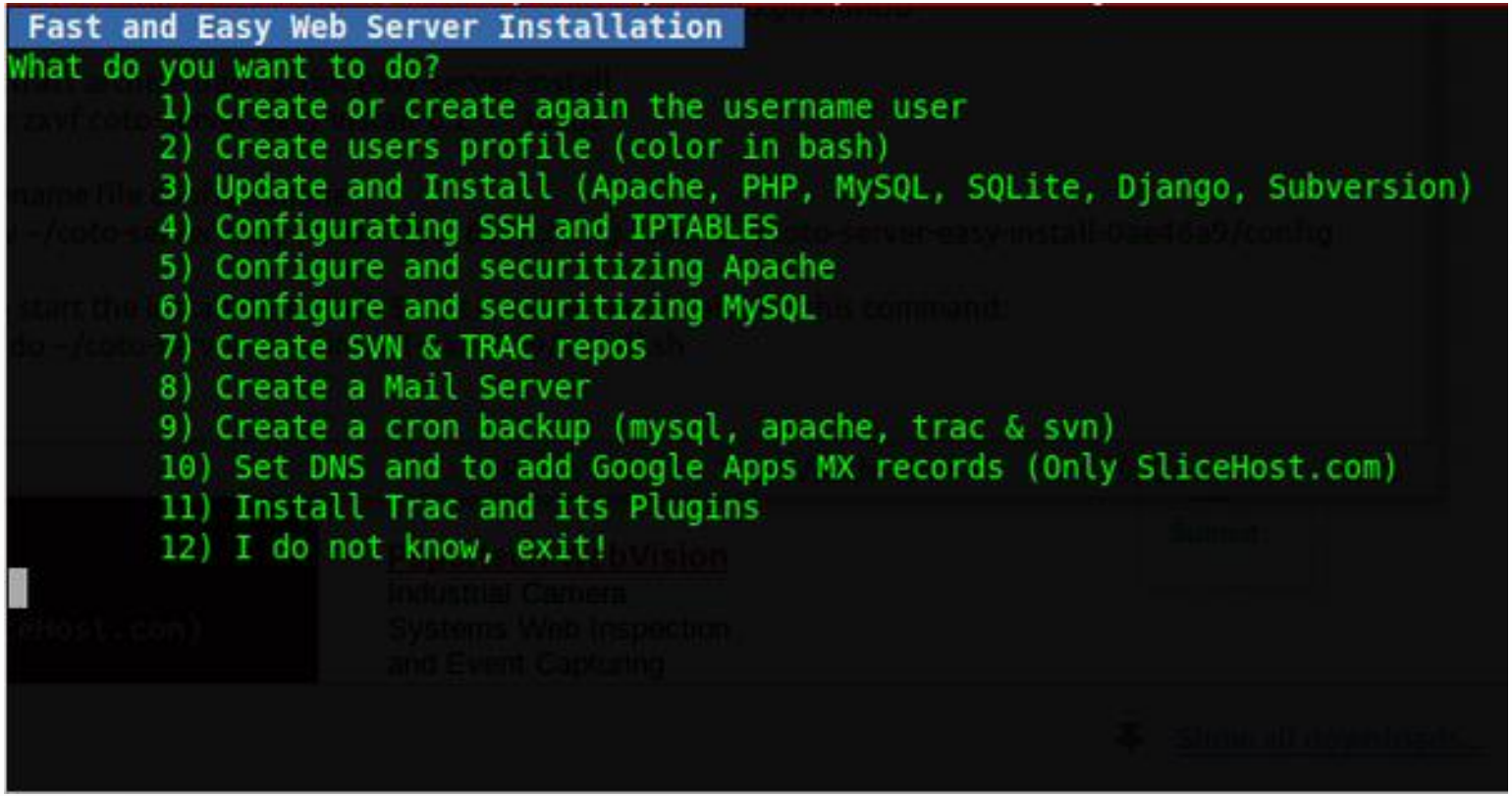
- The key to shell scripts is the **ability to enter multiple commands** and process the results from each command, even possibly passing the results of one command to another

- The shell allows you to chain commands together into a single step

- To **run two commands together**, enter them on the same prompt line, separated with a semicolon

date ; who → shell script

Creating a Script File



Creating a Script File

- Use a text editor to **create a file** and then **enter the commands**
- When creating a shell script file, **must specify the shell** you are using in the first line

