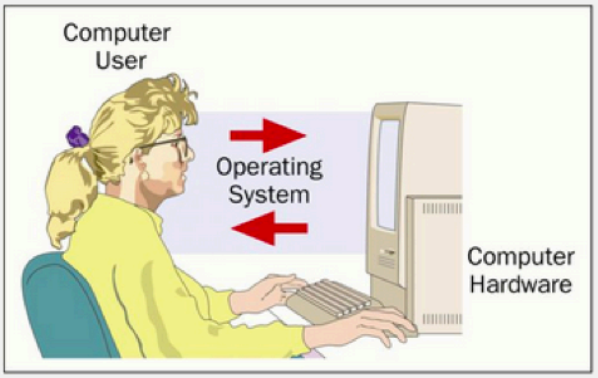
LINUX

LINUX

TOPICS

- Introduction to Linux
- Installation of Centos on VM
- File operations and Directory operations
- VI Editor
- Various Operations and Re-directions
- Managing Users and Groups
- Security
- Package Management
- System Monitoring
- Compression
- Service Management

What is Operating System



Interface between user and the computer hardware.

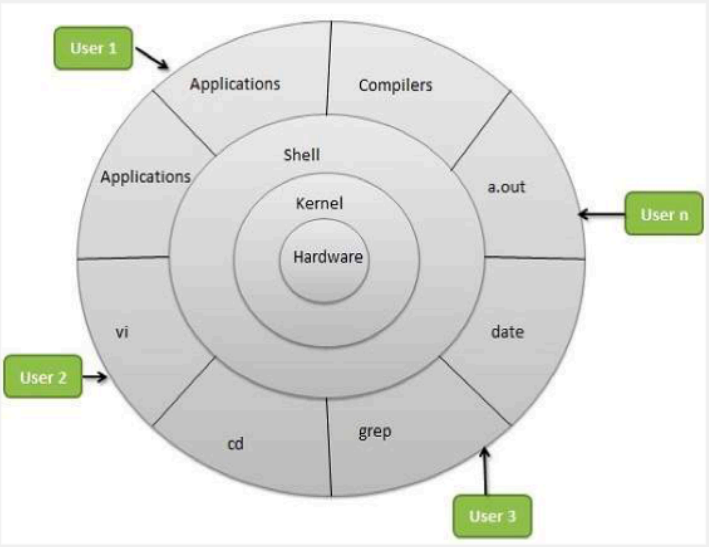
Types Of Operating System

- Single User – Single Tasking OS [MS-DOS]
- Single User – Multitasking OS [Win OS]
- Multi User - Multitasking OS [Unix, Linux]

Why Linux

- Open Source
- Virus-free/Secured
- Compatibility on most hardware
- Customizable
- Well Documented
- Leading operating system on servers

Architecture



- **Kernel**: Central module(Heart) of OS, interacts with Hardware and responsible for memory management, process management etc.
- **Shell**: Shell is a command line interpreter i.e, translates commands entered by user and converts them into a language that is understood by Kernel.
- Shell is an interface between user and kernel

Linux Distributions

- Typically, Linux is packaged in a form known as a *Linux distribution* (or *distro* for short) for both desktop and server use.
- Free Distributions are **CentOS**, **Ubuntu**, **Fedora**, **Debian**, **openSUSE**, etc.,
- Commercial distributions are **Red Hat Enterprise Linux**, **SUSE Linux Enterprise Server**, etc.,

- **Red Hat Enterprise Linux**: commercial version of Linux distro which is aimed at big companies using Linux Servers
- **Centos**: is very close to being RHEL without the branding, support and it's FREE
- **Debian**: Free and it's not for beginners, as it's not designed with ease of use in mind
- **SUSE Linux Enterprise Server**: commercial as it contains many commercial programs, although there's a stripped-down free version **openSUSE**.

Navigation

==========

 # cd {Change your directories}

cd ; cd ~; cd ..; cd - {home, home, exact parent dir, last working dir}

 # touch {create empty files & update the timestamp}

 # cat {create the files, put some data in files, view the data}

 # cat > file.name {using redirection we can copy the data, we can create files }

 # cat >> file.name {appending the data}

 # ls {list the file contents in directory}

cat

===

- Display file contents
- Display Multiple File contents
- Create File using cat
- Copy content from an existing file to new file
- Append one file content to another file
- Redirect multiple files content into a single file

File & Directory Management

==========

 # mkdir {to create the directories. :: **-p** means parent directory creation}

 # cp {to copy files & directories. -r }

 # mv {to move/rename files and directories}

 # rmdir {remove empty dirs}

 # rm {remove files & directories. -r }

Archivers & Compression

==========

An archive is a single file that contains any number of individual files. Archives are convenient for storing files.

Archives are also convenient for transmitting data and distributing programs. In fact, most software that is distributed over the Internet is distributed as an archive that contains all related files as well as documentation.

we can bundle multiple files into a single file using tar.
This can be useful when we want to transfer files from one place to another place.

This is generally used when we are taking backups.

Creating tar file without compression

 # tar -cvf <file_name>.tar <file1> <file2>.....<filen>

Creating tar file with compression

 # tar -cvzf <file_name>.tar.gz <file1> <file2>.....<filen>

Viewing files inside a tar ball

 # tar -tf <file_name>.tar

Extract the tar file

 # tar xvf <file_name>.tar

Extract the tar file in particular location

tar xvf <file_name>.tar -C /var/www/html

c: This is used to create a new archive instead of extracting.
x: Extracts files from the archive.
v: Verbose mode; shows the extraction process in the terminal.
f: Specifies the file to extract from (<file_name>.tar in this case).
C /var/www/html: Extracts the archive into the directory /var/www/html

zip & unzip

=========

Creating a zip file

 # zip <file>.zip <file1> <file2>.....<filen>

Viewing files inside a zip archive

 # unzip -l <file>.zip

Extracting the zip file

 # unzip <file>.zip

Extract the zip file in particular location

 # unzip <file>.zip -d /var/www/html

Linux File System

==========

A Linux file system is essential for organizing and managing data efficiently and securely. It provides a foundation for the operating system, applications, and users to interact with stored data seamlessly.

/bin :: contains the executable (i.e., ready to run) programs for all users ex: cat, ls

/boot :: Boot loader files ex startup files and kernel

/dev :: Device files interesting directory that highlights one important aspect of the Linux filesystem - everything is a file or a directory

Device files (e.g., `/dev/sda` for storage devices).

/etc :: system wide configuration files

/home :: Users' home directories, containing saved files, personal settings, etc.

/lib :: Library files, includes files for all kinds of programs in /bin and /sbin

/mnt :: Mount points for removable media such as CD-ROMs, usb sticks etc

/opt :: Optional softwares, Typically contains extra and third party software, anything that isn't part of the base system.

/proc :: Virtual filesystem providing process and kernel information

$ **cat /proc/meminfo** && **cat /proc/cpuinfo**

/root :: Home directory for the root user.

/sbin :: Programs for use by the system and the system administrator

/tmp :: Temporary space for use by the system, cleaned upon reboot, so don't use this for saving any work!

/usr :: User programs and libraries.

/var :: Variable files, whose content is expected to continually change such as logs.

What is Absolute Path & Relative Path

==========

The path with reference to the root directory is called absolute.

The path with reference to the current directory is called relative.

Secure Shell(Port no 22)

SSH, or Secure Shell, is a cryptographic network protocol used for securely connecting to and accessing remote servers or devices over an unsecured network. It provides a secure, encrypted connection for communication between the client and the server, allowing users to securely log in to remote systems and execute commands or transfer files.

SSH keygen command:

$ ssh-keygen (creates public key and private key under .ssh folder)

Vi/Vim {Text Editor}

==========

Cat has problems with the Navigation.

Vi/Vim can be used to edit the text files and it also provides following features:

Navigation Support

Search Functionality

Undo/Redo Operations

vi <file_name>

i - insert the data

a - insert (append) after the cursor.

esc {to come out of insert mode} && :wq! {save and quit}

Navigation:

==========

h - left arrow

l- right arrow
j-down arrow
k - up arrow

To go word by word in a forward direction - w & 5w
To go word by word in backward direction - b & 5b

Page Up - **ctrl + u**
Page Down - **ctrl + d**

Beginning of the File - gg
End of the file - G

To go into command mode use :
To view the line numbers **:se nu**
To go to a particular line we use **30G** (or) **:30**

Editing using Vi
==========

To go into insert mode we use "i"
In insert mode commands doesn't work
Once editing is done save changes by following:
:wq! {w-save, q-quit & !-forcefully}

a - append diff is in insert mode character is placed before the cursor position in append mode character is placed after the cursor position

I - insert at beginning of the line
A - insert at the end of the line

o - put text in a new line below current line
O - put text in a new line above current line

x - will delete a character for you
5x - from cursor position next 5 characters will be deleted
dw - delete word

4dw - delete 4 words

dd - delete your line

5dd - delete 5 lines

:3,15d - delete lines from 3 to 15

u - undo operation

cntrl + r - redo operation

yl + p - copying a character to paste it we use "p"

yw + p - copy and pastes a word {cursor position}

yy + p - copies and pastes current line

Y + p - copies and pastes current line

Search and Replace:

:s/old/new – "old" with "new" on current line

:%s/old/new/g – Replace all occurrences of "old" with "new" in the entire line

/pattern – Search forward for a pattern

?pattern – Search backward for a pattern

Setting Up User

==========

1. Login as root user
2. useradd <user_name> {adds a user}

   # useradd sysadmin

3. passwd <user_name> {setup password for user}

   # passwd sysadmin

4. # usermod -aG <group_name> <user_name> { grant **admin** privileges}

   # usermod -aG wheel sysadmin

5. # id <user_name> {get user info & group info}

   # id sysadmin

User & Group Management

==========

User:

====

is an entity who will login into the system and uses system resources to do some operations.

Types of Users

==========

We have 3 types of users in Linux:

1. Super user	**{root user}**
2. System user	**{apache, nginx etc}**
3. Normal user	**{useradd}**

Super/Root user

==========

special user who holds all kinds of permission to do any alteration to the system.

System Users

==========

this kind of users are typically associated with system applications like httpd, nginx etc

Normal Users

==========

These kinds of users are added by sys admins by useradd command.
These users work with the system and will use system resources.
All the user information is stored in this file # vi /etc/passwd
There are the 7 fields

Devops:x:1000:1000:deafult user:/home/devops:/bin/bash
ec2-user:x:1000:1000:EC2 Default User:/home/ec2-user:/bin/bash

1 2 3 4 5 6 7

1 : username
2 : password

3 : userid
4 : groupid
5 : Comment
6 : home directory
7 : Default shell

 # vi /etc/shadow { encrypted password location }

 # sudo su { to switch to root user }

User management
==========

- # useradd
- # usermod
- # userdel
- # useradd <uname>
- # passwd <uname>
- # useradd -c "Comments" <uname>
- # usermod -l <new_name> <old_name>
- # userdel <uname>

Use the chage command followed by the -M option to set the maximum number of days before the password must be changed. For example, to set the password expiration to 90 days for a user named "username":

$ sudo chage -M 90 username

$ sudo chage -l username

Group Management
==========

Group
=====

a group is a collection of users with some common goals.
Groups can be assigned to logically tie users together for a common security, privilege and access purpose.
groups can be assigned common permissions.

Types of Groups:

1. Primary Group

- Each user in Linux is assigned a primary group, which is specified in the `/etc/passwd` file.
- A primary group is the default group that a user's files are associated with when they are created.
- When a user creates a file, the file's group ownership is set to the user's primary group.

2. Supplementary (Secondary) Groups

- Users can belong to additional groups, which are referred to as supplementary or secondary groups.
- These groups provide users with additional permissions, often related to access to specific resources or directories.
- A user can be a member of multiple secondary groups, and these are defined in the `/etc/group` file.

All the group related information is present: **# vi /etc/group**

Adding a group:

 # sudo groupadd <group_name>

Remove a user from group

sudo gpasswd -d <username> <group_name>

File Permissions

==========

We got two ways of changing the permissions

1. **Symbolic Mode**
2. **Absolute Mode**

Symbolic Mode

==========

+ :: adds user designated permission to file/directory

- :: removes user designated permission to file/directory

= :: set user designated permission to file/directory

sudo chmod u+x <file>
sudo chmod g-x <file>
sudo chmod o=rwx <file>
sudo chmod u+x,g-r,o=rwx <file>

Absolute Mode

==========

Using numbers we specify permissions:

read :: 4
write :: 2
execute :: 1

sudo chmod 755 <file>. {rwx r-x r-x}

sudo chmod 644 <file>. {rw- r-- r--}

sudo chmod -R 644 <file>. {rw- r-- r--}

sudo chmod 700 <file>. {rwx --- ---}

Changing Ownership

==========

chown is used to change the ownership of the files & directories.

