Types of OS ?
----

1) Single User Single Tasking	(DOS)
2) Single User Multi tasking	(Windows)
3) Multiuser Multitasking	(Linux/Unix)
4) Real Time	(FlightRadar)
5) Embedded OS	(Robotics)

Linux: kernel

- Open Source

    * Free to study
    * Free to modify
    * Free to dist.

```
# which ls
/usr/bin/ls

# cat /usr/bin/ls
```
___ How to read the source code of a binary(command) ?

++

```
USER ----[kernel]i/p----> SHELL -----[kernel]i/p----> H/W

USER <----o/p[kernel]---- SHELL <-----o/p[kernel]---- H/W
```

SHELL: Shell is the platform which is used by the user to provide instructions to the h/w via kernel.

History of SHELL:
----

sh	[shell]	: Its a traditional shell of UNIX.
ksh	[Korn Shell]	: UNIX Script
csh	[C lang shell]	
tcsh	[Turbo C shell]	
bash	[ksh + tcsh]	

GUI
- GNOME (Basic)
- KDE (Many Other Utilities)

CLI
  _ TUI [TEXT MODE USER INTERFACE]

Unique Identity: Inode Number/Index number

Installtion:

Recommanded space: 20G

Partitioning:

/	:	[Parent Partition]
/boot	:	[Booting configuration files: GRUB]
SWAP	:	[RAM x 2]: 4GB x 2 = 8GB
		Phy Mem. Virtual Mem.

Virtual Mem: A part of HDD act as RAM.

/dev/sda
/dev/sda1
/dev/sda2

storage device 'a'

File System:

ext2, ext3, ext4, xfs, vfat

---

kiosk@kiosk-virtual-machine:~$

kiosk: username

kiosk-virtual-machine: hostname

~ [tilde] : home dir of logged in user

/home/user1
/home/natasha

/root

$: Normal User

# : Root user

/bin	[binary]
/sbin	[Super Binary]
/usr	[All system commands] /usr/bin, /usr/sbin/
/boot	[Booting Configuration] GRUB

/dev	[Devices]
/lib	[Library]
/lib64	[Library]

/mnt,/misc,/opt,/media : EMPTY

/home	[Home Dir of normal users] /home/natasha ; /home/harry
/root	[Home Dir of Super User]
/proc	[Process & hardware related information]

/selinux[RHEL/CentOS: Security Enhanced Linux] File Based Security [rwx]

/etc	[Important: system services & system related config]
/srv	[Service: Third party services]
/sys	[System: system driver database]
/tmp	[Temp]
/var	[variable data: spool dir(mail inbox), logs]
files	[static/dynamic]

Basic Commands:

PATH:
~~~

a) Absolute Path [/home/natasha/Desktop]
b) Relative Path [cd Dir]

1) pwd	:	Print/Present Working Dir
2) whoami	:	Print the loggedin username
3) date	:	Show the date & time
4) ls	:	Show the list of dir contents
5) mkdir dir_name	:	Make dir
6) touch filename	:	To create a blank file

7) cat > filename: Create a new file with text.
text

[ctrl+d : exit]: to save

>	stdout	[Standard output to the program]
<	stdin	[Standard input to the program]

```
# cat > secret
redhat
redhat
```

8) cat filename: To show the text of file

10) cat >> filename : To append the data in existing file.
text
[ctrl+d]: To save
~~~

11) remove:
~~~

a) A file:
`````````

```
# rm filename
```

b) A dir:
````````

```
# rm -rfv dir_name
```

-r [ Recursive ]
-f [ Forcefully ]
-v [ verbose ] : to view the process in detail.

12) MOVE:

```
# mv sourceFile/Dir DestinationDir
```

13) COPY:
~~~

a) File to File:
----------------

```
# cp srcFile dstFile

# cp -i srcFile dstFile
```

b) File/Dir to Dir:
-------------------

```
# cp -rfv srcFile/Dir dstDir
```

```
$ ls

$ ls -l [ List of dir content with property ]

drwxrwxr-x 3 kiosk kiosk       52 Sep  3 21:56 demo
|
|
\_> File type
```

| | |
|---|---|
| Regular files | [-] |
| Dir | [d] |
| block | [b] |
| char | [c] |
| pipe OR socket | [p OR s] |
| links | [l] |

Find data in linux:
~~~

syntax:
````````

find <where_to_find> -<attrib> <what_to_find>

1) NAME:
`````````

```
$ find /home/user -name dheeraj
$ find /home/user -iname dheeraj
```

2) Size:
````````

```
$ find /home/user -size 2k
                        2M
                        2G
                        +2M
                        -2M
$ find /home/user -size +2M -size -5M
$ find /home/user -size 5M -iname dheeraj
```

