

Yash Rajput

Weekly Documentation

Product Engineer

Servify

Table of contents

- Linux
 - Linux Commands
 - Linux File System
 - SSH Handshaking
- GIT
 - Git Commands
 - Git SSH-Key Setup
 - Git Workflow
 - Git Collaboration
 - Git Collaboration from remote
 - Git Conflict resolve
 - Git Conflict resolve from remote
- Web Development Building Blocks
 - IP Addresses
 - Ports

- DNS
- HTTP
 - HTTP Verbs
 - HTTP Headers
- HTTPS
- Database
- Software Licences
- Cloud Providers
- Javascript
 - Four Pillars
 - Variables
 - Tech Debt
 - Operators
 - Statements
 - ES6
 - Arrays and Objects

WEEK #1

LINUX

- A Linux Distribution is composed of a kernel and additional libraries and software.
- The kernel acts as an interface for communication between the software and the hardware.
- Linux provides a shell as a CLI which provides us with an interface to use the operating system.
- Advantages of using Linux:
 - Free Software licensing
 - Open-source
 - Secure
 - Multiple distributions(2000+ distribution):
 - Debian
 - Ubuntu => Most popular among consumers
 - Linux Mint
 - RedHat => Commercial version for Enterprise/Business
 - CentOS => Community version for Enterprise/Business
 - Fedora => Community based
 - Kali Linux => Security version
 - Android => Developed by Google
 - ChromeOS => Developed by Google

Linux Commands

Command		Description
cd		Change directory
	cd "Directory" name	Change directory from current to another
	cd, cd~	Change to user home directory
	cd -	Changes to the previous directory
	cd /	Changes from current to root directory
	cd ..	Moves Directory one step back
pwd	pwd	Prints current/working directory
ls	ls <directory-name>	List all files/directories present inside the current directory
	ls -l	List all the file details in detailed format
	ls -a	List all hidden files and folders
	ls -h	Lists the memory of files in a human-readable format

	ls -t	Lists the files/directories in ascending order
	ls -r	List the files/directories in reverse order
	ls -f	Allows us to recognize directories by adding ' / '
	ls -lahtrf	All the above can be used together all at once
man	man <command-name>	Provides us with a detailed description of the command
who	who	Lists all the users who are currently logged in
whoami	whoami	Displays the current user
touch	touch "filename"	Create a new filea .
history	history	Lists history commands that we have executed or used
clear	clear, cmd + l	Clears the screen
ping	ping <hostname>, <IP>	Used check the internet speed and connectivity
echo	echo \$PATH	Shows path variable
ps	ps	It shows the running applications

cp	cp <source-file> <new-file>	Makes a copy of a file
mkdir	mkdir <directory-name>	Creates a new directory.
rm	rm 'filename'	Remove the file
	rm -i	Displays a prompt before removing
	rm -r	Removes recursively
	rmdir	It only removes empty directories
	rm -rf	It removes both files and directories along with hidden files and directories
	rm -rf*	It removes both files and directories
mv	mv <source-file> <target-directory>	Moves file to specified directory
	mv <file-name><new-name>	Renames file
ifconfig	ifconfig	Gives the network information and addresses.
chmod		Used to change the read, write and execute permissions of files and directories
chown		Used to change or transfer the ownership of a file to specified username

passwd		Used to change password
exit	exit	Exits from the terminal
cat	cat <file-name>	Displays the content of the file
	cat > <file-name>	Adds content to the files or overwrites the previous content
	cat >> <file-name>	Updates or appends previous content of the file
less	less 'filename	It basically gives more information.
more	more 'filename'	It gives less data.
top	top	Gives us the system information.
ssh	ssh username@host-ip-address	Establishes a secure connection between two networks or systems

Linux File System

- /boot : It contains the kernel
- /bin : User level binary files are stored in bin directory
- /sbin : System level binary files are stored and accessed by system admins
- /home : New users are created in home directory
- /var : Variable files are stored in this directory
- /usr : It contains user system resources
- /root : Root users are created inside this directory
- /tmp : In this directory temporary files are stored
- /etc : System configuration can be found here
- /lib : It has system libraries
- /mnt : Can be used to mount CD ROMS. Obsolete path now
- /dev : Contains memory files stored devices connected to the system
- /opt : Used to install software packages

SSH Handshaking

- SSH(Secure Shell) is used to create a secure connection between local and remote network
- Typically used for secure remote login and workflow
- Command syntax :
 - `ssh <username>@<host-ip>`
 - Example: `ssh yashraj@13.127.234.41`
`password: <your-password>`
- We use SSH handshaking to enable passwordless login to the remote server
- SSH handshaking uses keys(Public key and Private key) to establish passwordless connection
- Advantages:
 - Secure passwordless login
 - Not needed to add password every time we login in to the remote server
 - Saves time and makes workflow better
 - We can create
- Following are the steps we follow for SSH Handshaking:
 - - Step 1: Creating keys on the local machine:
 - Go to the user's home directory and create key using keygen command:
 - `Ssh-keygen`
 - It'll prompt for a name for the key, you can put in suitable key name and hit return key couple of times and the key will be generated inside ssh directory

Command	Syntax	Description
cd	cd	Switches to the user's home directory
pwd	pwd	Shows current directory

ls -la	ls -la	List all hidden files and directory
ssh-keygen	ssh-keygen	Creates a unique key

→ Step 2: Copy the Public key from .ssh directory in the local machine

- Navigate to the .ssh directory and look for the key that name that you provided while generating the key
- There would be 2 key with the same name : Public key(.pub extension) and Private key
- Copy the key with extension .pub

Command	Syntax	Description
cd	cd	Switch to user home
	cd .ssh	Navigate to .ssh
ls	ls -la	List all hidden files/directories
cat	cat <public-keyname>	View the key and copy it

→ Step 3: Login into the server and create ssh directory and authorized_keys file is it doesn't exist

Command	Syntax	Description
ssh	ssh <username>@<host-ip>	Logins into the remote server using username hostname and password
cd	cd	switches to users home directory
mkdir	mkdir .ssh	Creates a .ssh directory

cd	cd .ssh	Switches to .ssh directory
touch	touch <authorized-keys>	Create a file authorized-keys if it doesn't exist
cat	cat > <authorized-keys>	Paste the public key

→ Step 4: Add the key in the local machine and establishing connection to the remote server without password

- ◆ Exit back to the local machine from the remote server
- ◆ Add the key to the local machine using ssh-add command
- ◆ Use ssh command to establish connection with the remote server
- ◆ If followed the steps properly you'd be able to login with the remote without any password

Command	Syntax	Description
exit	exit	Exits from remote to local machine
cd	cd .ssh	Navigate to .ssh directory
ssh-add	Ssh-add <key-name>	Adds the key to local machine key list
ssh	ssh <username>@<host-ip>	Connects to remote without using password

→ Miscellaneous:

ssh-add -D	Revokes all the keys in the system
ssh-add -l	Lists all the keys in the local machine

GIT

- Git is a version control distribution system that helps developers to collaborate and work on a project simultaneously