Changing only file owner

sudo chown <uname> <file>

Changing only group owner

sudo chgrp <group_name> <file>

Changing both file & group owner in one go

sudo chown <uname>:<group_name> <file>

sudo chown -R <uname>:<group_name> <file>

Note: Many operations on directories require execute (search) permission in addition to read permission. `chmod 666` clears the `x` bits, causing strange failures of `ls` and other basic stuff. Reasonable default permissions might be 644 for files and 755 for directories.

Package Management

==========

Package

=======

Package refers to a compressed file that contains all the files that come with a particular application.

code file
config file
Manual page
Example files

In RHEL/CentOS, the package management tool will be called RPM {Redhat Package Manager}

What would the package management system do for you ??

1. Install a new package {application}
2. Upgrade the package {application}

3. Erase the package {uninstall the application}

To work with packages we use a command called # rpm

rpm {command to work with Redhat Package Manager}

package Format: <package_name>-<version>-<release>-<arch>.rpm

elinks-0.12-1.mga1.x86_64.rpm
elinks-x.y.z elinks-major_Version.minor_Version.build_Version

Different options available with rpm:

rpm -q query

-a = show me all packages
-i = show me detailed info about package
-l = list of files inside the package
-f = find the relation b/w a file and package

rpm -qa → shows all packages

wc-l → count no of lines

rpm -qa | wc -l

rpm -qi <pkg_name> → detailed info abt package. { i }

rpm -ql perl → shows location of all files distributed across the system when an application{perl} is installed

rpm -qf /etc/hosts → gives info about, what package the file belongs to

Using rpm command
==========

Instal =======

Download the rpm file {rpmfinder}

wget <file>.rpm

rpm -ivh <file>.rpm

Upgrade

=======

Download the new upgraded rpm then upgrade it

rpm -Uvh <new_file>.rpm

Erase

======

rpm -e <application>

Limitations through RPM

==========

We need to download the package

Need to check the dependencies if any

In olden days to install a package there were many problems bcz of the above two problems.

To overcome the above problems and make the process easy, they have created a function called repository.

What is a Package Repository ?

==========

It is a collection of all related files, w.r.t given OS version

Yum ==========> command to access repo

Yum :: Yellowdog Updater, Modified

By default we have some basic repositories for our use. But to use them:

We need internet

We Need to identify which package we need to work

 # man yum

Sl No.	RPM	YUM
1	If we want to install an application(Ex: apache), rpm need to install all the packages required for this application, these packages may vary from 1 rpm to several rpm's depending on shared rpm packages.	Install an application with single command Ex: yum install httpd
2	RPM package dependencies is bit tough	YUM resolves dependencies with ease
3	Batch installation of applications is possible with one command	YUM command can install number of applications in one single command Ex: yum install httpd vsftpd
4	RPM can not handle updated software installation automatically	Does YUM install updates of the existing packages by using yum install upgrade
5	Can not connect to online repositories	Can connect to on-line repositories to get latest software before installing the applications

Advantages of repo

==========

Speeds up the download process

It falls over to nearest repo automatically

Repos info and repos are stored in **# /etc/yum.repos.d**

we have .repo files in yum.repos.d =======> these are files what contain pointer towards the actual repo

Commands to work with repos

==========

yum clean all → to clear all clutter

yum repolist → to give current active repos

yum install <package_name> → Install package

yum install epel-release → Important {epel - Extra Packages For Enterprise Linux}

above command Installs epel repo

YUM {Yellowdog Updater Modified}
==========

It can download the packages(rpm's) from repositories
It can automatically update the packages
It can download the dependencies automatically

**** We need to have an internet connection ****

yum/apt info <app>

yum list installed

yum install <app>

yum update <app>

yum remove <app>

yum search <app>

yum whatprovides <app> (Find which package provides a specific file, command, or library.)

cd /etc/yum.repos.d

Common package management systems and their associated repositories include:

Advanced Package Tool (APT):
Used by Debian and Ubuntu-based distributions.
Repositories include Debian repositories and Ubuntu repositories.

Yellowdog Updater, Modified (YUM):
Used by Red Hat, CentOS, and Fedora.

Repositories include Red Hat repositories, CentOS repositories, and Fedora repositories.

PackageKit:
A cross-distribution package management solution used by several Linux distributions.

Homebrew:
Used on macOS.

Chocolatey:
Used on Windows.

Service Command
==========

The service command starts or stops a service by running an initialization script.

The SCRIPT parameter, located in /etc/init.d/SCRIPT

For upstart jobs,
start, stop, and status are passed through to their upstart equivalents.

service <service> start

Systemctl command
==========

What is systemctl?
systemctl is the command-line interface (CLI) tool to manage systemd. It allows you to:

1. Start, stop, restart, enable, or disable services.
2. Check the status of services.
3. Manage system targets.
4. Reload the configuration of services without restarting them.

systemd is a system and service manager for Linux that initializes the system during boot and manages system processes and services.

systemctl start <service>

stop

status

reload

restart

systemctl list-units

systemctl --failed

systemctl is-enabled <service>

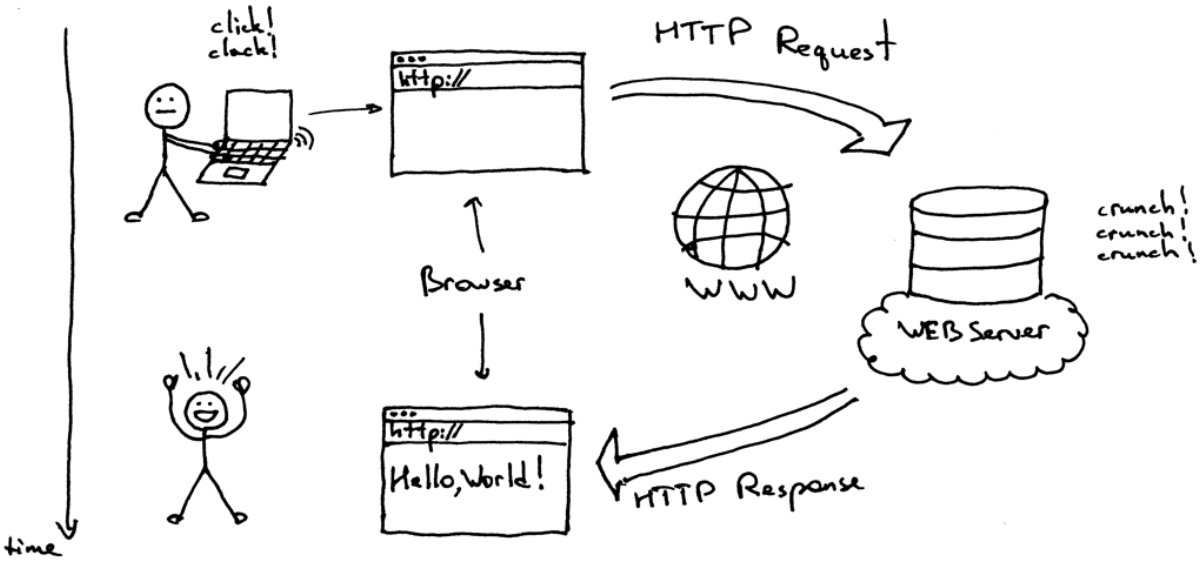
systemctl enable <>

systemctl disable <>

systemctl kill <service>

Web Server
==========

A web server is a program which serves web pages to users in response to their requests, which are forwarded by their computers' HTTP clients(Browsers).



All computers that host websites must have a web server program.

Purpose of Web server

==========

A web server's main purpose is to store web site files and broadcast them over the internet for you site visitors to see. In essence, a web server is simply a powerful computer that stores and transmits data via the internet.

Web servers are the gateway between the average individual and the world wide web.

Apache

=======

An **open source web server** used mostly for **Unix and Linux platforms**.

It is **fast, secure and reliable**.

Since 1996 Apache has been the most popular web server, presently **Apache holds 49.5%** of market share i.e, 49.5% of all the websites followed by **Nginx 34%** and **Microsoft IIS 11%.**

Parameters for Apache (httpd)

==========

Packages	**-**	**httpd/apache2 & mod_ssl**
Port	**-**	**80 443**
Protocol	**-**	**http https**
Server Root	**-**	**/etc/httpd, /etc/apache2**
Main config file	**-**	**/etc/httpd/conf/httpd.conf**
Document root	**-**	**/var/www/html {all our web pages }**
Logs	**-**	**/var/log/httpd/error_log**
		/var/log/httpd/access_log

What version of OS you have
==========

 # cat /etc/os-release

Checking httpd service is running or not
==========

6.x(Os-version) - Whenever you have problem with httpd {troubleshooting httpd}

 # service httpd status
 # netstat -ntpl | grep 80
 # ps -ef | grep httpd

7.x(Os-version) - Whenever you have problem with httpd {troubleshooting httpd}

 # systemctl status httpd
 # netstat -ntpl | grep 80
 # ps -ef | grep httpd

Now when you are doing server configuration, we need to be very careful while doing server configuration, coz sometimes you may do some typos and you are unaware why the server is not starting, so to confirm your config is correct use following command:

Check configuration file (httpd.conf) is correct or not

==========

 # httpd -t

Installing httpd service
==========

 # sudo yum -y install httpd

Let's **check status** of the web server:

systemctl status httpd
netstat -ntpl | grep 80
ps -ef | grep httpd

Let's **start the web server**:

systemctl start httpd
systemctl status httpd
netstat -ntpl | grep 80
ps -ef | grep httpd

Now we have started the service
If you see **# ls -l /var/www/html**, you have no files in there, let's create a index.html

 # cd /var/www/html
 # vi index.html {put some content}

Hosting a Sample Food website

Switch to /var/www/html and Clone the source code by using below command

$ git clone https://github.com/saiurakrishna/food-app.git

Host multiple Websites by using Virtual Hosting

Apache Virtual Hosting
==========
At the end of file add these lines

<VirtualHost *:81>
DocumentRoot /var/www/html/website-2

</VirtualHost>

<VirtualHost *:82>
DocumentRoot /var/www/html/website-3
</VirtualHost>

Shell Scripting:

Shell scripting is the process of writing a series of commands in a file to be executed by a Unix/Linux shell (such as Bash, Zsh, or Sh). It allows for task automation, system administration, and repetitive command execution.

Two types of variables -->

#!/bin/bash → Shebang
System variables: In upper cases $BASH, $BASH_VERSION $HOME $PWD $USER

User Defined variables: name=bob
echo The name is $name

Whenever you want to take the input from the terminal use Read command

```
echo "Enter name : "
read name1 name2 name3
echo "Entered name : $name1, $name2, $name3"
```

Read inputs from your terminals (take inputs from terminal and allows to store in variables)

```
read -p 'username :' user_var
read -sp 'password :' pass_var

echo "username : $user_var"
echo "password : $pass_var"
```

IF --> To evalute some conditions

```
if [ condition ]
then
 statement
fi
```

Integer comparison:

x=10
-eq --> is equal to -
-ne --> not equal to
-gt --> greater than
-ge --> greater than or equal to

-lt ---> less than
-le --> less than or equal to

Simple IF:

```
if [ $x -eq 9 ]
then
  echo "condition is true"
fi
```

```
if (( $x > 9 ))
then
echo "condition is true"
fi
```

adding else condition

```
if [ $count > 9 ]
then
echo "condition is true"
else
echo "condition is false"
fi
```

If, elif and else:

You want to evaluate each condition in sequence and proceed to the next one only if the previous one is false. Here's how you can do it using `if`, `elif`, and `else`:

```
#!/bin/bash

# Read an integer input from the user
read -p "Enter an integer: " number

# Check the conditions in sequence
if [ "$number" -gt 10 ]; then
    echo "The number is greater than 10."
elif [ "$number" -eq 5 ]; then
    echo "The number is equal to 5."
elif [ "$number" -lt 0 ]; then
```

```
    echo "The number is less than 0."
else
    echo "The number does not satisfy any of the conditions."
fi
```

True + True = True → AND

True + False = True → OR

AND Operator:

```
age=24

if [ "$age" -gt 12 ] && [ "$age" -lt 35 ]  --> if [ "$age" -gt 12 -a "$age" -lt 35 ]
then
echo "valid age"
else
echo "age is not valid"
fi
```

```
OR Operator:if [ "$age" -gt 12 -o "$age" -lt 35 ]
then
echo "valid age"
else
echo "age is not valid"
fi

if [ "$age" -gt 12 ] || [ "$age" -lt 35 ]  -->
```

While Loop:

loops are used to repeat a specific set of commands or actions multiple times. They provide a way to automate repetitive tasks or process a series of elements in a collection
The while loop repeatedly executes a block of code as long as a specified condition remains true

A **for loop** in shell scripting is used to **iterate** over a list of items or a range of numbers. It allows you to perform a set of commands repeatedly for each item in the list or range.

```
for variable in list
do
  # Commands to execute for each item
done
```

Script for creating users named user1, user2… user10
#!/bin/bash

```
for i in {1..10}
do
  username="user$i"
  sudo useradd $username
  echo "User $username created successfully."
done
```

Explanation:

- The loop iterates from 1 to 10.
- `i` in the script is a **loop control variable** used to represent the current iteration number in the **`for` loop**.
- useradd is used to create a new user with the name user$i.
- Each user is named as user1, user2, ..., user10.
- sudo is required for user creation, so make sure you run this script with root privileges.