  #!/bin/bash
- After indicating the shell, commands are entered onto each line of the file

  can use the semicolon (;) and put both commands on the same line
- A line starts with the # symbol aren't interpreted → used to leave **comments**

Creating a Script File

- **Save this script** in a file
- **Give the file owner permission to execute** the file, using the **chmod** command

  chmod +x file_name
- **Run the script,** using an absolute or relative file path to reference our shell script file in the prompt

  ./file_name

Creating a Script File

Write the first script

- Script name: test.sh
- Script code:

```
#!/bin/bash
# This is the first script and the first comment
pwd
ls
```


Q&A

References

- **Unix Vs Linux: What is Difference Between UNIX and Linux**

  https://www.softwaretestinghelp.com/unix-vs-linux/amp/
- **The Linux command line for beginner**

  https://ubuntu.com/tutorials/command-line-for-beginners#6-a-bit-of-plumbing


Read More

Displaying Messages

- The echo command can display a simple text string
- To delineate text strings:
  - By default, you don’t need to use quotes
  - Can uses either double or single quotes
- Use the **-n parameter** to echo a text string on **the same line** as a command output
- Example:
  - echo This is a test
  - echo “This is a test to see if you're paying attention”
  - echo ‘Rich says "scripting is easy".’
  - echo -n “The time and date are: “

User Input

- The read command is used to ask the user for input. This command takes the input and will save it into a variable.
- Syntax: **read [options] [variable_name(s)]**
- Options:
  - -p: specify a prompt without line break
  - -s: hide the input from terminal
- E.g.:
  - read var_name
  - read –p ‘Username: ’ username
  - read –sp ‘Password: ’ password

Using Variables

Variables allow you to **temporarily store information within the shell script** for use with other commands in the script

- **Environment variables**
  - **Set** command to display a complete list of active environment variables available

```
$ set
BASH=/bin/bash
[...]
HOME=/home/Samantha
HOSTNAME=localhost.localdomain
HOSTTYPE=i386
IFS=$' \t\n'
IMSETTINGS_INTEGRATE_DESKTOP=yes
IMSETTINGS_MODULE=none
LANG=en_US.utf8
LESSOPEN='|/usr/bin/lesspipe.sh %s'
LINES=24
LOGNAME=Samantha
[...]
```

Using Variables

- **Environment variables**
  - Use them within scripts by using the environment variable's name preceded by **$**

```
$ cat test2
#!/bin/bash
# display user information from the system.
echo "User info for userid: $USER"
echo UID: $UID
echo HOME: $HOME
$
```

  - To display an actual $, must precede it with a backslash character (\)

    echo "The cost of the item is \$15"

Using Variables

- **User variables**
  - A shell script allows you to set and use your own variables within the script
  - Any text string of up to 20 letters, digits, or an underscore character and case sensitive
  - Values are **assigned** to user variables using an **equal sign**. **No spaces** can appear between the variable, the equal sign, and the value

    var1=10
    var2=-57
    var3=testing
    var4="still more testing"

  - User variables can be referenced using **$** or **${**variable**}**

Using Variables

- **User variables**

```
$ cat test3
#!/bin/bash
# testing variables
days=10
guest="Katie"
echo "$guest checked in $days days ago"
days=5
guest="Jessica"
echo "$guest checked in $days days ago"
$
```

Using Variables

- **Command substitution**
  - The ability to extract information from the output of a command and assign it to a variable
  - There are **2 ways to assign the output of a command to a variable**:
    - ✓The backtick character **(`)**: test=`date`
    - ✓The **$()** format: test=$(date)

Using Variables

- **Special variables**
  - $0: The filename of the current script
  - $n: These variables correspond to the arguments with which a script was invoked
  - $#: The number of arguments supplied to a script
  - $@: The variable includes all the command line parameters within a single variable

Working with the if-then Statement

The most basic type of structured command is the if-then statement

```
if command
then
     commands
fi
```

```
$ cat test1.sh
#!/bin/bash
# testing the if statement
if pwd
then
     echo "It worked"
fi
$
```