Git commands

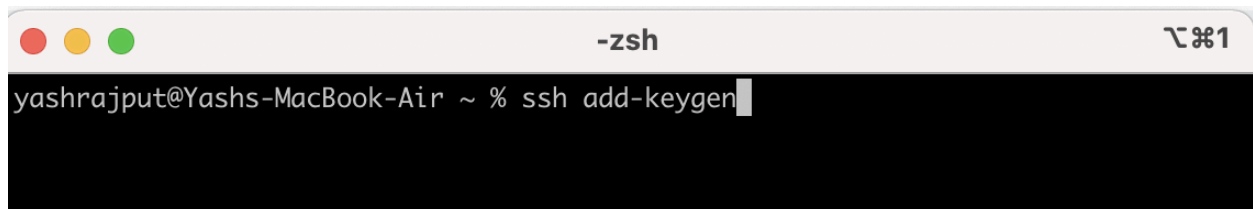
Commads		Description
git config	git config	Show who you are
	git config user.name	Configure the git author name
	git config user.email	Configure the git author email
git init		Create a new local repository
git clone <remote-url>		Clone a repository from existing url
git add	git add <file-name>	Adds a file to staging area
	git add *	Adds one or more to staging area
	git add .	Adds all files and directory to staging area
git commit	git commit -m"message"	Records or snapshots the file in the version history
git diff		Preview changes before and after the commit
git status		List the files we have changed and those that we still need to add or commit
git log		Used to list the version history for current branch
git show		Shows the metadata and content changes of specific commit
git branch	git branch	Lists all the local branches in the current repository
	git branch <branch-name>	Creates a new branch

	git branch -d <branch-name> git branch <branch-name> -D	Deletes the specified branch
git checkout	git checkout <branch-name>	Switches to the specified branch
	git checkout -b <branch-name>	Creates a new branch and switches to that branch
	git checkout -- <file-name>	Reverts changes to last commit already present in the commit history
git switch	git switch <branch-name>	Switches to the specified branch
	git checkout -c <branch-name>	Creates a new branch and switches to that branch
git merge		Merges specified branch's history into current branch
git push		Sends committed changes of local repository to the remote repository
git pull		Fetches and merges changes on the remote server to the working dir
git stash	git stash <file-name>	Take changes and stores locally in the system
git stash list		Displays list of stash
git stash pop		Brings back the changes

Git SSH-Key Setup

- SSH key setup for git enables us to login to the git server without using password but ssh handshaking instead

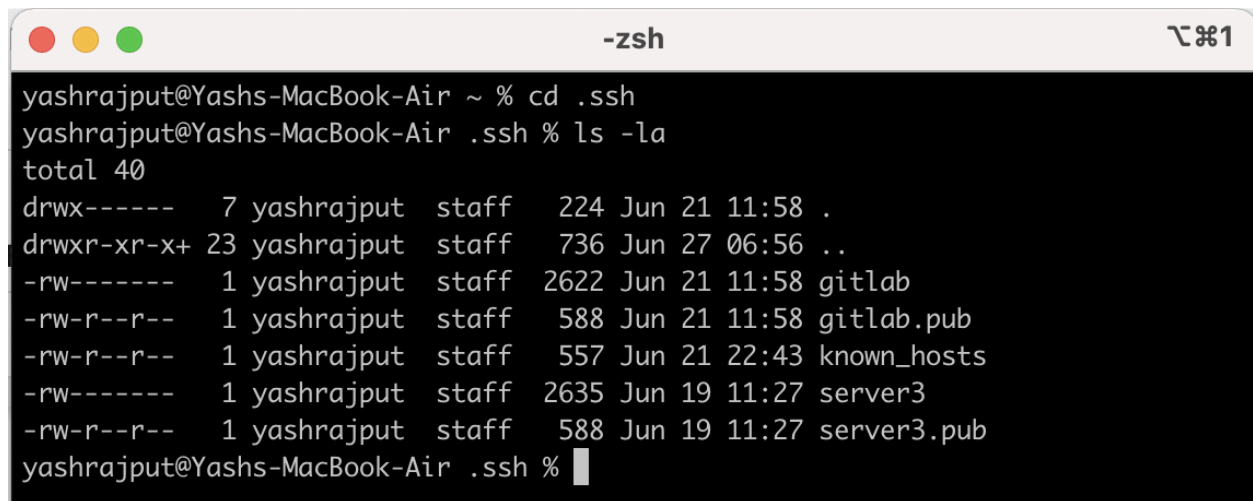
→ Step 1: Generate key



```
yashrajput@Yashs-MacBook-Air ~ % ssh add-keygen
```

A terminal window titled "-zsh" with a window icon on the left and a zoom icon on the right. The prompt is "yashrajput@Yashs-MacBook-Air ~ %". The command "ssh add-keygen" is entered, and a cursor is visible at the end of the line.

→ Step 2: Navigate to .ssh directory



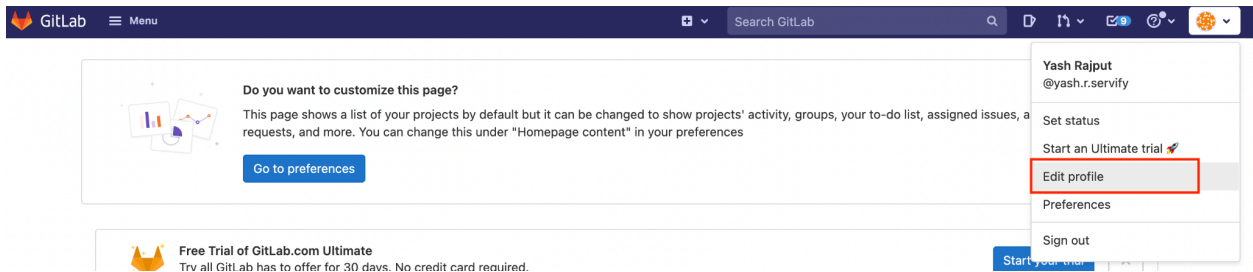
```
yashrajput@Yashs-MacBook-Air ~ % cd .ssh
yashrajput@Yashs-MacBook-Air .ssh % ls -la
total 40
drwx-----  7 yashrajput  staff   224 Jun 21 11:58 .
drwxr-xr-x+ 23 yashrajput  staff   736 Jun 27 06:56 ..
-rw-----  1 yashrajput  staff  2622 Jun 21 11:58 gitlab
-rw-r--r--  1 yashrajput  staff   588 Jun 21 11:58 gitlab.pub
-rw-r--r--  1 yashrajput  staff   557 Jun 21 22:43 known_hosts
-rw-----  1 yashrajput  staff  2635 Jun 19 11:27 server3
-rw-r--r--  1 yashrajput  staff   588 Jun 19 11:27 server3.pub
yashrajput@Yashs-MacBook-Air .ssh %
```

A terminal window titled "-zsh" with a window icon on the left and a zoom icon on the right. The prompt is "yashrajput@Yashs-MacBook-Air ~ %". The command "cd .ssh" is entered, and the prompt changes to "yashrajput@Yashs-MacBook-Air .ssh %". The command "ls -la" is entered, and the output is displayed. The output shows the contents of the .ssh directory, including files like gitlab, gitlab.pub, known_hosts, server3, and server3.pub. A cursor is visible at the end of the last line of the output.