Diff B/w For and While Loop

1. for Loop
   - Used to iterate over a fixed list or range of values.
   - Best when the number of iterations is known beforehand.
   - Easier to work with sequences, lists, and arrays.

2. while Loop
   - Executes repeatedly as long as a condition is true.
   - Best when the number of iterations is unknown beforehand or based on a condition.
   - Often used for reading files, waiting for a process, or validating user input.

Shell script to perform basic arithmetic operations (addition, subtraction, multiplication, and division) on two variables a and b where a=10 and b=20.

1. Addition

```
#!/bin/bash
a=10
b=20
sum=$((a + b))
echo "Addition of $a and $b is: $sum"
```

2. Subtraction

```
#!/bin/bash
a=10
b=20
```

```
diff=$((a - b))
echo "Subtraction of $a and $b is: $diff"
```

3. Multiplication

```
#!/bin/bash
a=10
b=20
prod=$((a * b))
echo "Multiplication of $a and $b is: $prod"
```

4. Division

```
#!/bin/bash
a=10
b=20
div=$((a / b))
echo "Division of $a by $b is: $div"
```

Install Food-app script

```
#!/bin/bash
sudo yum update -y
sudo yum install httpd git -y
sudo systemctl start httpd
sudo systemctl enable httpd
sudo chown -R $USER:$USER /var/www/html
cd /var/www/html && git clone https://github.com/saiurakrishna/food-app.git
sudo cp /etc/httpd/conf/httpd.conf /etc/httpd/conf/httpd.conf.default
sudo sed -i 's|/var/www/html|/var/www/html/food-app|g' /etc/httpd/conf/httpd.conf
sudo systemctl restart httpd
```

CronJob:

In Linux, cron is a time-based job scheduler that allows users to automate repetitive tasks by scheduling scripts or commands to run at specific intervals. It is widely used for system maintenance, backups, and other repetitive tasks.

Cron Job

A cron job refers to the scheduled task itself, which can be a command or script that you want to run automatically at a specified time and frequency. Cron jobs are

managed by the `cron` daemon (`crond`), which runs in the background and executes tasks at the scheduled time.

Crontab

The crontab (cron table) is a file that contains the cron jobs for a user. It defines the schedule and the tasks that should be executed by the `cron` service. Each user can have their own crontab file, and there is also a system-wide crontab that can be used for scheduling tasks for system maintenance or admin-level tasks.

Crontab Syntax

Each line in the crontab file represents a cron job and follows this syntax:

```
* * * * * command_to_be_executed/script.sh
- - - - -
| | | | |
| | | | ----- Day of the week (0 - 7) (Sunday=0 or 7)
| | | ------- Month (1 - 12)
| | --------- Day of the month (1 - 31)
| ----------- Hour (0 - 23)
------------- Minute (0 - 59)
```

Example of a Crontab Entry

```
0 2 * * * /home/user/backup.sh
```

This cron job runs the script `/home/ec2-user/backup.sh` at 2:00 AM every day.

Managing Cron Jobs

To manage cron jobs, you can use the `crontab` command:

- `crontab -l`: List your cron jobs.
- `crontab -e`: Edit your crontab file.
- `crontab -r`: Remove your crontab file (delete all cron jobs).

Crontab allows you to automate and schedule tasks with great flexibility, making it an essential tool for system administrators and developers.

Sample File Backup Script: `backup.sh`

```
#!/bin/bash

# Variables
SOURCE_DIR="/home/ec2-user/devops"  # Directory to backup
BACKUP_DIR="/home/ec2-user/backup"      # Where the backup
will be stored
TIMESTAMP=$(date +"%Y-%m-%d_%H-%M-%S")      # Timestamp
format for the backup
BACKUP_NAME="backup_$TIMESTAMP.tar.gz"      # Name of the
backup file

# Create backup directory if it doesn't exist

mkdir -p "$BACKUP_DIR"

# Backup the directory
tar -czf "$BACKUP_DIR/$BACKUP_NAME" "$SOURCE_DIR"

# Output message
echo "Backup of $SOURCE_DIR completed. Backup stored as
$BACKUP_DIR/$BACKUP_NAME"
```

How the script works:

1. **Variables:**
    - `SOURCE_DIR`: The directory that you want to back up.
    - `BACKUP_DIR`: The directory where the backup will be stored.
    - `TIMESTAMP`: Adds a timestamp to the backup file name for uniqueness.
    - `BACKUP_NAME`: The name of the backup file, which includes the timestamp.
2. **Backup Creation:**
    - The `mkdir -p` command ensures that the backup directory exists; if it doesn't, it creates the directory.
    - The `tar` command compresses the contents of `SOURCE_DIR` into a `.tar.gz` file and stores it in `BACKUP_DIR`.
3. **Output:**
    - The script will print a message when the backup is complete, showing where the backup has been stored.

You can find the shell scripts here
https://github.com/saiurakrishna/shell-script/tree/master/shell-script-file

Sed command:
Here, the -i option is used to edit the file in place. The s command is used to substitute the old folder path with the new folder path. The | character is used as a delimiter instead of the usual / to avoid issues with the folder path itself containing / characters. The g flag is used to make the substitution globally, replacing all occurrences of the old folder path in the file.
$ sed 's/old_content/new_content/g' input_file

Assignments:

1. Create a user with own home directory
2. Remove the user along with home directory
3. Create a script that asks the user input from terminal for two numbers and performs basic arithmetic (addition, subtraction, multiplication, division)
4. Set an expiration date for a specific user account
5. Create a New User with a Specific Group and Home Directory
6. Create 10 users in linux with shell script (UID: 1010 to 1020)
7. Execute the backup script on alternate days(Mon, Wed, Friday) by using Cron Jobs
8. Create a Shell script for install and configure the Foodapp

GIT

GIT

Why Version Control ??
==========

Have you ever:

- Made a change to code, realized it was a mistake and wanted to revert back?
- Lost code and didn't have a backup of that code ?
- Had to maintain multiple versions of a product ?
- Wanted to see the difference between two (or more) versions of your code ?
- Wanted to prove that a particular change in code broke an application or fixed an application ?
- Wanted to review the history of some code ?
- Wanted to submit a change to someone else's code ?
- Wanted to share your code, or let other people work on your code ?
- Wanted to see how much work is being done, and where, when and by whom ?
- Wanted to experiment with a new feature without interfering with working code ?

In these cases, and no doubt others, a version control system should make your life easier.

Key Points

========

- **Backup**
- **Collaboration**
- **Storing Versions**
- **Restoring Previous Versions**
- **Understanding What Happened**

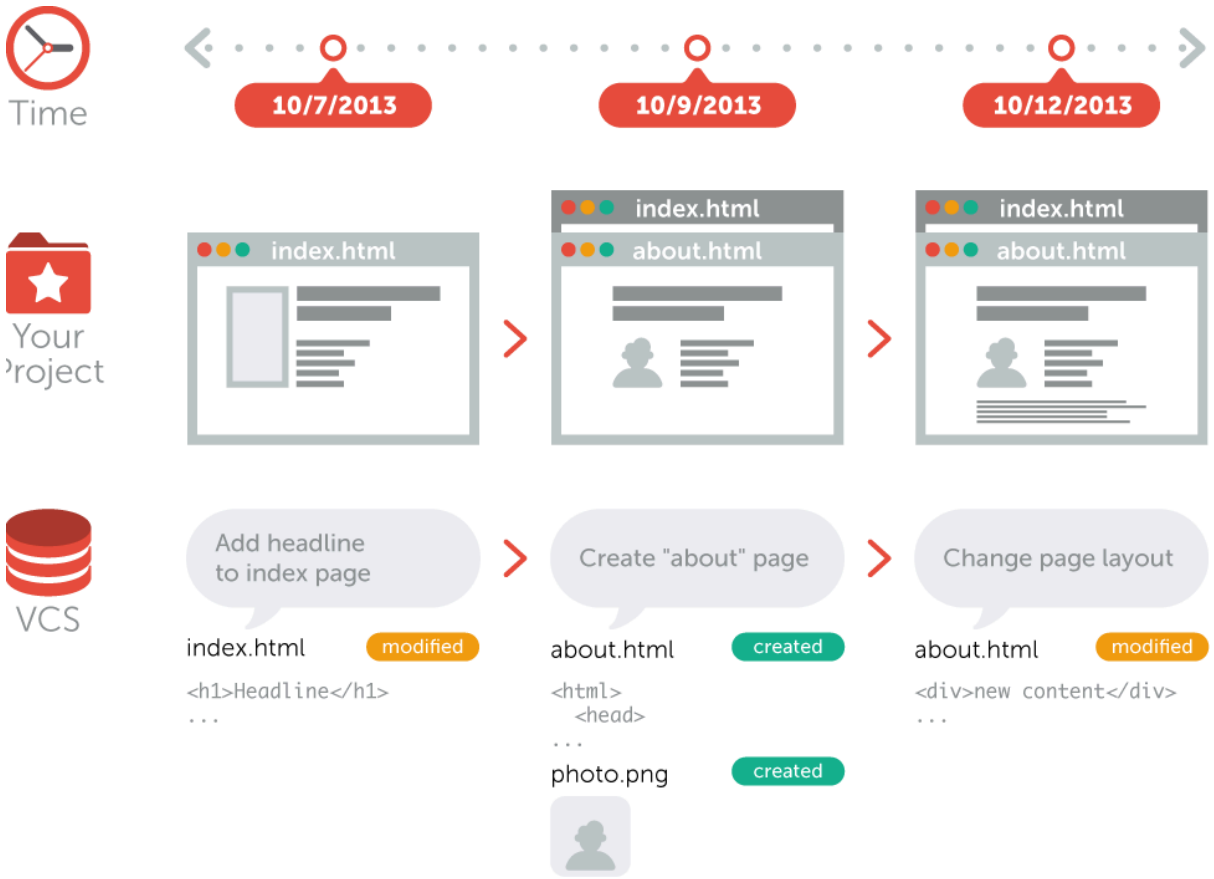
Version Control / Revision control / Source Control is a software that helps software developers to work together and maintain a complete history of their work.

You can think of a version control system ("VCS") as a kind of "**database**".

It lets you save a snapshot of your complete project at any time you want. When you later take a look at an older snapshot ("version"), your VCS shows you exactly how it differed from the previous one.

A version control system **records the changes** you make to your project’s files.

This is what version control is about. It's really as simple as it sounds.


Time
10/7/2013
10/9/2013
10/12/2013
Your Project
index.html
index.html
about.html
index.html
about.html
VCS
Add headline to index page
Create "about" page
Change page layout
index.html
modified
<h1>Headline</h1>
...
about.html
created
<html>
<head>
...
photo.png
created
about.html
modified
new content
...

Popular VCS

==========

Types of VCS

==========

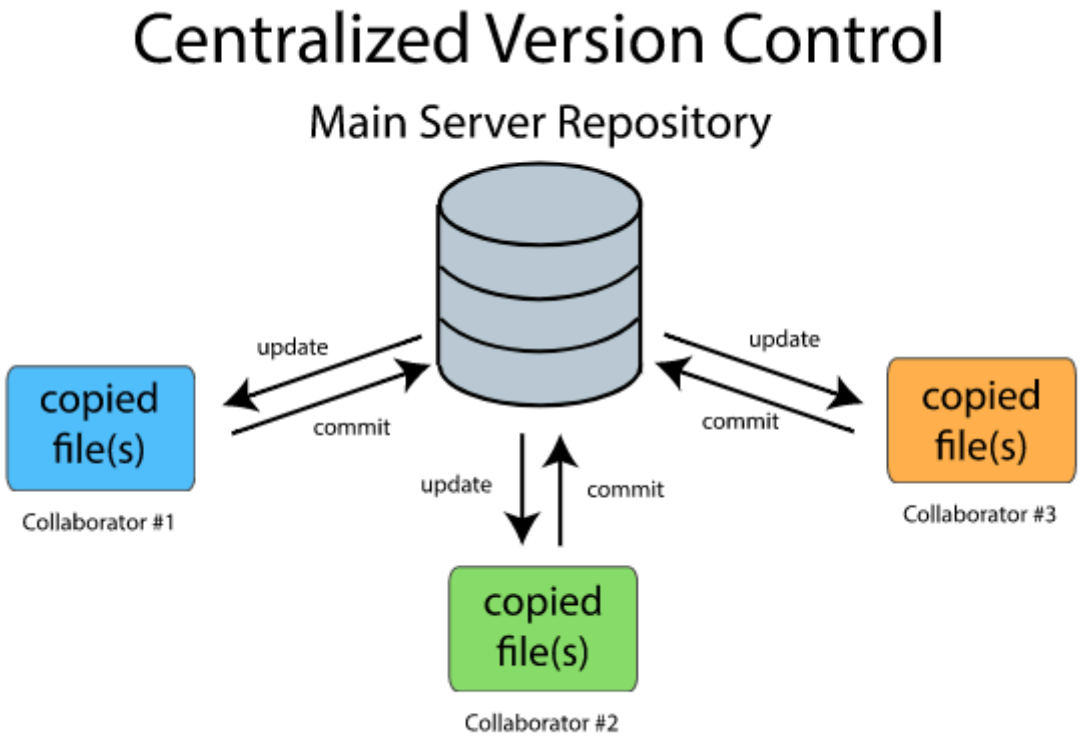
- **Centralized version control system (CVCS)**
  - **Ex: CVS, SVN**
- **Distributed version control system (DVCS)**
  - **Ex: Git, Mercurial**

Centralised Version Control System (CVCS)

==========

Uses a central server to store all files and enables team collaboration.