Exploring the if-then-else Statement

The if-then-else statement provides another group of commands in the statement

```
if command
then
    commands
else
    commands
fi
```

```
#!/bin/bash
# testing the else section
#
testuser=NoSuchUser
#
if grep $testuser /etc/passwd
then
   echo "The bash files for user $testuser are:"
   ls -a /home/$testuser/.b*
   echo
else
   echo "The user $testuser does not exist on this system."
   echo
fi
```

Nesting ifs

```
if command1
then
   commands
elif command2
then
    more commands
fi
```

```
$ cat test5.sh
#!/bin/bash
# Testing nested ifs - use elif
#
testuser=NoSuchUser
#
if grep $testuser /etc/passwd
then
   echo "The user $testuser exists on this system."
#
elif ls -d /home/$testuser
then
   echo "The user $testuser does not exist on this system."
   echo "However, $testuser has a directory."
#
```

Testing condition

- To test a condition, use square brackets:

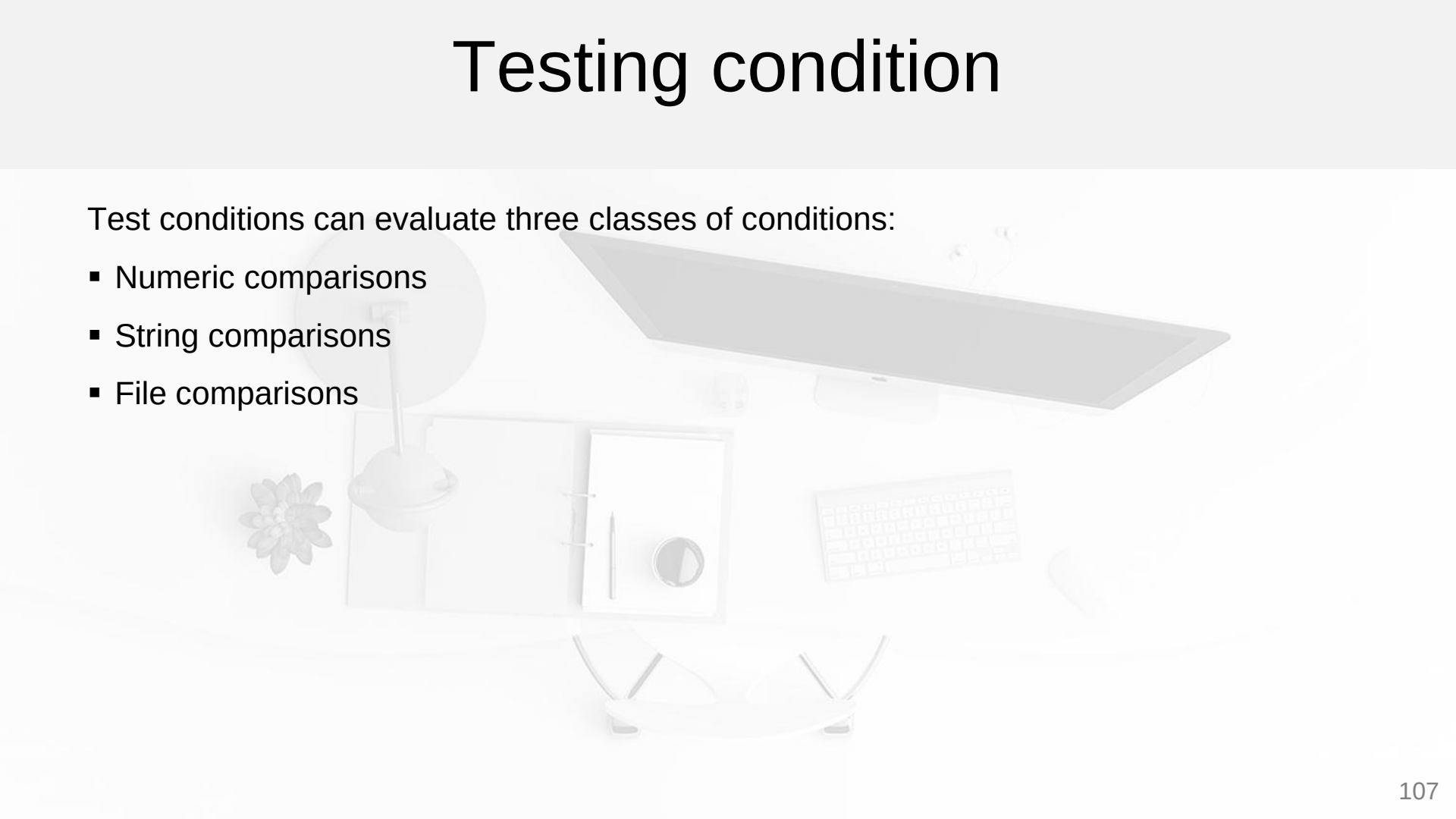
```
if [ condition ]
then
    commands
fi
```