3) Type:
``````````

```
$ find /home/user -type d
$ find /home/user -type d -iname dheeraj
```

4) inum:
````````

```
$ find /home/user -inum 37226204
```

5) User owner:
``````````````

```
$ find /home -user kiosk
```

BACKUP & RESTORE:
````````````````

tarball:
````````````.tar
create:
```````

```
$ tar -cvf backup.tar /home
```

-c : create
-v : verbose
-f : forcefully

2) View:
````````
~~~

~~~
$ tar -tvf backup.tar

-t : Tree view

3) Extract:
```````````
$ tar -xvf backup.tar

-x : extarct

$ mkdir data
$ tar -xvf backup.tar -C data/

-C : create/extract on specific location
~~~

~~~
GUNZIP:
```````
$ gzip backup.tar
$ ls
backup.tar.gz

$ gunzip backup.tar.gz
$ ls
backup.tar
TARBALL + GUNZIP:
```````````````````[.tgz]

1) Create
``````````
$ tar -czvf backup.tgz /home

2) View:
`````````
$ tar -tzvf backup.tgz

3) Extarct:
```````````
$ tar -xzvf backup.tgz
$ tar -xzvf backup.tgz -C data/
~~~

Assignment: https://tmpfiles.org/dl/12280017/carbon20.png
~~~

Ownerships:

```
# ls -l
-rw-r--r-- 1 root root    1985 Jul  6 22:48 health.txt
```

a) user owner
`````````````
```
# chown username file/dir
```

b) group owner
``````````````
```
# chgrp groupname file/dir
```

Q. Create a dir and creates 10000 files under it, assign permissions to all files as user owner can read, edit and execute, group owner can read & execute, and read permission to others.

Q. Set a permission in a way that a normal user create files with default permission 644 and directory with default permission 755.

Q. Write a command that displays permission of a file in numeric format

---

ACL:

Set ACL:

```
# setfacl -m u:harry:rw- file/dir
# setfacl -m g:dbda:rwx file/dir
```

-m [modify]

View ACL:

```
# getfacl file/dir
```
~~~

Q. Create the file "/tmp/acl_file".
- Allow "larry" and "curly" to rwx the file.
- Don’t allow "moe" to access the file (rwx).
- All members of group "stooges" should be able to access the file (rw).

```
# setfacl -m u:larry:7 /tmp/acl_file
# setfacl -m u:curly:7 /tmp/acl_file
# setfacl -m u:moe:0 /tmp/acl_file
# setfacl -m g:stooges:6 /tmp/acl_file
```

Basic of I/O system with mount and unmount
Commands like telnet, ftp, ssh, and sftp

VIM editor:
----
1) Command Line Mode
2) Insert mode
3) Last Line Mode (search, save, exit, replace ...)

i: insert mode
Esc: go back to command mode
yy (Yanked) : Copy the current line.
p: paste
nyy (n=1,2,3,4...n)
dd: cut or delete the current line.
ndd (n=1,2,3,4...n)

:w [write the file (save)]
:q [quit]
:wq [save & quit]
:wq! [save & quite forcefully]
:set nu [show number of lines]
:set nonu
:line-number

/word : to search
:%s/old-word/new-word [replace]

Basic Linux:[cert]

----
https://training.linuxfoundation.org/training/introduction-to-linux/

https://kodekloud.com/courses/the-linux-basics-course/

++
+++++++++++++++++++++++++

Basic Shell Scripting:
~~~

https://www.shellscript.sh/
https://guide.bash.academy/

```
#!/bin/bash
function help(){
        echo "SYNTAX: bash script.sh -d DOMAIN"
    echo "-d DOMAIN : Provide a domain"
    echo "-h/--help : Check usage/help"
}
function run(){
while read sub;do
        if host "$sub.$domain" &> /dev/null;then
                echo "$sub.$domain"
        fi
done < $wordlist
}
for i in {1..2};do
case $1 in
    "-d")
        domain=$2
        shift 2
        ;;
    "-w")
        wordlist=$2
        shift 2
        ;;
    "-h"|"--help")
            help
            exit 0
            ;;
    *)
            echo "Error: $1 wrong argument, use -h/--help"
            exit 127
            ;;
esac
done
run
```
~~~

+++

Process Management:
~~~

Process:
````````

- Any running program is a process
- Multiple instances of the same program are processes.
- Shell is also a process