But the major drawback of CVCS is its **single point of failure,** i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all.

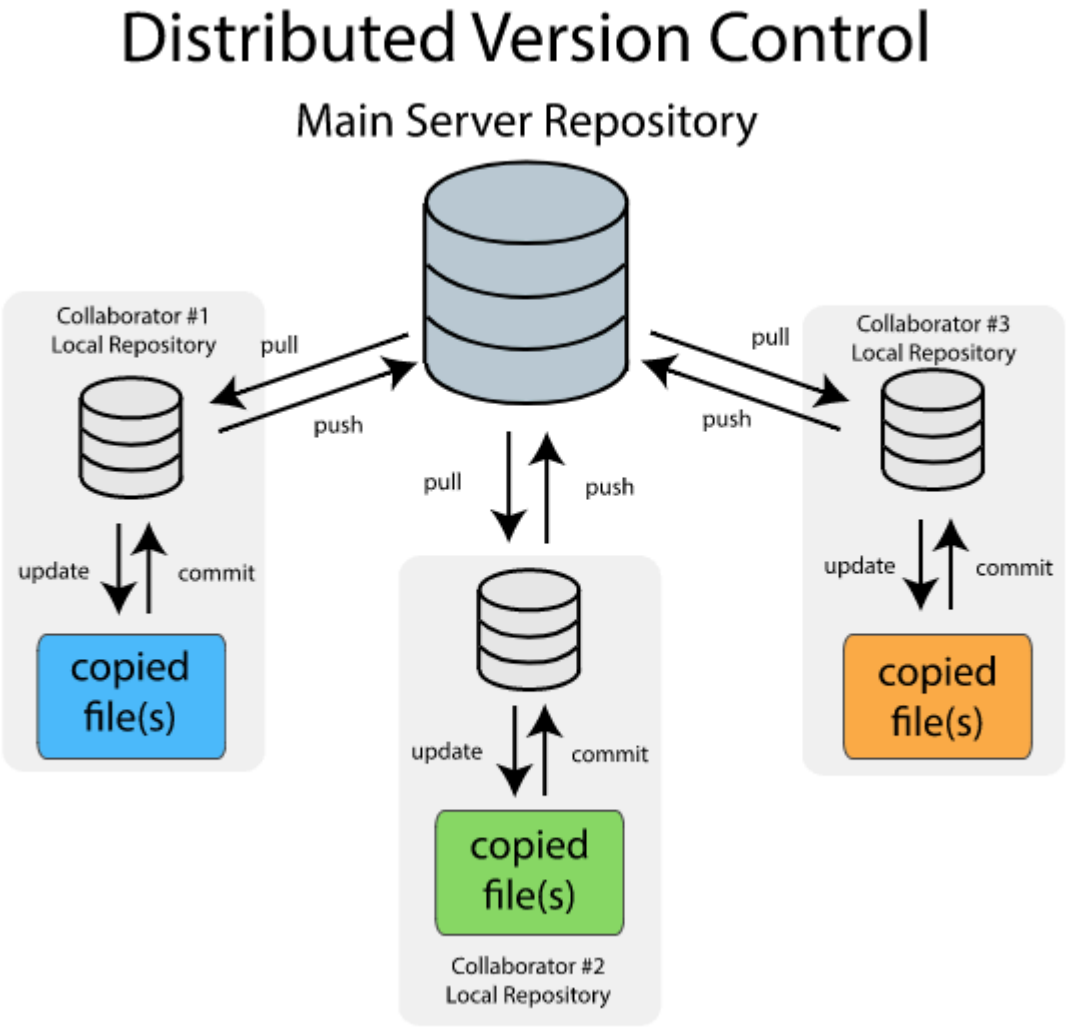


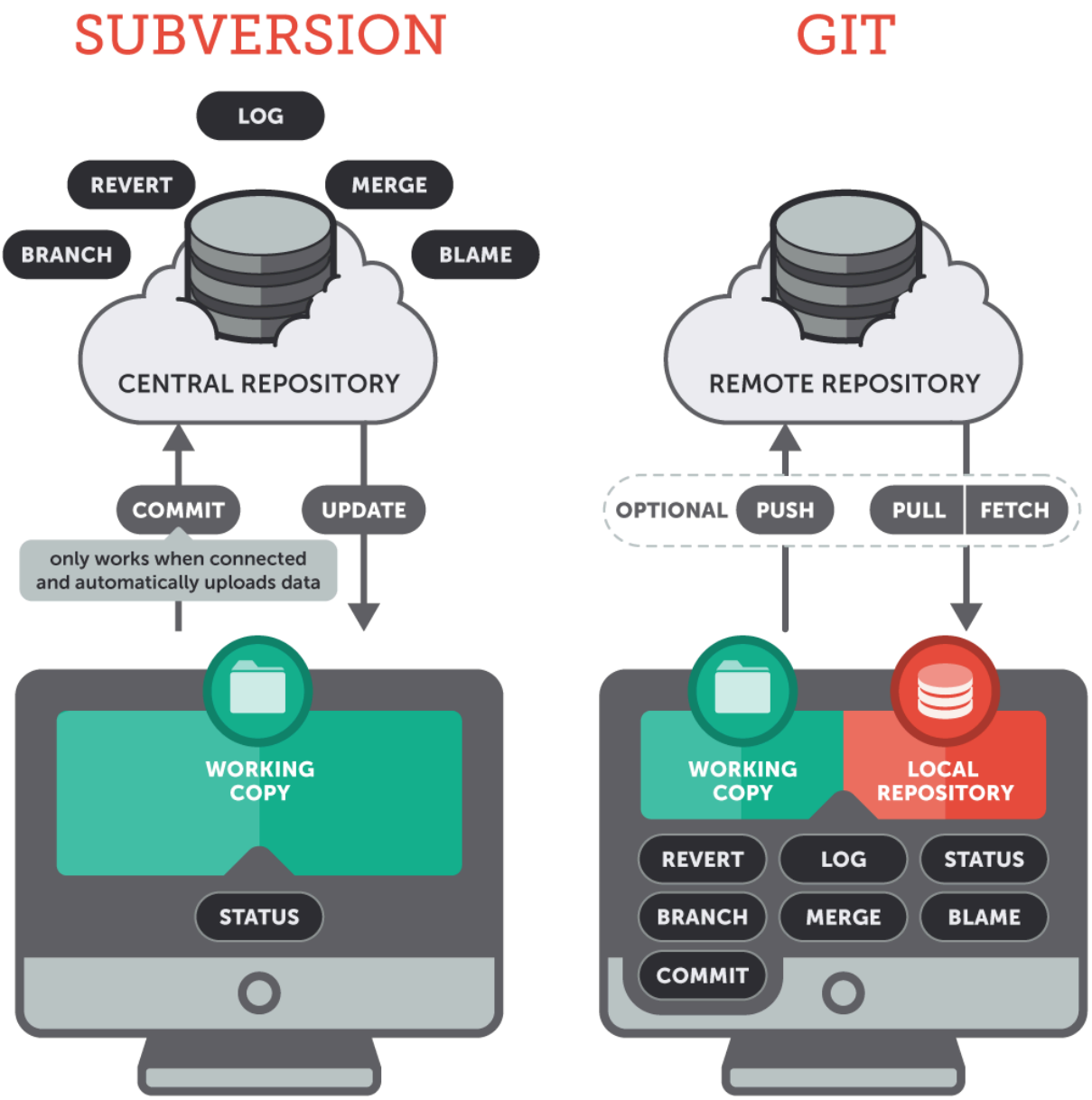
Distributed Version Control System (DVCS)

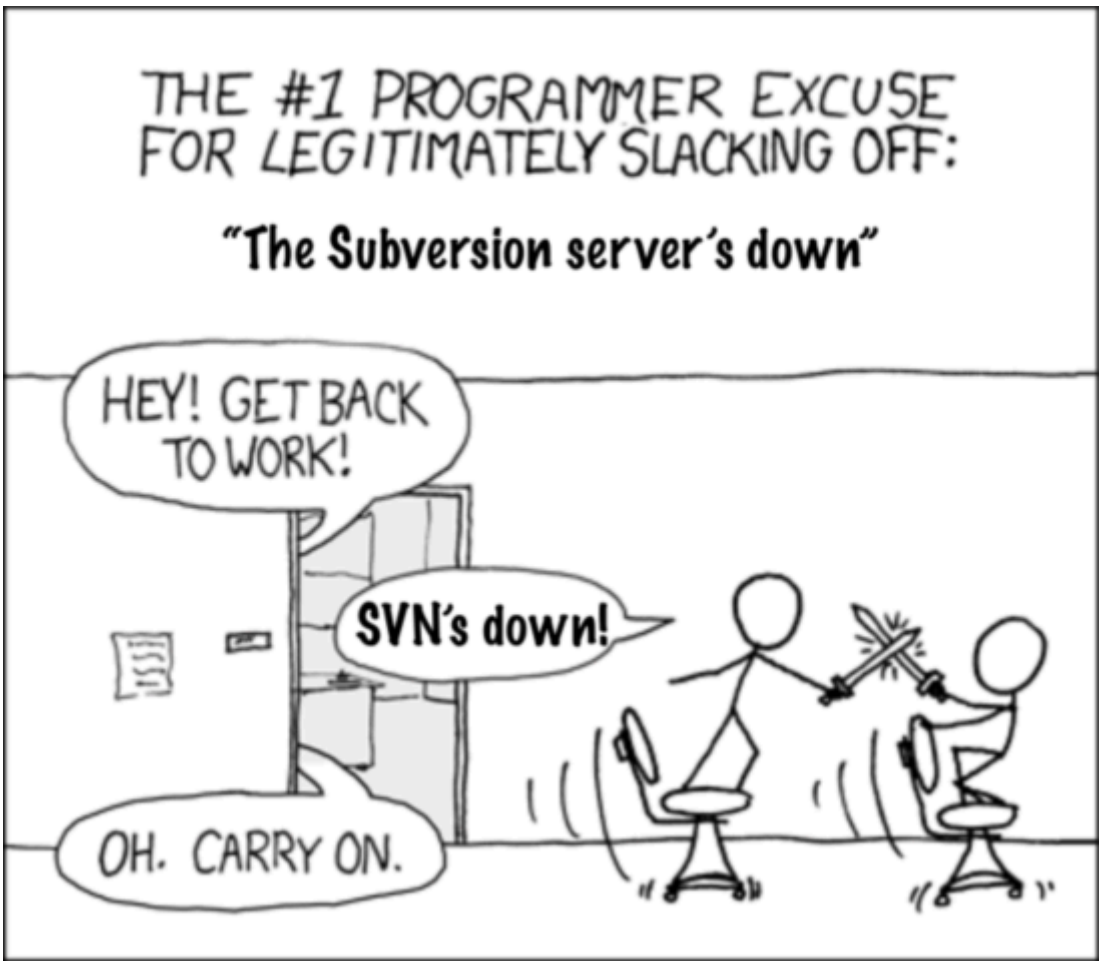
==========

DVCS does not rely on the central server and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline.

You require a network connection only to publish your changes and take the latest changes.




SUBVERSION
LOG
REVERT
MERGE
BRANCH
BLAME
CENTRAL REPOSITORY
COMMIT
UPDATE
only works when connected
and automatically uploads data
WORKING
COPY
STATUS
GIT
REMOTE REPOSITORY
OPTIONAL
PUSH
PULL
FETCH
WORKING
COPY
LOCAL
REPOSITORY
REVERT
LOG
STATUS
BRANCH
MERGE
BLAME
COMMIT



How the Typical VCS works

==========

A typical VCS uses something called **Two tree architecture**, this is what a lot of other VCS use apart from git.