- **Must have a space** after the first bracket and a space before the last bracket

Testing condition

Test conditions can evaluate three classes of conditions:

- Numeric comparisons
- String comparisons
- File comparisons



Testing condition

Using numeric comparisons

Comparison	Description
`n1 -eq n2`	Checks if `n1` is equal to `n2`
`n1 -ge n2`	Checks if `n1` is greater than or equal to `n2`
`n1 -gt n2`	Checks if `n1` is greater than `n2`
`n1 -le n2`	Checks if `n1` is less than or equal to `n2`
`n1 -lt n2`	Checks if `n1` is less than `n2`
`n1 -ne n2`	Checks if `n1` is not equal to `n2`

Testing condition

Using numeric comparisons

Example:

```
$ cat numeric_test.sh
#!/bin/bash
# Using numeric test evaluations
#
value1=10
value2=11
#
if [ $value1 -gt 5 ]
then
    echo "The test value $value1 is greater than 5"
fi
#
if [ $value1 -eq $value2 ]
then
    echo "The values are equal"
else
    echo "The values are different"
fi
```

Testing condition

Using string comparisons

Comparison	Description
`str1 = str2`	Checks if `str1` is the same as string `str2`
`str1 != str2`	Checks if `str1` is not the same as `str2`
`str1 < str2`	Checks if `str1` is less than `str2`
`str1 > str2`	Checks if `str1` is greater than `str2`
`-n str1`	Checks if `str1` has a length greater than zero
`-z str1`	Checks if `str1` has a length of zero

Testing condition

Using string comparisons

Example 1:

```
$ cat test7.sh
#!/bin/bash
# testing string equality
testuser=rich
#
if [ $USER = $testuser ]
then
   echo "Welcome $testuser"
fi
$
$ ./test7.sh
Welcome rich
$
```

Testing condition

Using string comparisons

Example 2:

```
$ cat test9b.sh
#!/bin/bash
# testing string sort order
val1=Testing
val2=testing
#
if [ $val1 \> $val2 ]
then
   echo "$val1 is greater than $val2"
else
   echo "$val1 is less than $val2"
fi
$
$ ./test9b.sh
Testing is less than testing
$
$ sort testfile
testing
Testing
```

Testing condition

Using file comparisons

Comparison	Description
`-d file`	Checks if `file` exists and is a directory
`-e file`	Checks if `file` exists
`-f file`	Checks if `file` exists and is a file

Testing condition

Using file comparisons

Comparison	Description
`-r file`	Checks if `file` exists and is readable
`-s file`	Checks if `file` exists and is not empty
`-w file`	Checks if `file` exists and is writable
`-x file`	Checks if `file` exists and is executable
`-O file`	Checks if `file` exists and is owned by the current user
`-G file`	Checks if `file` exists and the default group is the same as the current user
`file1 -nt file2`	Checks if `file1` is newer than `file2`
`file1 -ot file2`	Checks if `file1` is older than `file2`

Testing condition

Using file comparisons

Example:

```
$ cat test11.sh
#!/bin/bash
# Look before you leap
#
jump_directory=/home/arthur
#
if [ -d $jump_directory ]
then
   echo "The $jump_directory directory exists"
   cd $jump_directory
   ls
else
   echo "The $jump_directory directory does not exist"
fi
#
$
$ ./test11.sh
The /home/arthur directory does not exist
$
```

Considering Compound Testing

The if-then statement allows you to use Boolean logic to combine tests.

You can use these two Boolean operators:

```
[ condition1 ] && [ condition2 ]
[ condition1 ] || [ condition2 ]
```