Process ID (PID)
````````````````

- Each linux process is identified by unique id
- Every process has a Parent Process ID (PPID)
    + Except "init"

- When a process is loaded into memory there is structure:
    + stack
    + heap
    + data segment
    + code segment

[+] Stack: Used for static memory allocation.

[+] Heap: Used for dynamic memory allocation.

[+] Data: Stores any static or global variables if defined.

[+] Code: Is the instructions of the program.
~~~

Create a Process:
~~~

Terminal#1
``````````

```
# sleep 100
```

Terminal#2
``````````

```
# ps -la                [ Show the processes with PID & PPID ]
```

Process States:
``````````````

- The process is built and executing , so now...
- We enter the Process State Machine

- States of Process:

| | |
|---|---|
| [N] New | When a new process is being created |
| [R] Running | Instructions are being executed |
| [W] Waiting | The process is waiting for some event to occur |
| [R] Ready | The process is waiting to be assigned to a processor |
| [T] Terminated | The processes has finished execution and is exiting |

Managing Processes:
~~~

- Linux kernel tracks what each process is doing
- Process is assigned a priority
- Address space assigned to the process
- Files is the process allowed to access
- Is the process a ?:-
    + Parent
    + Child
    + Zombie

=> Sometimes the parent dies first
``````````````````````````````````
``````````````````````````````````

- If the parent just exists or dies, the child process is left running
- The child's PPID is no longer valid due to parent is gone.
- Linux calls these children processes "Zombies"
- In Linux a zombie is just a process in which the children are adopted by the "init" process
- The init process will eventually cleanup the zombie childrens

```
PPID[kill]
        \___________PID[kill]

PPID[die]
        \___________PID[zombie]
```

Process Termination:
~~~

- When a parent forks a child, they can finish in any order( parent first or child first)
- Sometimes the parent process could encounter and error and die
- Sometimes the parent process will just wait around until the child processes all complete before exiting
  + it calls a wait() command
- There are variety of wait command.

==================================================================================

process manager:
```````````````

ps, top, htop
`````````````

```
# top
```

press 'q' to exit

```
# sleep 10 &

# ps          [ PID - Process ID]
# ps -l   [ Show the PPID - Parent PID ]
# ps -e       [ All the processes ]

# ps -la
```
~~~

Kill:
-----
```
# kill PID
OR
# kill -9 PID
```

-9 : Forcefully

```
# pkill sleep

# killall sleep
```
-------------------

```
# sleep 10000 &
# sleep 20000 &
```

Check backgroup process:

```
# jobs
[1]-  Running                 sleep 10000 &
[2]+  Running                 sleep 20000 &
```

-  : Second last added in jobs
+  : last added in jobs

```
# jobs -l  [ with PID ]
[1]-   971 Running                 sleep 10000 &
[2]+   972 Running                 sleep 20000 &
```

```
# fg %2
```

ctrl+z [stop]

```
# bg %2 [ To start in BG ]
```

To check the IP :
~~~
```
# ip addr
OR
# ip a
```

Machine#1 [ ubuntu ]:
~~~

```````````````````
IP : 192.168.206.130

Machine#2 [ Debian ]:
````````````````````
IP : 192.168.206.135

Remote Management:
````````````````
SSH [ Secure Shell ]
```````````````````
port: 22/tcp

package: openssh-server

Service : ssh

1) Install package:
`````````````````

~~~
# apt install openssh-server
~~~

2) Start the service:
````````````````````

~~~
# systemctl start ssh
# systemctl status ssh
~~~

systemctl : This linux command is used to manage linux services. Perform ops like: start, stop, status etc....

Access:
```````
Remote user
Remote IP

~~~
# ssh username@x.x.x.x
~~~

~~~
# git clone https://github.com/sinhakiara/edbda123.git

# cd edbda123/

# cat > username.txt
NAME

# git config --global user.name "NAME"
# git config --global user.email "EMAIL"

# git add .
~~~
```````

~~~
# git commit -m "MSG"

# git branch

# git status

# git log

# git push -u origin main

username: sinhakiara
password: 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
-----------------------
# git branch

# git branch <new_branch>

# git checkout <new_branch>

# cat >> code.py
Modify

# git add .

# git commit -m "commit in branch dev"
--------
Merge the "dev" branch to "main"
````````````````````````````
# git checkout main

# git merge dev
~~~

GIT Version Control System [GVCS]
~~~

***Git was created by Linus Torvalds for the development of the Linux Kernel***

What is Git?
```````````

Git is a version control system.

This means that you can "capture" the exact state of your files and can come back to it any time if you feel like you made some mistakes while changing something there.