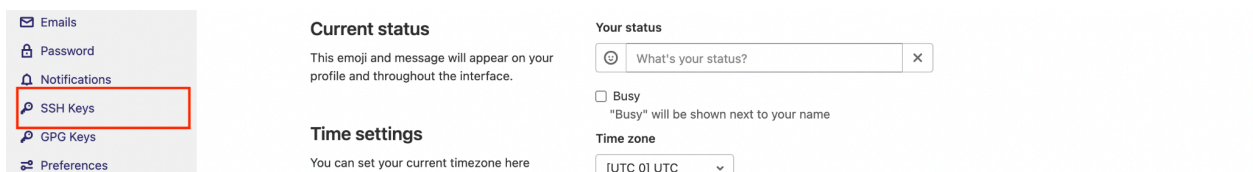
→ Step 3: Locate the generated public key and copy it

```
yashrajput@Yashs-MacBook-Air .ssh % cat gitlab.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCv9cKilSTnjsrBop4hLD46F/WDF5pCNednocsLSTV1
PB4ZAsGYtfo3K/jfmhyxcv4LxgyLRJnf07MLZgE7r9SX00uIxRExB0iH1z6Z5xeyb1ZVitQneLfzq53.
lNQ+yV83mCknrDQ0mwYWg6PUjGM9BNkXyZyXKw4uK0gIsnfWMRLZ3n3zAQzQ4urldwzc8kCBIMNujIoS
DFcvAU50308EXDZ3sEFnHt1ljf2cPq0mv7M4WYhbCbYW66AywxQNwLq4dmLJ4CGh0l fu5XS5tEqcErB0
IN8Rdug1TfhRSDEH8Cr4a71881dAcV9B2fg96lKAoWDu3WJKH7wqxc1H7uh1FutsmBafmG6K5bxpeJZZ
xkPncjQ1e70oetSpBUBWjsF91mC7e6aVLBIvYrEbhsiPRQH2vLac1uIMGunzoytkYfCRC8R/MBRgHbuA
+pQkI0rQU9MLcZCewlONY1UF27uDVeavJRtWiizxbhKohLG5wNxxwQ4p1GgYz3IHDi qbd/c= yashraj
put@Yashs-MacBook-Air.local
yashrajput@Yashs-MacBook-Air .ssh %
```

→ Step 4: Go to the remote server (Gitlab) and your profile



Step 5: Locate and navigate to SSH Keys



→ Step 6: Paste the copied public key and give the key a title

GitLab Menu

User Settings

Profile

Account

Billing

Applications

Chat

Access Tokens

Emails

Password

Notifications

SSH Keys

GPG Keys

Preferences

Active Sessions

Authentication log

Usage Quotas

User Settings > SSH Keys

Search settings

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to [generate one](#) or use an [existing key](#).

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_ed25519.pub' or '~/.ssh/id_rsa.pub' and begins with 'ssh-ed25519' or 'ssh-rsa'. Do not paste your private SSH key, as that can compromise your identity.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGCv9cKilSTnjsrBop4hLD46F/WDF5pCNednocslSTVl
PB4ZAsGy1t93K/lfmhyxcv4LxgvlRjnt07MIZqE7r9SXQQulxRExB0Ht1z6Z5keyb1ZVltQneLtzq5
3JlNQ+yV83mCknrDQOmWYVg6PUJGM9BNkXyZyXKw4UKOgismfWMRI23n3zAQzQ4unDwzc
8KCBIMNujloSDFcvaU5O308EXDZ3sEFnHt1lj2cPqOmV7M4WYhbCbYW66AywxQNwLq4dml
J4CGhOlFu5XS5tEqcEr8CIN8Rdug1TfRSDHEH8Cr4a71B81dAcV9B2fg96IKAoWdu3WJkH7wq
xc1H7uh1FutsmBafmG6K5bxpeJZ2xkPncjQ1e70oetSpBUBWjsF91mC7e6aVlBivYrEbhsiPRQH
2vLac1uIMGunzoytkYfRCRC8R/MBRgHbuA+pQk0rQU9MlcZCewlQNY1JUF27uDVeavJrtWli2x
bKohLG5wNxwvQ4plGqYz3IHdiqbd/c= yashrajput@Yashs-MacBook-Air.local
```

Title

yashrajput-ssh-key

Expires at

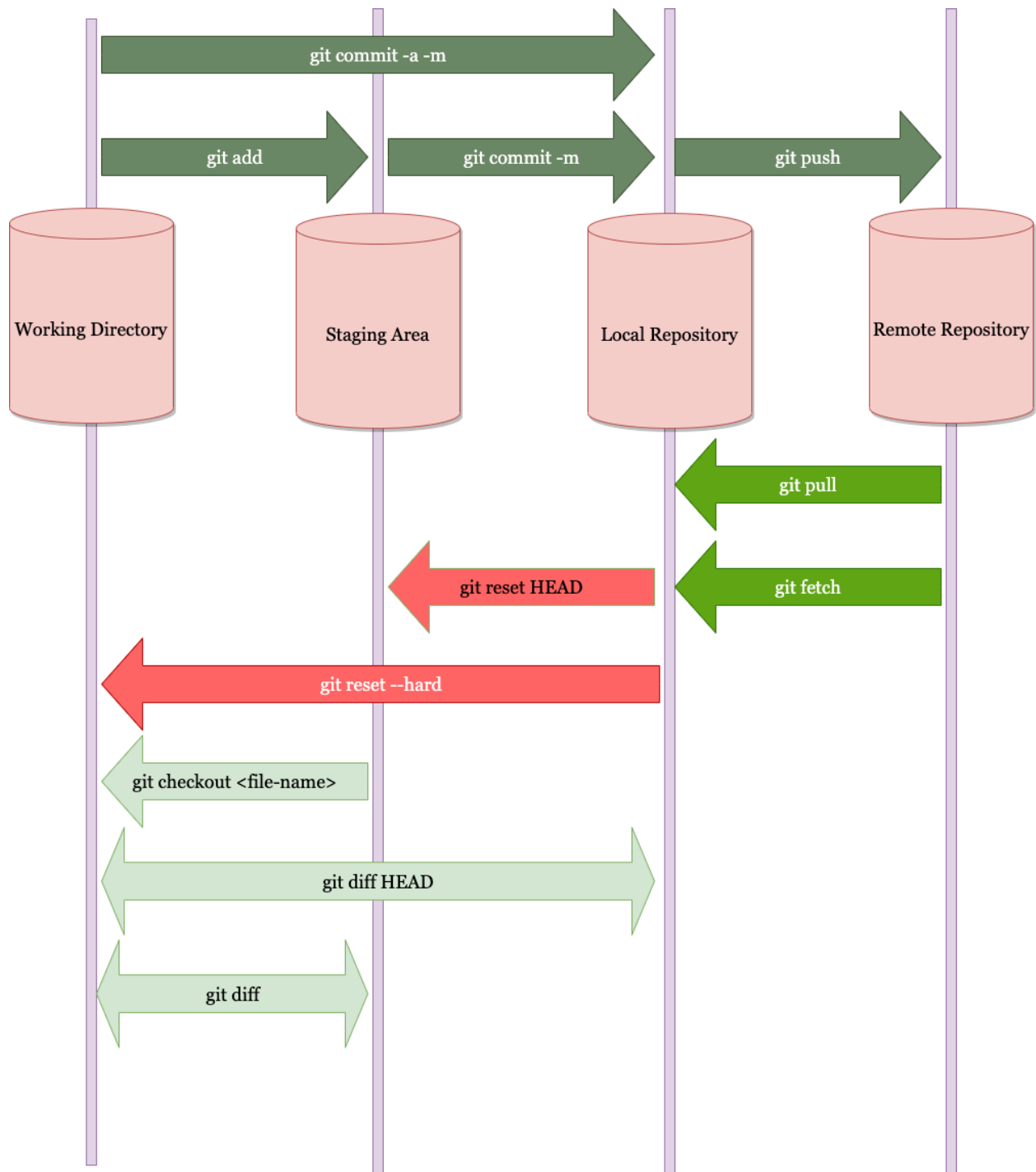
dd/mm/yyyy

Add key

→ Step 7: Add the key in the local machine using ssh-add