Usually, a VCS works by having two places to store things:

1. **Working Copy**
2. **Repository**

These are our two trees, we call them trees because they represent a file structure.

Working copy [CLIENT] is the place where you make your changes.

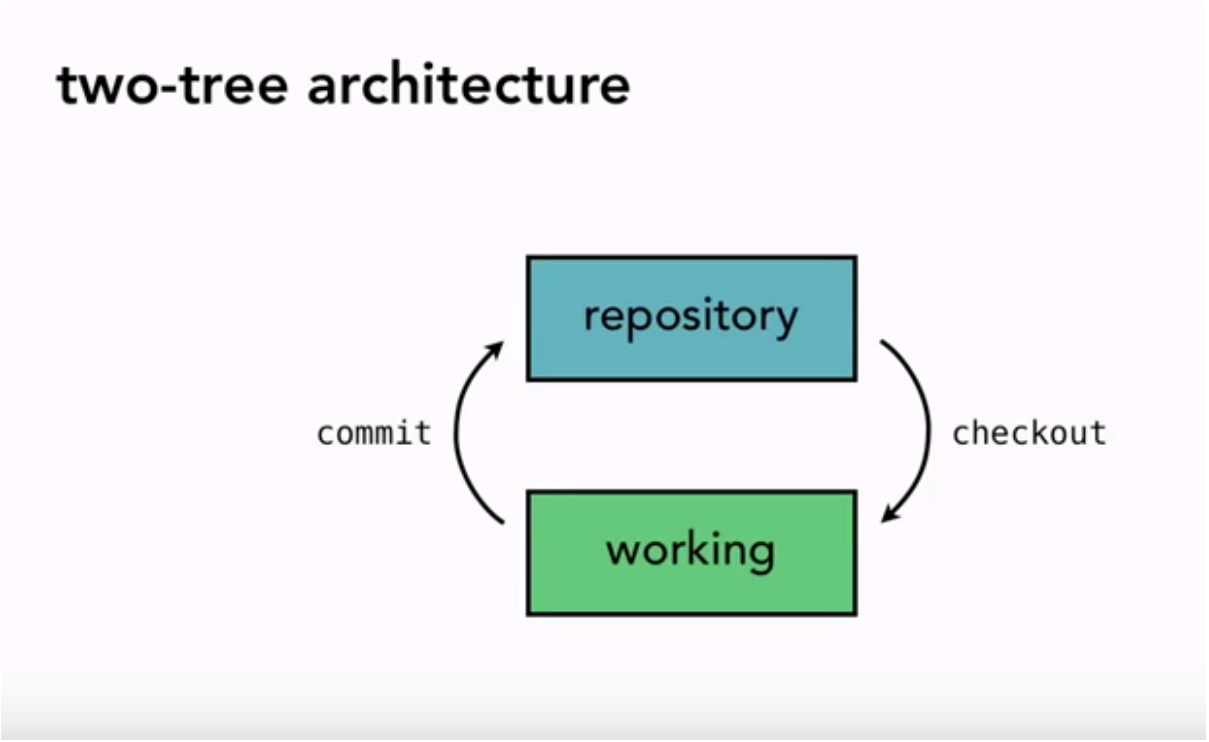
Whenever you edit something, it is saved in a working copy and it is physically stored in a disk.

Repository [SERVER] is the place where all the versions of the files or commits, logs etc are stored. It is also saved in a disk and has its own set of files.

You cannot however change or get the files in a repository directly, in able to retrieve a specific file from there, you have to checkout

Checking-out is the process of getting files from repository to your working copy. This is because you can only edit files when it is on your working copy. When you are done editing the file, you will save it back to the repository by committing it, so that it can be used by other developers.

Committing is the process of putting back the files from working copy to repository.



Hence, this architecture is called **2 Tree Architecture.**

Because you have two tree in there **Working Copy** and **Repository**.

The famous VCS with this kind of architecture is Subversion or SVN.

How the Distributed DVCS works

==========

Unusually, a DVCS works by having three places to store things:

1. **Working Copy**
2. **Staging**
3. **Repository**

As Git uses a Distributed version control system, So let’s talk about Git which will give you an understanding of DVCS.

Git was initially designed and developed by Linus Torvalds in 2005 for Linux kernel development. Git is an Open Source tool.

History

========

For developing and maintaining Linux Kernel, Linus Torvalds used **BitKeeper** which is also one of the VCS, and was open source till 2004.

So instead of depending on other tools, they developed their own VCS.

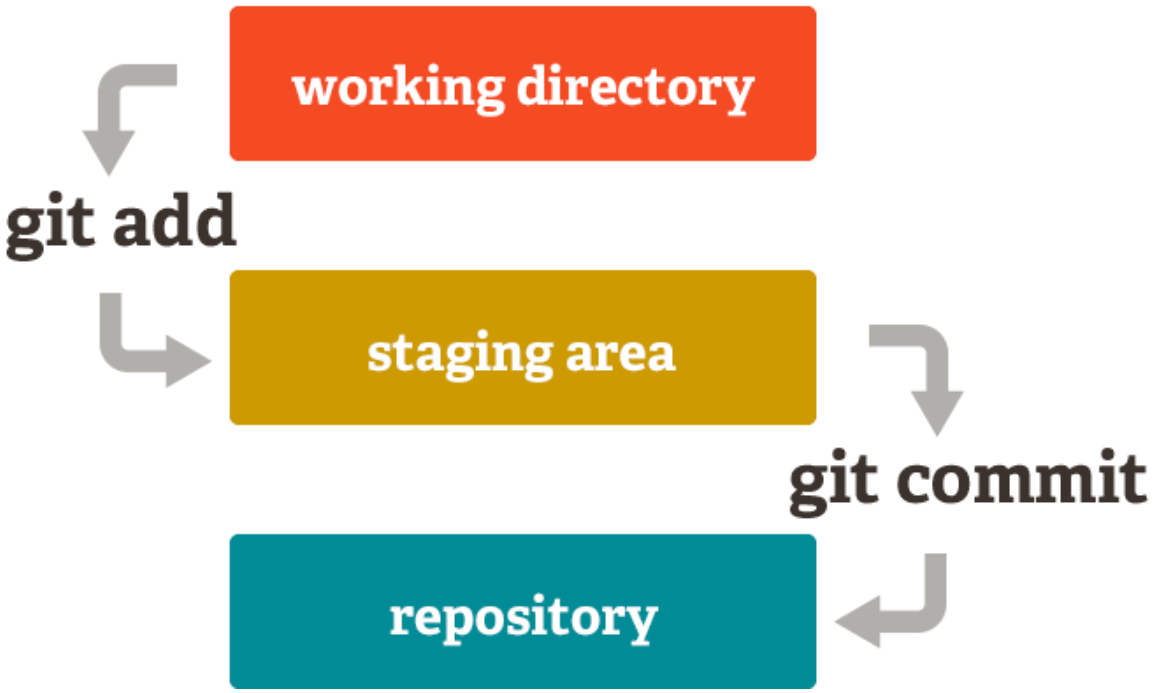
Just see the wiki of Git.

Git Architecture

==========

Git uses three tree architecture.

Well interestingly Git has the **Working Copy** and **Repository** as well but it has added an extra tree **Staging** in between:



As you can see above, there is a new tree called **Staging**.

What is this for ?

This is one of the fundamental differences of Git that sets it apart from other VCS, this **Staging tree** (usually termed as **Staging area**) is a place where you prepare all the things that you are going to commit.

In Git, you don't move things directly from your working copy to the repository, you have to stage them first, one of the main benefits of this is, **to break up your working changes into smaller, self-contained pieces.**

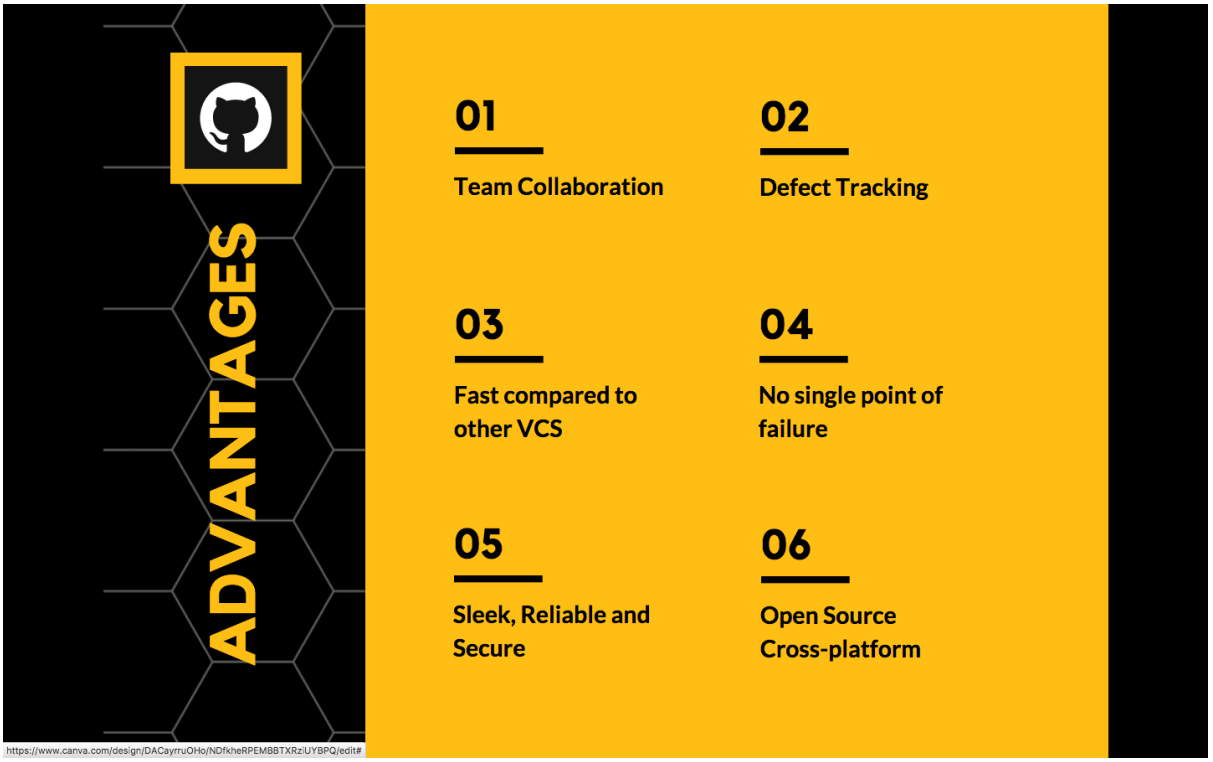
To stage a file is to prepare it for a commit.

Staging allows you finer control over exactly how you want to approach version control.

Advantages Of Git

==========

Installing Git



Git works on most OS: Linux, Windows, Solaris and MAC.

==========

- Download Git {git website}

To check if git is available or not use:

 # rpm -qa | grep git

 # sudo yum install git

 # git --version

Setting the Configuration

==========

 # git config --global user.name "Sai Krishna"

 # git config --global user.email "info@gmail.com"

git config --list

NOTE :: The above info is not the authentication information.

What is the need of git config

==========

When we setup git and before adding a bunch of files, We need to fill up username & email and it's basically a git way of creating an account.

Working with Git

==========

Getting a Git Repository

 # mkdir website

Initializing a repository into directory, to initialize git we use:

 # git init

The purpose of Git is to manage a project, or a set of files, as they change over time. Git stores this information in a data structure called a repository.

git init is only for when you create your own new repository from scratch.

It turns a directory into an empty git repository.

Let's have some configuration set:

git config --global user.name "sai"

git config --global user.email "sai@gmail.com"

Take e-commerce sites as an example.

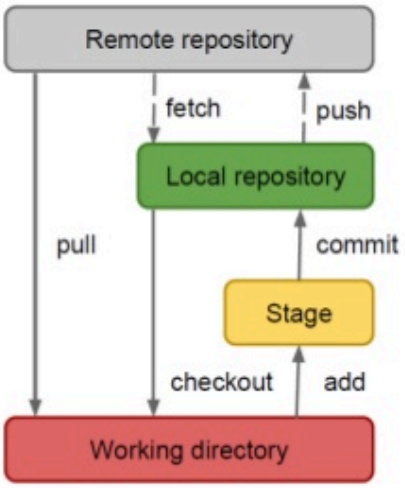
Basic Git Workflow

==========

1. You modify files in working directory
2. You stage files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot to your git repository.