When we as Developers are working on a project, then we continuously keep making changes to the code according to the project requirements.

But then sometime we may need to go back to check the previous versions of the code too.
~~~

Basic concepts
``````````````

Your local code
``````````````

This is the work that you do on your computer. Any edits, formats, features or development work that you have in  your computer is your local code.

Staged site
~~~

Once you are happy with the changes or amount of work done, you can mark it as ready for stage. It means that you are declaring that these lines of code are ready to be committed.

The server
``````````

Once you are feel ready with the files you have staged, you can send them out to the server which stores all your code so that other people can use it too. Now your files can be viewed by other people and be worked on.

We may do this to see if:

- The previous version worked better.
- To see what changes were done and when.
- And In Identifying bugs.

✅ Git Helps us track and manage all these changes.

What is GitHub?
~~~

GitHub is a platform which allows you to store this version history on the cloud.

How are Git and GitHub different?
``````````````````````````````

Git - is a tool that helps us track & manage all the changes that were done over time. Git is run and maintained on your Local system.

Whereas,

Github - is a website where you host your project in the form of Git repositories Github is completely Cloud-based.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++
``````````````````````````````

Basics:
```````

Install:
````````

```
# apt install git -y
```

Check version:
~~~

```
# git --version
```

When you start using Git, you'll need to put in a username and an email id :

```
# git config --global user.name "username"
# git config --global user.email "username@email.com"
# git config --global --list
OR
# git config --global -l
```

Use the --unset option to remove a setting:
~~~

```
# git config --unset --global user.email
```

```
# git clone https://githib.com/account_name/repo.git
# cd repo/
# ls -a
.git
```

TO initializing a git repository in the current directory (specified in your terminal)
~~~

```
# mkdir dir
# cd dir
# git init

# ls -a
.git/

# git add .
OR
# git add FILENAME
```

Look at just the repository specific settings
~~~

```
# cat .git/config
[core]
	repositoryformatversion = 0
```

~~~~
        filemode = true
        bare = false
        logallrefupdates = true
[remote "var_name"]
        url = https://github.com/sinhakiara/rce.git
        fetch = +refs/heads/*:refs/remotes/var_name/*


Create a branch:
```````````````
# git checkout -b first
-b: branch name

[Used to switch and create a new branch]

Switch to branch
```````````````
# git checkout first

List branch:
````````````
# git branch

List Remote Branches
```````````````````
# git branch -a




---
# cat >> readme.md
llll
---
# git diff
diff --git a/readme.md b/readme.md
index 039727e..11d14ae 100644
--- a/readme.md
+++ b/readme.md
@@ -1 +1,2 @@
 lol
+llll
```````````````
# git add readme.md

OR

# git add .
# git commit -m "lol1"
[first 835e51c] lol1
 1 file changed, 1 insertion(+)
~~~~

Merge 'test' branch with 'main' branch:
~~~

# git checkout main
# git merge test

To View Your Commits
``````````````````

# git log

show-branch
``````````

# git show-branch
~~~

commit to remote git repository
~~~

1. Setup Name and Email
`````````````````````

# git config --global user.name "Your Name"
# git config --global user.email "your_email@whatever.com"

# mkdir hello
# cd hello

# git clone URL
# cat > hello.py

#!/usr/bin/python3
print("Hello, World")

2. Check the status of the repository
~~~

Use the git status command to check the current status of the repository.

# git status

3. Add Changes
`````````````

# git add hello.py
# git status
# git commit -m "Changes hello"
`````````````

Getting a listing of what changes have been made is the function of the git log command.

~~~
# git log
~~~

One Line Histories
```````````````````

~~~
# git log --pretty=oneline
~~~

Controlling Which Entries are Displayed
`````````````````````````````````````

~~~
# git log --pretty=oneline --max-count=2
# git log --pretty=oneline --since='5 minutes ago'
# git log --pretty=oneline --until='5 minutes ago'
# git log --pretty=oneline --author=<your name>
# git log --pretty=oneline --all
~~~

`````````````````````````````````````````````````````````````

Public Repo Workflow
``````````````````

1. Create a public/private repo on GitHub:

https://github.com/sinhakiara/DevOps-Demo

2. Clone the repo:

~~~
# git clone https://github.com/sinhakiara/DevOps-Demo.git
~~~

3. Create a branch:

~~~
# git checkout "test"
# git branch
~~~

4. Make changes in 'test' branch:

~~~
# cat >> README.md

LOL

# cat > index.html
<h1>ulala</h1>

# git add .

# git status
~~~

```
# git commit -m "Home web page in test branch"
```