```
-zsh
yashrajput@Yashs-MacBook-Air .ssh % ssh-add gitlab.pub
```

Git Workflow



Git Normal Command Workflow

Git collaboration

Step 1: Create a new project on the remote, the creator will become an owner.

Step 2: Clone git repository from remote.

```
> git clone <remote-url>
```

Step 3: Create a new branch from master.

```
> git branch <branch-name>
```

Step 4: Create file inside the new branch and push the file to the newly created branch.(You can create multiple changes and push them to your branch)

```
> touch <file-name>
> git status
> git add <file-name>
> git commit -m "commit msg"
> git push <remote-name> <branch-name>
```

Step 5: Checkout to Remote branch and pull the changes

```
> git checkout <remote-branch>
> git pull <remote-name> <remote-branch>
```

Step 6: Checkout a new branch and merge changes from the remote branches with the new branch.

```
> git checkout <branch-name>
```

```
➤ git merge <remote-branch>
```

Step 7: Push the new branch to Remote.

```
➤ git push <remote-name> <branch-name>
```

NOTE

1. If the remote branch is ahead of the local branch then it will cause an error.
2. Conflict errors occur only when some command is missed or changes are not pulled properly.
3. Check if ssh-key is added if access is not given for pushing the changes to remote.

Git Collaboration From Remote

Step 1: Clone the repository OR if already cloned then only pull the changes from the repository. Create the new branch from the master branch. After creating branch checkout to your branch and creating a new file add content in the file push the file in your branch.

- git clone <remote-repository-url>
- git checkout -b <new-branch>
- touch <new-file>
- cat > <new-file>
- git add <new-file>
- git commit -m "commit-message"
- git push <remote-name> <new-branch>

Step 2: Switch back to the master branch and pull all the changes from master by using given command

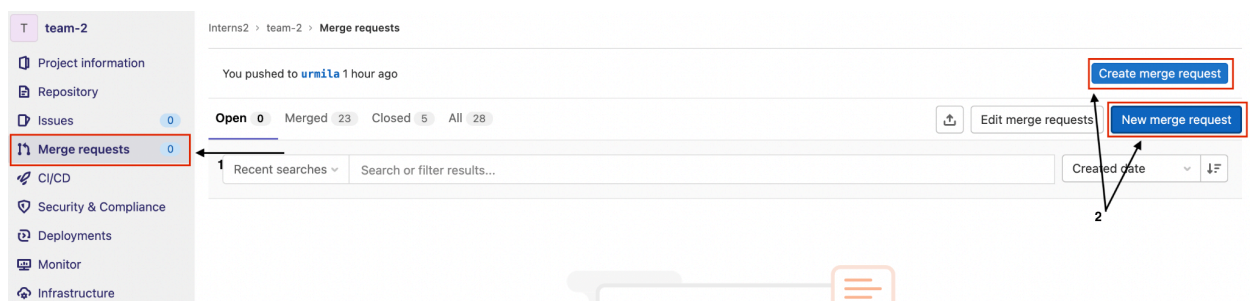
- git checkout master
- git pull <remote-name>master

Step 3: First check the branch by using the given command . If you are not in that branch, switch to the master branch before pulling. After switching merge master branch into individual branch and push it to the remote.

- git branch
- git checkout <branch-name>
- git merge master
- git push origin <branch-name>

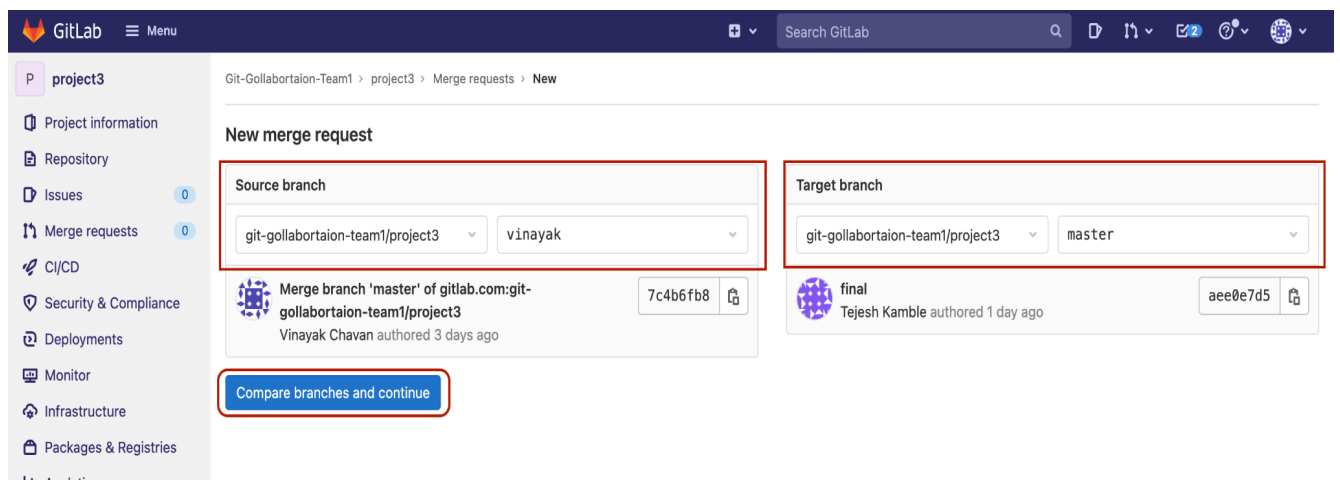
Step 4: Go to remote repository

- This should only be done by Developer => Click on merge request(because only maintainer or owner can merge the branch)
- Then click on create new merge request
-



Step 5: Select source and target branch

- Add the branchName into select source field
- Add master/target branch into target field
- Click on compare branches and continue



Step 6: Create merge request

- Add assignee and reviewer name you want to send request
- Uncheck delete source file option
- Click on create merge request

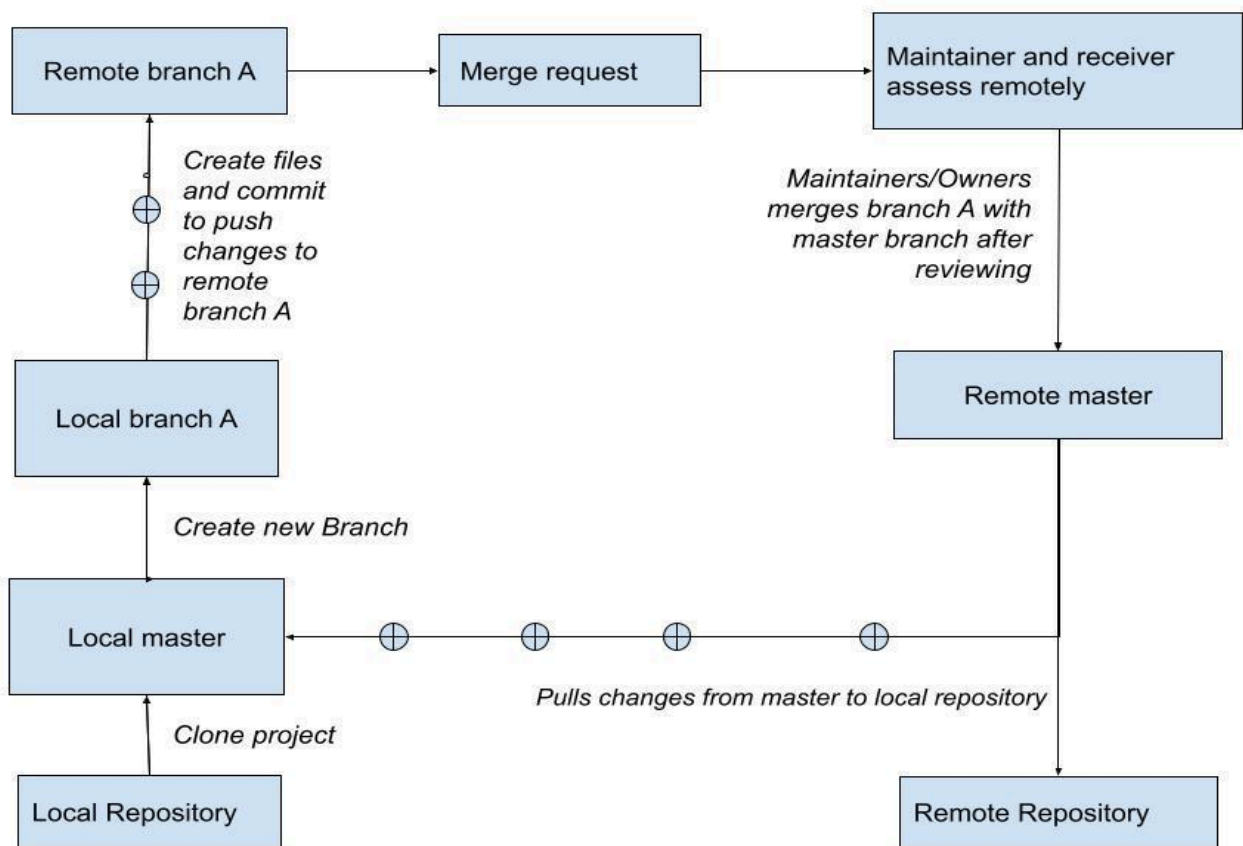