Understanding of Workflow

- Obtain a repository
  - *git init* or *git clone*
- Make some changes
- Stage your changes
  - *git add*
- Commit changes to the local repository
  - git commit –m “My message”
- Push changes to remote
  - git push remotename remotebranch



mkdir website

git init

git status {Branches will talk later}

vi index.html

{put some tags <html><title><h1><body> just structure}

git status

git add index.html {staged the changes}

git commit -m “Message” {moves file from staging area to local repo}

git status

You can skip the staging area by # git commit -a -m “New Changes”

Commit History - How many Commits have happened ??

==========

To see what commits have been done so far we use a command:

 # git log

It gives commit history basically commit number, author info, date and commit message.

Want to see what happend at this commit, zoom in info we use:

 # git show <commit number>

Let’s understand this **commit number**

This is a sha1 value randomly generated number which is a 40 character hexadecimal number which will be unique.

Let’s change the title in index.html and go with

git add status commit

git log {gives the latest commit on top and old will get down}

Git diff

========

Let’s see, the diff command gives the difference b/w two commits.

 # git diff xxxxxx..xxxxxx

You can get diff b/w any sha's like sha1..sha20.

Now our log started increasing, like this the changes keep on adding file is one but there are different versions of this file.

git log --since YYYY-MM-DD

git log --author saiurakrishna

git log --oneline

Git Branching

==========

In a collaborative environment, it is common for several developers to share and work on the same source code.

Some developers will be fixing bugs while others would be implementing new features.

Therefore, there has got to be a manageable way to **maintain different versions** of the same code base.

This is where the branch function comes to the rescue. **Branch allows** each developer **to branch out from the original code base and isolate their work from others.** Another good thing about branch is that **it helps Git to easily merge** the versions later on.

It is a common practice to create a new branch for each task (eg. bug fixing, new features etc.)

Branching means you diverge from the main line(master-working copy of application) of development and continue to do work without messing with that main line.

Basically, you have your master branch and you don't want to mess anything up on that branch.

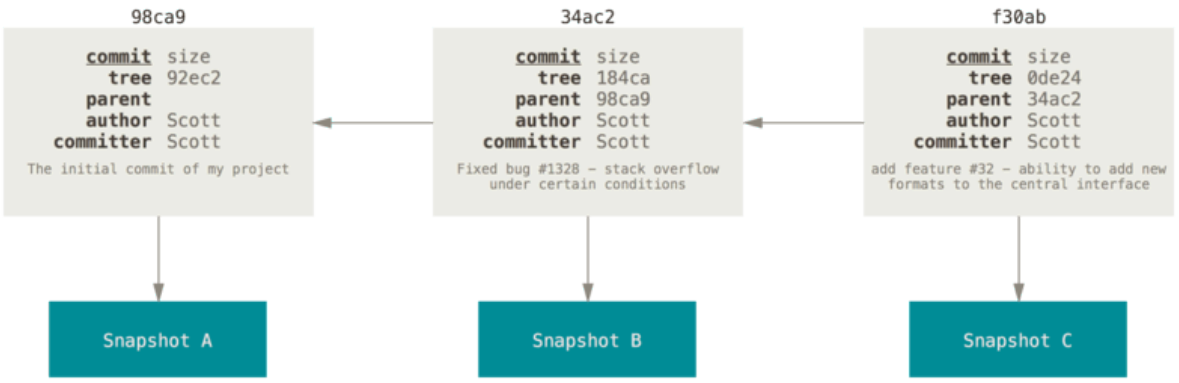
In many VCS tools, **branching** is an **expensive process**, often requiring you to create a new copy of your source code directory, which can take a long time for large projects.

Some people refer to **Git's branching model as its "killer feature"** and it certainly sets Git apart in the VCS community.

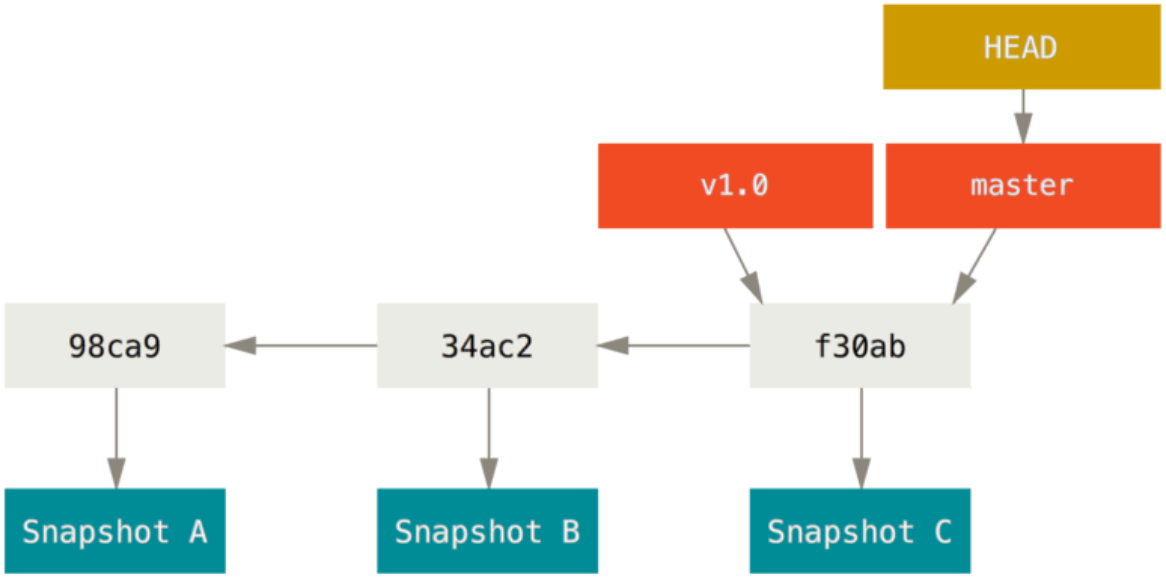
Why is it so special?

The way Git branches is incredibly **lightweight**, making branching operations nearly **instantaneous**, and switching back and forth between branches generally just as fast.

When we make a commits, this is how git stores them.



A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master. As you start making commits, you're given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically.



What happens if you create a new branch? Well, doing so creates a new pointer for you to move around.

Let’s say you create a new branch called testing.

 # git branch testing

This creates a new pointer to the same commit you’re currently on.

Two branches pointing into the same series of commits.

How does Git know what branch you’re currently on?

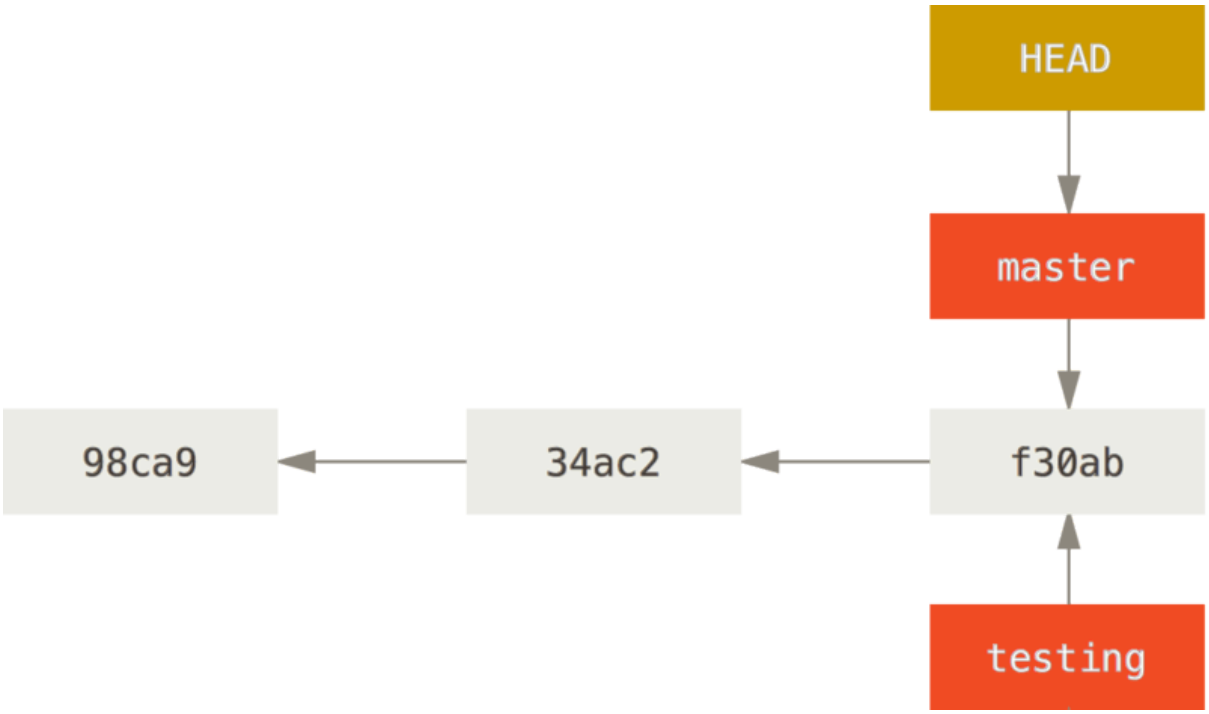
It keeps a **special pointer** called **HEAD**.

HEAD is a pointer to the latest commit id and is **always moving**, not stable.

 # git show HEAD

In Git, this is a pointer to the local branch you're currently on.

In this case, you're still on the master. The git branch command only created a new branch — it didn't switch to that branch.



This command shows you **where the branch pointers are pointing:**

```
# git log --oneline --decorate
```

You can see the "**master**" and "**testing**" branches that are right there next to the f30ab commit.

To **switch to an existing branch**, you run the git checkout command.

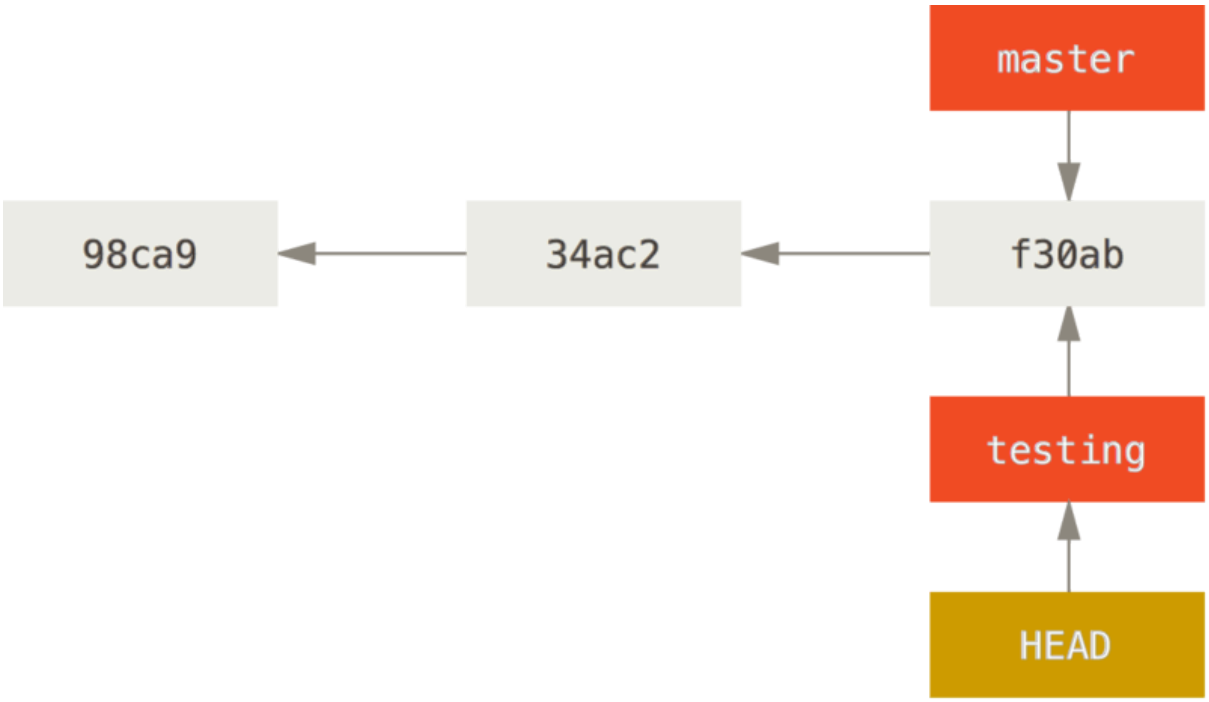
```
# git checkout testing
```

This **moves HEAD** to point to the **testing** branch.

What is the significance of that?