5. Merge 'test' branch in 'main' branch:
`

```
# git checkout main
# git merge test
```

6. Remote add the repo:
`

```
# git remote add origin https://github.com/sinhakiara/DevOps-Demo.git

# cat .git/config
```

7. Generate Token :
~~~

Settings --> Developer Settings --> Personal Access Token --> Generate New Token
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

```
# git config --global credential.helper store
```

8. Push the data:
``

```
# git push -u origin main
username:
pass: token
```

.git/

--------------------------------------------
username: sinhakiara

```
# git clone https://github.com/sinhakiara/DAI.git
```

--------------------------------------------

https://github.com/Dheerajmadhukar/karma_v2

Git Internals:
--------------

The .git directory

```
# ls .git
```

The Object Store
````````````````

```
# ls .git/objects
```
~~~

You should see a bunch of directories with 2 letter names. The directory names are the first two letters of the SHA1 hash of the object stored in git.

## Deeper into the Object Store

```
# ls -C .git/objects/<dir>
```

- Look in one of the two-letter directories.
- You should see some files with 38-character names.
- These are the files that contain the objects stored in git.
- These files are compressed and encoded, so looking at their contents directly won't be very helpful, but we will take a closer look in a bit.

## Config File

```
# cat .git/config
```

- This is a project-specific configuration file.

## Branches and Tags

```
# ls .git/refs
# ls .git/refs/heads
# ls .git/refs/tags
# cat .git/refs/tags/v1
```

Each file corresponds to a tag you created with the git tag command earlier. Its content is just the hash of the commit tied to the tag.

## The HEAD File

```
# cat .git/HEAD
```

The HEAD file contains a reference to the current branch.

## Dumping the Latest Commit

```
# git hist --all [not working now]
```

Using the SHA1 hash from the commit listed above ...

```
# git cat-file -t <hash>
# git cat-file -p <hash>
```

## Finding the Tree

```````````````

We can dump the directory tree referenced in the commit. This should be a description of the (top level) files in our project (for that commit). Use the SHA1 hash from the “tree” line listed above.

~~~
# git cat-file -p <treehash>
100644 blob 28e0e9d6ea7e25f35ec64a43fe8386f90        Rakefile
040000 tree e46f374f5b36c6f02fb379044f754d795        lib
~~~

Dumping the lib directory
```````````````````````

~~~
# git cat-file -p <libhash>

100644 blob c45f26b6779fc4c385d9d24fc12cf72          hello.rb
~~~

Dumping the hello.rb file
````````````````````````

~~~
# git cat-file -p <rbhash>

# Default is World
# Author: Jim Weirich (jim@somewhere.com)
name = ARGV.first || "World"

puts "Hello, #{name}!"
-----------
~~~

Manual way to extract blob's content:
```````````````````````````````````

~~~
# git log --pretty=oneline
or
# git log --stat --pretty=oneline
# git show ab35d03c8f4f238fd94f0ec3abd533dbe0b5a352
# git cat-file --batch-check --batch-all-objects | grep blob
# git cat-file -p HASH
~~~

~~~
# git cat-file --batch-check --batch-all-objects | grep blob | awk '{print $1}' | while read -r hash;do git cat-file -p $hash;done | grep "username\|password\|db_user\|db_pass"
~~~

https://learn.kodekloud.com/certificate/8135A1C304-812FB9EBC0-7F11A04DDC
https://learn.kodekloud.com/certificate/2DEF3760A9BA-2DEF315C22BE-2DEF2BD04E92

Q. Create your own image which can run a basic Node.js web server as following:

- Use Image: mhart/alpine-node:4.4

- Use your favourite text editor to add app.js:

````Code Snippet Start````

```
var http = require('http');
http.createServer(function (req, res) {
  console.log(new Date().toUTCString() + " - " + req.url);

  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello, Docker.\n');
}).listen(3000);

console.log('Server running at http://0.0.0.0:3000/');
```

````Code Snippet End````

- Create an entrypoint with the command:

```
 /usr/bin/node app.js
```

Q. Deploy an app using python Flask server & create Dockerfile to build image as follwing:

- Install all required dependencies

- Install Flask

```
pip install flask
```

- The code "app.py":

```
import os
from flask import Flask
app = Flask(__name__)

@app.route("/")
def main():
    return "Welcome!"

@app.route('/hackers')
def hello():
    return 'Hey buddy, how are you?'

if __name__ == "__main__":
```

```
    app.run(host="0.0.0.0", port=8080)
```

- Entrypoint to Start Web Server:

```
python3 app.py
```