The screenshot shows the GitLab interface for creating a merge request. The left sidebar contains navigation links for project3, including Project information, Repository, Issues, Merge requests, CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The main content area has a 'Title' field with the value 'Draft: Vinayak' and a note to remove the 'Draft' prefix. Below the title is a 'Description' field with a rich text editor. The 'Assignee' field is set to 'Vinayak Chavan' and the 'Reviewer' field is set to 'Yash Rajput'. The 'Milestone' field is set to 'Milestone' and the 'Labels' field is empty. The 'Merge options' section has two checkboxes: 'Delete source branch when merge request is accepted' (which is checked) and 'Squash commits when merge request is accepted'. A red box highlights the 'Delete source branch' checkbox, with an arrow pointing to it from the text 'uncheck this option'. Another red box highlights the 'Assignee' and 'Reviewer' fields, with an arrow pointing to it from the text 'select assignee and reviewer'. The 'Create merge request' button is at the bottom.

Step 7:

- The request will be accepted/approved by the maintainer only.
- If there is some error in the file the maintainer closes the merge request.

NOTE

1. If the remote branch is ahead of the local branch then it will cause an error.
2. Conflict errors occur only when some command is missed or changes are not pulled properly like commits are not pushed to the remote.
3. 'git pull' while in master branch to get a list of all available branches, with which we can access all files from different branches(Not recommended)



Working of Remote Collaboration

Git Conflict Resolve Steps From Command Line

Step 1: First checkout to master branch. Pull all the changes from remote repository to your local repository.

- `git checkout <remote-branch-name>`
- `git pull <remote-name> <remote-branch-name>`

Step 2:

Add some changes in any file in the master branch.

```
> cat >> <file-name>
> git status
> git add <file-name>
> git commit -m "adding changes in file of remote-branch"
```

Step 3:

Checkout to the branch that you have created.

```
> git checkout <your-branch-name>
```

Step 4:

Add some changes in the same file in your branch .

```
> cat >> <file-name>
> git status
> git add <file-name>
> git commit -m "adding changes in file "
> git merge <remote-branch-name>
```

Step 5:

When you merge the master branch to your branch you will see the conflict occur in the file (as shown in fig 1.1). To solve the conflict click on one of the three options given in the file [(1)Accept current changes (2)Accept incoming changes (3)Accept both the changes (4)Compare Changes] .Save ,add and commit the file.

```

file1
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<<< HEAD (Current Change)
2 heyyy
3 blessed
4 gratitude
5 =====
6 hello
7 grateful
8 >>>>>> master (Incoming Change)
9

```

Conflict Error

- git status
- git add <file-name>
- git commit -m "solve the conflict"

Step 6:

Now the conflict is solved (as shown in fig 1.2.). Then checkout to master branch. Merge your branch in master branch. Push the master branch to remote repository.