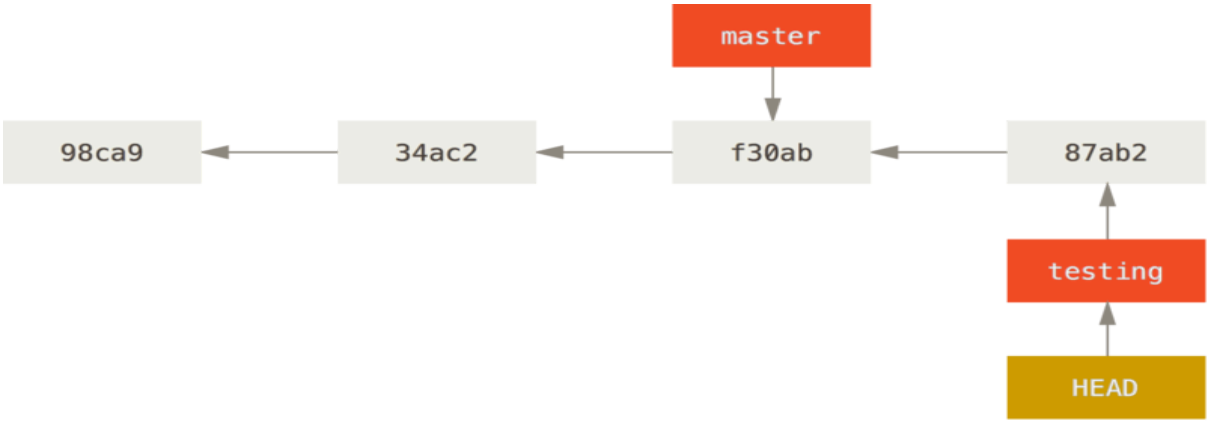
Well, let's do another commit:

vim test.rb



git commit -a -m 'made a change'

 # git log --oneline --decorate

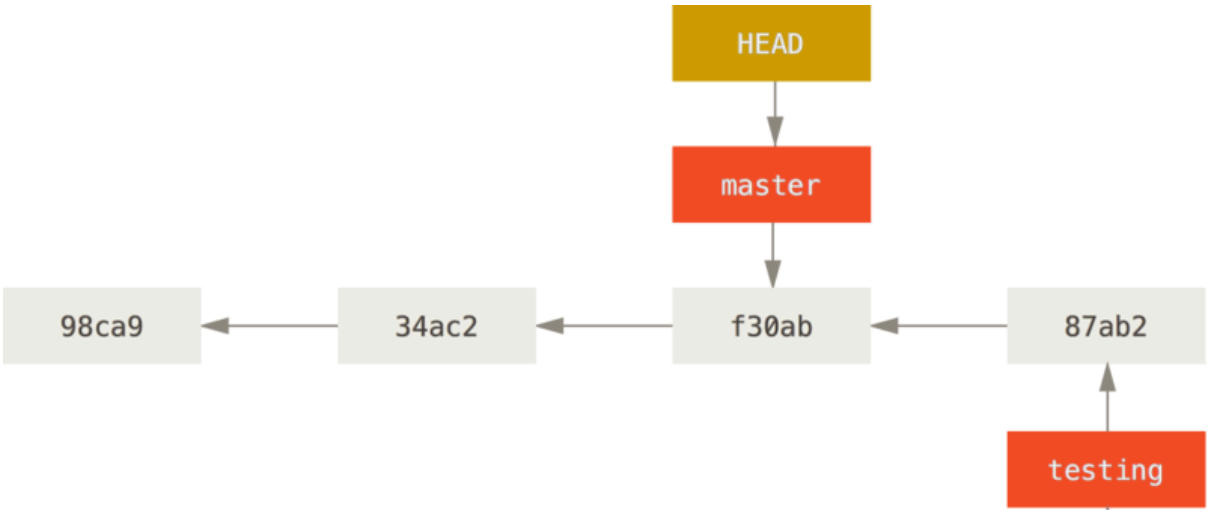


The HEAD branch moves forward when a commit is made.

This is interesting because now your **testing** branch has **moved forward**, but your master branch still points to the commit you were on when you ran git checkout to switch branches.

Let's switch back to the master branch:

 # git checkout master



HEAD moves when you checkout.

That command(git checkout master) did two things. It **moved** the **HEAD** pointer back to point to the **master branch**, and it **reverted the files** in your working directory **back to the snapshot** that **master points to**.

Let’s make a few changes and commit again:

vim test.rb

git commit -a -m 'made other changes’

Now your project history has diverged.

You created and switched to a branch, did some work on it, and then switched back to your main branch and did other work.

Both of those changes are isolated in separate branches: you can switch back and forth between the branches and merge them together when you’re ready.

And you did all that with simple **branch**, **checkout** and **commit** commands.

Divergent history, You can also see this easily with the git log command.

 # git log --oneline --decorate --graph --all

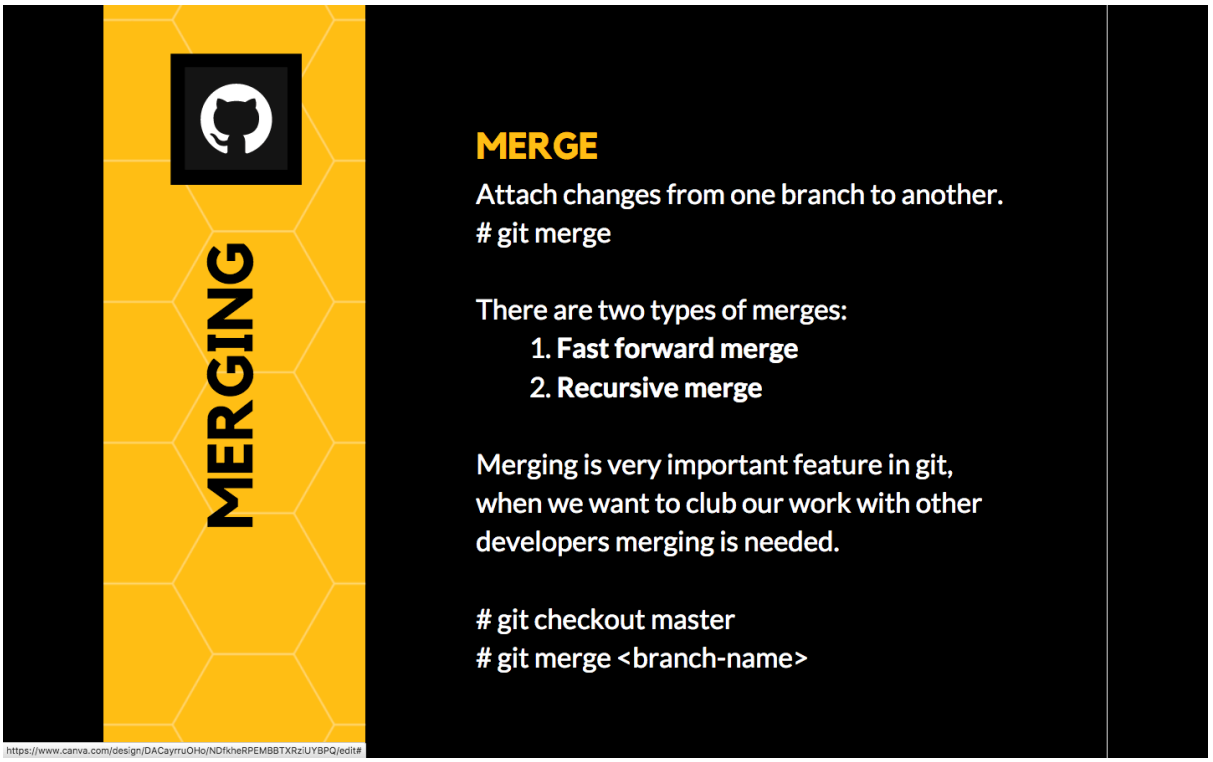
To see all available branches

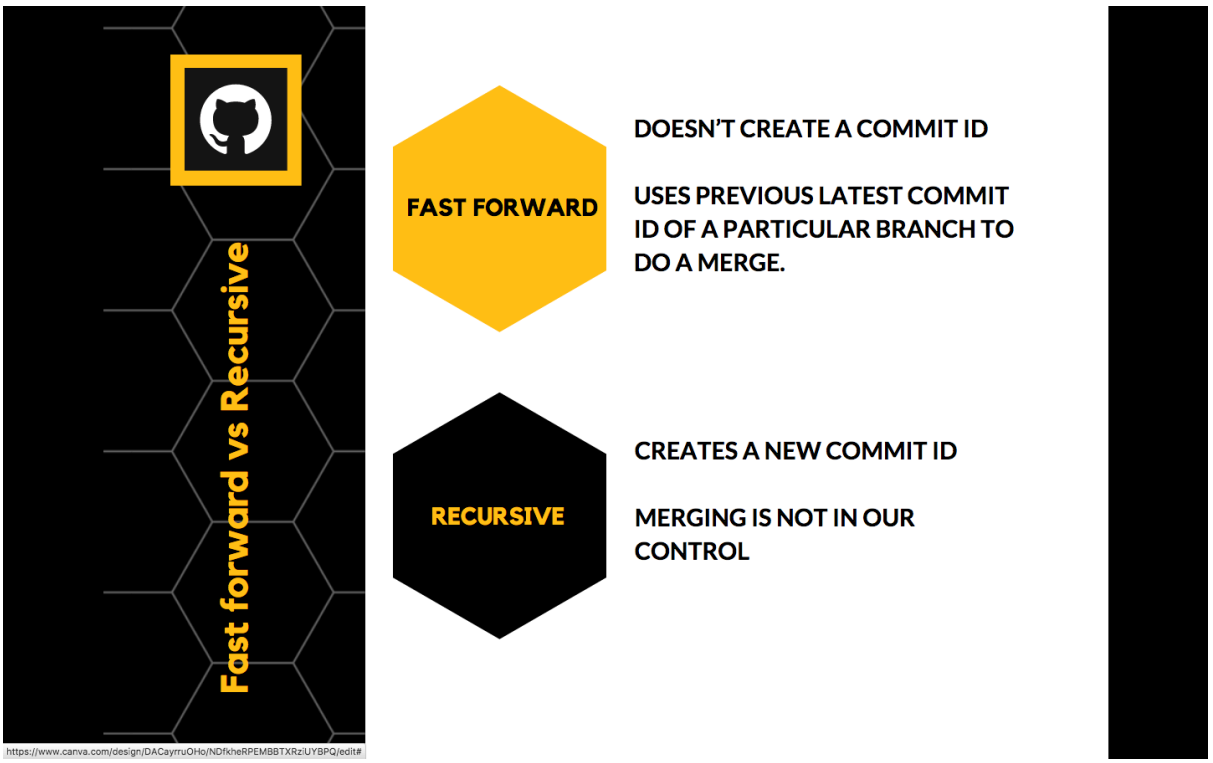
 # git branch -a

 # git reflog {short logs}

Merging

==========





Git Merge Conflict

==========

A merge conflict happens when two branches both modify the same region of a file and are subsequently merged.

Git doesn't know which of the changes to keep, and thus needs human intervention to resolve the conflict.

Showing the example **R&D**, **Training,** and **Consulting**.

Automatic merge failed

Git fetch vs Git Pull

git fetch and git pull are both commands used in Git to update a local repository with changes made to a remote repository. However, they have different functionality:

git fetch downloads the latest changes from a remote repository but does not integrate them into the current branch. It only updates the remote-tracking branches (e.g. origin/master) that are used to keep track of changes in the remote repository. It does not modify the working directory or the local branch you are currently on.

git pull is a combination of two Git commands: git fetch followed by git merge. It downloads the latest changes from the remote repository and integrates them into

the current branch. It updates both the remote-tracking branches and the local branch you are currently on. If there are conflicts between the changes made locally and remotely, Git will prompt you to resolve them.

Git Reset:

git reset is used when we want to unstage a file and bring our changes back to the working directory. git reset can also be used to remove commits from the local repository.

$ *git reset HEAD <filename>*

There are different ways in which git reset can actually keep your changes.

- **git reset –soft HEAD~1** – This command will remove the commit but would not unstage a file. Our changes still would be in the staging area.
- **git reset –mixed HEAD~1** or git reset HEAD~1 – This is the default command that we have used in the above example which removes the commit as well as unstages the file and our changes are stored in the working directory.
- **git reset –hard HEAD~1** – This command removes the commit as well as the changes from your working directory. This command can also be called a destructive command as we would not be able to get back the changes so be careful while using this command.

Git Revert:

git revert is used to remove the commits from the remote repository.

We could have used the git reset command but that would have deleted the commit just from the local repository and not the remote repository. If we do this then we would get conflict that the remote commit is not present locally. So, we do not use git reset here. The best we can use here is git revert.