```

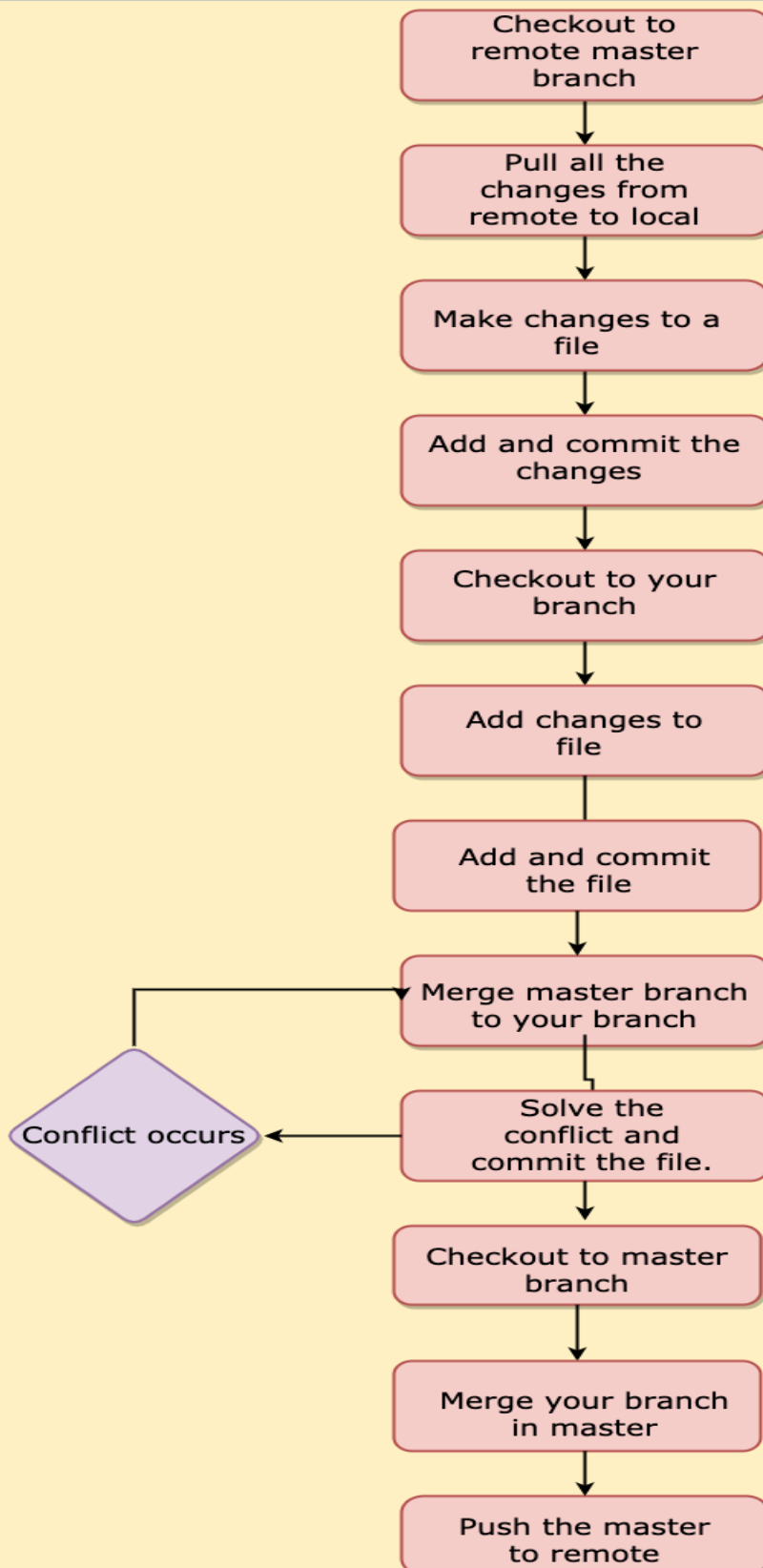
file1
1 heyyy
2 blessed
3 gratitude
4 hello
5 grateful
6 heyyy
7

```

Fig 1.2. Conflict Error Resolved

- git checkout <remote-branch-name>
- git merge <your-branch-name>
- git push <remote-name> <remote-branch-name>

Git Conflict Resolve From CLI



1. Git Conflict Resolve Steps From Remote GUI

Step 1:

First checkout to master branch. Pull all the changes from remote repository to your local repository.

- `git checkout <remote-branch-name>`
- `git pull <remote-name> <remote-branch-name>`

Step 2:

Checkout to your branch and add some changes in file (Readme.md) in the branch. Push your branch to remote.

- `git checkout <Your-branch-name>`
- `cat >> <file-name>`
- `git status`
- `git add <file-name>`
- `git commit -m "adding changes in file of remote-branch"`
- `git push <remote-name> <Your-branch-name>`

Step 3 :

Go on remote and send a request to merge your branch into <remote-branch> and select maintainer as assignee and reviewer. The first one to merge won't get any conflict error and the rest will get conflict (as shown below fig 2.1 & fig 2.2.).

added new file

!22 · created 1 minute ago by Ajay Ugale

shows that there is a conflict error



updated 1 minute ago

Fig 2.1.

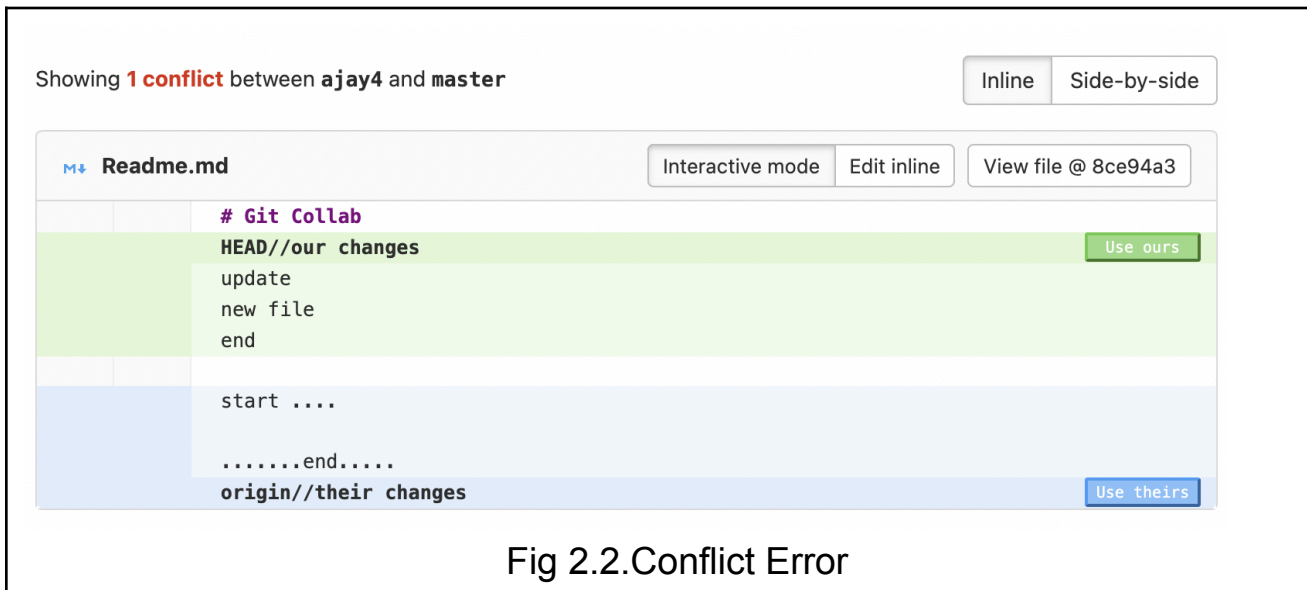


Fig 2.2.Conflict Error

Step 4:

Since we are going to resolve the conflict in local you have to checkout to your branch .When you merge the <remote-branch> to your branch you will see the same conflict occur in the file as shown in above figure.To solve the conflict click on one of the three options given in the file[(1)Accept current changes (2)Accept incoming changes (3)Accept both the changes (4)Compare changes] .Save ,add and commit the file.

- git checkout <your-branch-name>
- git merge <remote-branch >
- git status
- git add <file-name>
- git commit -m "solve the conflict"

Step 5 :

Then push your branch to remote and now the maintainer will approve and merge your request (as shown in fig 2.3.).

```
> git push <remote> <Your-branch>
```

 **Merge request approved.** Approved by 

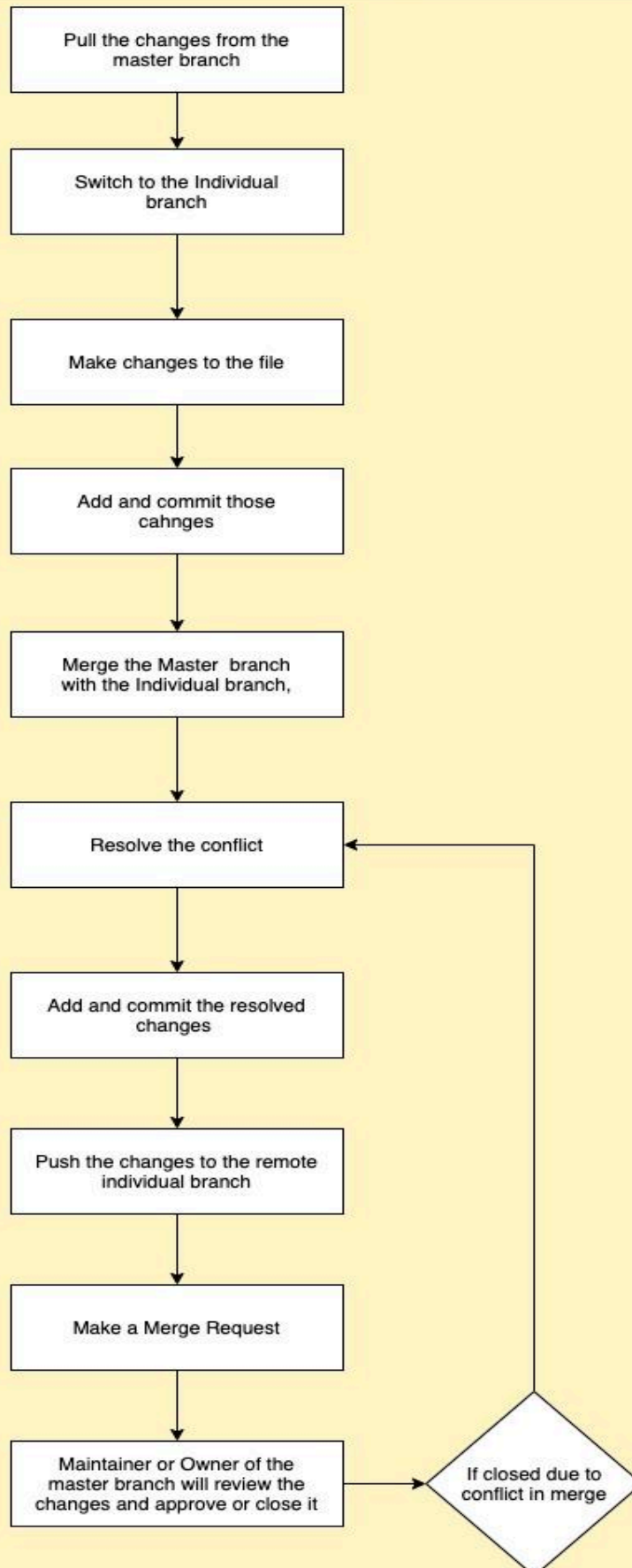
 **Merged by**  maintainer 1 minute ago [Revert](#) [Cherry-pick](#)

The changes were merged into [master](#) with [69e83a90](#) 

You can delete the source branch now [Delete source branch](#)

Fig 2.3.Error Resolved & Merged

Workflow of conflict resolve from remote CLI



Note:

- Conflict should not be solved in the main branch or the branch that is shared with other as it will pollute the shared branch
- Solve the conflict in your individual branches so the changes doesn't affect the shared branches
- Always save the file before committing
- Commit only files that has any changes that are needed to be pushed to the remote and not all the files in the directory as it will commit all the unnecessary changes
- Give the commit proper message for better understandability

WEB DEVELOPMENT BUILDING BLOCKS

IP Address

- There is a source and destination in every interaction over a network
- Every system has an IP assigned to it
- IP Address is a unique address that identifies a device on the internet or local network.
- It helps in connecting your computer to other devices on your network and all over the world.
- IP Address was created by IANA(Internet Assigned Number Authority).

Types of IP Address:-

- IPv4:
 - ◆ IP Addresses are made up of binary values.
 - ◆ Pv4 Addresses are 32 bits long.
 - ◆ The network part indicates the distinctive variety that's appointed to the network.
 - ◆ The host part uniquely identifies the machine on your network.
 - ◆ Local networks that have massive numbers of hosts are divided into subnets.
- IPv4 Classes:
 - Class A:
 - ◆ 1-126
 - ◆ Used for Large Network
 - ◆ 255.0.0.0 subnet mask 8 bits
 - ◆ Example 1.1.1.1
 - ◆ IP that starts with 127 is assigned to a local system
 - ◆ 10.0.0.0- 10.255.255.255 private network IP
 - Class B:-
 - ◆ 255.255.0.0 subnet mask 16 bits
 - ◆ 128-191
 - ◆ 128.0.0.0 - 191.0.0.0
 - ◆ 172.16.0.0 to 172.31.255.255 private network IP
 - ◆ Used for medium size networks.

→ Class C:-

- ◆ Used for local area networks.
- ◆ 255.255.255.0 subnet mask 24 bits
- ◆ 192-223
- ◆ 192.168.0.0 local network ID
- ◆ 192.168.0.0 to 192.168.255.255

→ Class D:-

- ◆ Reserved
- ◆ 224-239
- ◆ Not used

→ Class E:-

- ◆ 240 to 255
- ◆ Never used.

→ IPv6

- ◆ IPv6 addresses are 128 bits long.
- ◆ IPv6 addresses are alpha numeric unlike IPv4 which have numeric addresses.
- ◆ IPv4 had a limited number of addresses, hence IPv6 was introduced.

PORTS

- It is the entry point of the application.
- IP address is just an identification it needs port to establish connection over the network from both ends
- 443 is reserved for HTTPS
- 80 is reserved for HTTP
- 22 is used in SSH connection
- There are around 65535 ports.
 - ◆ 0 - 1023:
 - These ports used only by system
 - ◆ 1024 - 49150:
 - 3000 - 9999 are mostly used for web application

◆ 49151 - 65535:

- These are open ports
- System uses these ports to connect to other applications

DNS

- DNS stands for domain name system
- We can't always remember IP addresses so they are given unique names.
- Domain gives us readability.
- Domain name is mapped to IP Address.
- When we enter something it will first call the DNS and get the IP from DNS.
- DNS resolution is the process of translating IP addresses to domain name
- 1.1.1.1 is a free Domain Name System service by Cloudflare
- 8.8.8.8 is address provided by Google for Google Public DNS

HTTP

- HTTP stands for HyperText Transfer Protocol.
- HTTP is a protocol which allows us a way to interact with web resources such as HTML, audio, video and other files.
- HTTP connection allows data to be sniffed.
- HTTP uses port 80 used by default.
- Normal web pages use HTTP.

HTTP verb:-

- GET - Fetch data from the server => Read
- POST - Submit data to the server => Create
- PUT - Update data on the server => Update
- DELETE - Delete data from the server => Delete

Above verbs are called as CRUD

- OPTION-Used in the browser for queries
- HEAD- Used in the browser for queries

HTTP Headers

- HTTP headers let the client and the server pass additional information with an HTTP request or response
- Request header holds the information about the resource to be fetched
- Response header holds the information about the response

HTTP Status Codes

- 100 - Informational (not used)
- 200
 - ◆ 200 - ok (request successfully)
 - ◆ 201 - created(request has succeeded and has led to the creation of a resource)
 - ◆ 204 - no content(there is no content to send back to client.)
- 300
 - ◆ 301- Moved Permanently
 - ◆ 302- Not Found
 - ◆ 307- Temporary Redirect
 - ◆ 308- Permanent Redirect
- 400
 - ◆ 400- bad request(invalid request)
 - ◆ 401-unauthorized(did not have access)
 - ◆ 403-forbidden(refuse authorize)
 - ◆ 404- not found
 - ◆ 429- to many request
 - ◆ 422-unprocessable entity