$ git revert <commit id>

Git Stash: The git stash command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy. For example:

$ git stash
You can reapply previously stashed changes with **git stash pop**

Merge VS Rebase:

https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Git Cherry-pick:
https://www.atlassian.com/git/tutorials/cherry-pick

Git tag: https://www.atlassian.com/git/tutorials/inspecting-a-repository/git-tag

To fix git pull issue:
https://stackoverflow.com/questions/13106179/error-fatal-not-possible-to-fast-forward-aborting

Ref blog: https://www.atlassian.com/git/glossary

Maven

MAVEN

A VCS plays a vital role in any kind of organization, the entire software industry is built around code.

What are we doing with this code ??

- Are we seeing the code when we open the application ?? NO
- Are we seeing the code when we open the app in the browser ?? NO

So we are seeing the executable format of the code, that is called build result of the code.

What is build ??
==========
Build is the end result of your source code.

Build tool is nothing but, it takes your source code and converts it into executable format.

Build
====
The term build may refer to the process by which source code is converted into a stand-alone form that can be run on a computer.

One of the most important steps of a software build is the compilation process, where source code files are converted into executable code.

The process of building software is usually managed by a build tool i.e, maven.

Builds are created when a certain point in development has been reached or the code has been ready for implementation, either for testing or outright release.

***Build**: Developers write the code, compile it, compress the code and save it in a compressed folder. This is called Build.*

***Release**: As a part of this, starting from System study, developing the software and testing it for multiple cycles and deploying the same in the production server. In short, one release consists of multiple builds.*

Maven Objectives

==========

- A comprehensive model for projects which is reusable, maintainable, and easier to understand.
- plugins

Convention over configuration

==========

Maven uses *Convention* over *Configuration* which means developers are not required to create the build process themselves. Developers do not have to mention each and every configuration detail.

Earlier to maven we had ANT, which was pretty famous before maven.

Disadvantages of ANT

==========

ANT - Ant scripts need to be written for building
[build.xml need to tell src & classes]
ANT - There is no dependency management
ANT - No project structure is defined

Advantages of Maven

==========

No script is required for building [automatically generated - pom.xml]
Dependencies are automatically downloaded
Project structure is generated by maven
Documentation for project can be generated

Maven is also called a project **management tool**, the reason is earlier when we used to create projects and we used to create the directory structure and all by yourself, but now maven will take care of that process.

MAVEN has the ability to create project structure.

Maven can generate documentation for the project.

Whenever I generate a project using maven I will get src and test all by default.

MAVEN FEATURES

==========

Dependency System

==========

Initially in any kind of a build, whenever a dependency is needed, if I'm using ANT I have to download the dependency then keep it in a place where ANT can be understood.

If i'm not giving the dependency manually my build will fail due to dependency issues.

Maven handles dependency in a beautiful manner, there is a place called MAVEN CENTRAL. Maven central is a centralized location where all the dependencies are stored over web/internet.

For example, I'm using a project and I'm having a dependency on junit. Whenever my build reaches a phase where it needs junit then it will download the dependencies automatically and it will store those dependencies in your machine. There is a directory called as **.m2** created in your machine, where all the dependencies are going to be saved. Next time when it comes across the same dependency, then it doesn't download it because it's already available in the .m2 directory.

Plugin Oriented

==========

Maven has so many plugins that i can integrate, i can integrate junit, jmeter, sonarqube, tomcat, cobertura and so many others.

IMPORTANT FILE IN MAVEN

==========

Projects in maven are defined by **POM** (Project Object Model) pom.xml.

Maven lifecycle phases
==========

What is the build life cycle?
The sequence of steps which is defined in order to execute the tasks and goals of any maven project is known as build lifecycle in maven.

The following are most common *default* lifecycle phases executed:

- **validate**: validate the project is correct and all necessary information[dependencies] are available and keep it in local repo
- **compile**: compile the source code of the project
- **test**: Execution of unit tests, test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **verify**: run any checks to verify the package is valid and meets quality criteria, keeps the HelloWorld.jar in .m2 local repo
- **install**: Deploy to local repo [.m2], install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects. This will push the libraries from .m2 to remote repo.

There are two other Maven lifecycles of note beyond the *default* list above. They are

- clean: cleans up artifacts created by prior builds
- site: generates site documentation for this project

These lifecycle phases are executed sequentially to complete the default life cycle.

POM { will be in XML format }
====

GAV

Maven uniquely identifies a project using:

- **groupID**: Usually it will be the domain name used in reverse format (going to be given by the project manager).
- **artifactID**: This should be the name of the artifact that is going to be generated
- **Version** : Version of the project, Format {Major}.{Minor}.{Maintenance} and add “-SNAPSHOT” to identify in development
  Version can be two things here, a SNAPSHOT and other is RELEASE.

Snapshot - whenever your project is in working condition i mean we are still working on it that would be a snapshot version, you can have multiple snapshots for one single project.
But there would be only one release for it, for example after my 20th snapshot we decided that 8th snapshot should goto release then will remove -SNAPSHOT for 8th.
In real time we will get the basic pom which is already written with groupid, artifactid and version, we can build up required plugins and dependencies.

Packaging
=========
Build type is identified by <packaging> element, this element will tell how to build the project.

Example packaging types: jar, war etc.

Archetype
=========
Maven archetypes are project templates which can be generated for you by Maven. In other words, when you are starting a new project you can generate a template for that project with Maven.
In Maven a template is called an *archetype*.
Each Maven archetype thus corresponds to a project template that Maven can generate.

Installation
==========

Maven is dependent on java as we are running java applications, so to have maven, we also need to have java in system.

Install java

=========

You need Java to install for Maven

Install Maven

==========

```
# yum -y install maven
```

Maven repository are of three types

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

For maven to download the required artifacts of the build and dependencies (jar files) and other plugins which are configured as part of any project, there should be a common place. This common shared area is called a Repository in maven.

Local

=====

The repository which resides in our local machine which is cached from the remote/central repository downloads and ready for the usage.

Central

======

This is the repository provided by the maven community. This repository contains a large set of commonly used/required libraries for any java project. Basically, internet connection is required if developers want to make use of this central repository. But, no configuration is required for accessing this central repository.
https://repo.maven.apache.org/maven2/

Remote

=======

These are custom repositories that are set up by organizations or individuals to host their own artifacts. Remote repositories can be public or private, and can be hosted

on a local or remote server. Developers can add remote repositories to their Maven build configuration to download dependencies that are not available in the Central Repository.

How does Maven search for Dependencies?

==========

Basically, when maven starts executing the build commands, maven starts for searching the dependencies as explained below :

- It scans through the local repositories for all the configured dependencies. If found, then it continues with the further execution. If the configured dependencies are not found in the local repository, then it scans through the central repository.

- If the specified dependencies are found in the central repository, then those dependencies are downloaded to the local repository for the future reference and usage. If not found, then maven starts scanning into the remote repositories.

- If no remote repository has been configured, then maven will throw an exception saying not able to find the dependencies & stops processing. If found, then those dependencies are downloaded to the local repository for the future reference and usage.

Setting up stand alone project

==========

```
# cd ~
# mvn archetype:generate          { generates project structure }
# we get some number like 1085 beside it we have 2xxx, which means maven
 currently supports 2xxx project structures, 1085 is like default project
# press enter
# choose a number :: 6 which means latest, so press enter
# groupId: com.digital.academy    { unique in world, generally domain }
# artifactId: project1            { project name }
# version: press enter            { snapshot - intermediate version }
# package: enter { package name is java package }
# Press: y
```

Now maven has successfully created a project structure for you:

tree -a project1

We have pom.xml which contains all the definitions for your project generated, this is the main file of the project.

ls -l ~/.m2/repository

mvn validate { whatever in the pom.xml is correct or not }

Let's make some mistakes and try to fail this phase,

mv pom.xml pom.xml.bk
mvn validate { build failure }
mv pom.xml.bk pom.xml
vi App.java { welcome to Devops }
mvn compile { after changing code we do compilation right }
{ this generates a new structure - # tree -a . with class files}
mvn test { test the application }
mvn package { generates the artifact - jar }

 # java -cp target/xxxx.jar groupid(com.digital.proj1).App

Plugins

=======

We saw maven is only performing phases like validate, compile, test, package, install, deploy but if you remember there is no execution of a jar file,

Can you see any of the phases running jar file, no right ??

Executing jar file is not part not the part of life cycle,
apart from the above phases such as validate, compile, test, package, install, deploy all the other come under <build> under <plugins> </plugins>

These plugins will define, other then regular maven lifecycle phases,

Let's take example, i want to run my jar file,
After </dependencies> in your pom.xml

After </dependencies> in your pom.xml add the following

```
<build>
<plugins>
<plugin>
<groupId>org.codehaus.mojo</groupId>
  <artifactId>exec-maven-plugin</artifactId>
  <version>1.2.1</version>
  <configuration>
    <mainClass>com.sample.project.App</mainClass>
       <arguments>
            <argument>-jar</argument>
            <argument>target/*.jar</argument>
        </arguments>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

In pom.xml after </dependencies> add <build> <plugins> <plugin>

We need to run

 # mvn exec:java

WEB APP SETUP

Setting up web project

==========

mvn archetype:generate | grep maven-archetype-webapp

Sample O/P:

2xxx: remote -> org.apache.maven.archetypes:maven-archetype-webapp (An archetype which contains a sample Maven Webapp project.

type the number you get

tree -a project/

mvn clean package

tree -a project

You can see the **war** generated under target directory

TOMCAT

Tomcat Installation [Binaries]

==========

Apache Tomcat is an open-source web server and servlet container developed by the Apache Software Foundation. It's primarily used to serve Java-based web applications. Unlike traditional web servers like Apache HTTP Server or Nginx, which serve static content like HTML pages, images, and videos, Tomcat is specifically designed to handle dynamic content, particularly Java Servlets and JavaServer Pages (JSPs).

Key Concepts:

1. **Servlet**:
    - A Servlet is a Java class that runs within a web server (like Tomcat) and handles client requests. It processes incoming HTTP requests, performs backend operations (like interacting with databases), and generates HTTP responses (like HTML pages or JSON data).
    - Servlets are a key component of Java EE (Enterprise Edition) and are used to build dynamic web applications.
2. **web.xml**:
    - The `web.xml` file is a deployment descriptor located in the `WEB-INF` directory of a Java web application. It is an XML file that configures various aspects of the web application.
    - It typically contains:
        - **Servlet declarations**: Mapping specific URLs to servlets.

        - **Servlet initialization parameters**.
        - **Security configurations**: Defining roles, constraints, and authentication methods.
        - **Session management settings**.
    - While `web.xml` was traditionally essential for setting up servlets and other web components, modern Java EE frameworks like Spring Boot often use annotations to replace or simplify `web.xml` configurations. However, it remains an important part of many legacy and enterprise Java applications.

How It Works in Tomcat:

- **Tomcat** serves as the runtime environment for Java Servlets. When a client (e.g., a web browser) sends a request to a Java-based web application deployed on Tomcat, Tomcat forwards this request to the appropriate servlet based on the mappings defined in `web.xml` (or annotations in the servlet classes).
- The servlet processes the request, performs any necessary business logic (like querying a database or processing form data), and sends back a response to the client, typically in the form of an HTML page, XML, or JSON.

Tomcat essentially bridges the gap between the client and the server-side Java code, allowing for the creation and deployment of dynamic, interactive web applications.

Google tomcat 7 download

wget https://archive.apache.org/dist/tomcat/tomcat-7/v7.0.105/bin/apache-tomcat-7.0.105.tar.gz

tar xvf apache-tomcat-7.0.105.tar.gz

cd apache-tomcat-7.0.105/bin

./startup.sh

Check for port to be opened in firewall
netstat -ntpl { # sudo yum -y install net-tools }

ps -ef | grep tomcat

Goto http://ip-address/8080 {click cancel and change tomcat-users.xml file}

<role rolename="manager-gui"/>

<user username="tomcat" password="tomcat" roles="manager-gui"/>

<user username="tomcat1" password="tomcat1" roles="manager-script"/>

Change the port number in server.xml
cd apache-tomcat-7.0.81/bin

./shutdown.sh

Copy the generated war file to webapps dir of tomcat

Refresh the tomcat page

Deploy to tomcat maven tomcat plugin

==========

Add Manager-Script Role

==========

Add new role under conf/tomcat-users.xml

vi **conf/tomcat-users.xml**

*<user username="tomcat1" password="tomcat1" roles="**manager-script**"/>*

Add Tomcat 7 Maven Plugin

==========

vi pom.xml

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<url>http://174.129.182.181:8080/m

anager/text</url>

<server>TomcatServer</server>

<path>/demoapp</path>

</configuration>

</plugin>

Add Maven-Tomcat Authentication

==========

vi ~/.m2/settings.xml

<settings>

<servers>

<server>

<id>TomcatServer</id>

```
            <username>tomcat1</username>
            <password>tomcat1</password>
        </server>
    </servers>
</settings>
```

 # mvn tomcat7:deploy

 # mvn tomcat7:undeploy

 # mvn tomcat7:redeploy