- 500 - server error
 - ◆ 501- internal server error
 - ◆ 502-Bad gateway

HTTPS

- HTTP stands for HyperText Transfer Protocol Secure.
- HTTPS encrypts data that prevents data sniffing.
- HTTPS uses port 443.
- All production applications run on HTTPS.

Database

- The collection of multiple data entries together creates a database
- Database is a collection of information that can be stored, updated and deleted.
- There are two types of Databases
 - ◆ Relational Database
 - It is uses table approach
 - mySQL
 - postgresQL
 - ◆ Non Relational Database
 - It is used schema (key value pair)
 - mongoDB
 - Redis

Software License

- Apache 2.0
- MIT
- GPL(General public license)
- BSD(Berkeley Software Distribution)
- Mozilla

Cloud Provider

- AWS
- Google Cloud
- Azure
- Digital Ocean
- Vultr

JAVASCRIPT

Four Pillars of Javascript:

1. Variables
2. Objects
3. Arrays
4. Functions

Variables:

- Variables are containers that hold a value.
- Javascript variables are dynamic type of variables
- There are two data types of variables:

- ◆ Primitive
- ◆ Non-Primitive

- Primitive variables:

- ◆ Types of primitive variables:

- number
- string
- boolean
- undefined
- null
- symbol

- ◆ Properties:

- Primitive variables are copied or passed by value
- 'false', "", NaN, 0, undefined, null are falsy
- All other values are truthy

→ Non-Primitive variables:

◆ Types of Non-Primitive variables:

- Arrays
- Objects

◆ Properties:

- Non-Primitives are copy or passed by reference
- All Non-Primitives are always truthy even when empty

→ We declare variables using var(ES5), let and const(ES6)

Var	Let	Const
ES5	ES6	ES6
Values can be reassigned	Values can be reassigned	Values cannot be reassigned
Functional Scope	Lexical Scope	Lexical Scope
Hoisted	Doesn't get Hoisted	Doesn't get Hoisted

→ ES5:

- ◆ Declared using 'var' keyword
- ◆ Declaration gets hoisted
- ◆ Can be redeclared, reinitialized and reassigned
- ◆ Has functional scope
 - Local variables with same name overrides the global variables
- ◆ Variables can be called before declaration but will hold the value undefined

→ ES6 and above:

- ◆ Declared using 'let' and 'const' keywords
- ◆ Doesn't get hoisted

- ◆ Cannot be redeclared
- ◆ 'let' can be reassigned while 'const' cannot be reassigned
- ◆ 'const' should always be initialized while declaring
- ◆ Variables cannot be used before it is declared
- ◆ It has lexical or block scope (With in { })

Tech debt

→ '+'

- ◆ Used concat and add depending on variable type.
- ◆ Concat if any one of the variables is string type.
- ◆ Ex. 1 + "1" = 11

→ '=='

- ◆ identify/type check.(only check number not a datatype)

→ 'Null'

- ◆ If checking for null type returns a type of object.
- ◆ typeof(null) => object

Operators

→ Operators are used in expressions

→ Arithmetic : +, -, *, /, %, ++, --, +=, -=, ==, ===, >=, <=.

→ Logical: '&&', '||', '!'

→ "&&"

true	true	true
false	true	false
true	false	false
false	false	false

→ " || "

true	true	true
false	true	true
true	false	true
false	false	false

→ " ! "

true	false
false	true

→ Ternary : " ? : "

◆ A ? B : C

Falsy value: When the Boolean equivalent is false it is considered as Falsy value.

- NAN
- undefined
- 0
- " "
- false

Truthy value: When the Boolean equivalent is true it is considered as Truthy value. All non-primitive and except than truthy are truthy value

- { }
- []

Statements

→ If-else

- ◆ Use the if statement to specify a block of code to be executed if a condition is true.
- ◆ `if(condition){`

`} else {`

`}`
- ◆

→ For-each

- ◆ Used with arrays most of the time

→ For

- ◆ The for loop will loop through a block of code a number of times.
- ◆ `for(condition){`

`}`

→ Switch-Case

- ◆ switch statement to select one of many code blocks to be executed

→ Do-while

→ While loop

ES6

→ Arrow function

- ◆ Always an expression
- ◆ Cannot get hoisted
- ◆ Doesn't have its own 'this'
- ◆ `() => {}`
- ◆ Unnamed/Anonymous

→ ... spread operator

- ◆ Extracting the values from an array or object.
- ◆ Appears on the right-hand in destructuring.
- ◆ ... rest
 - Extracting the values from an array or object.
 - Appears on the left-hand side in destructuring

- 'let' and 'const'
- class(constructor)
- symbols (we don't use it)
- Destructure
 - ◆ Destructuring helps us unpack values from arrays and properties from objects into variables.
 - ◆ Destructuring is a convenient way of extracting multiple values from data stored in (possibly nested) objects and Arrays
- Template literals
 - ◆ Template literals or String literals allows to embed values within a string
 - ◆ Template literals uses backticks
 - ◆ `Hello, this is an \$(example) of template literal`

Array and Object

- Non- Primitives are containers
- Collection of data which can hold all data types
- Arrays is a special object with numeric keys as keys
- Object containers with key-value pairs
- Object keys are denoted or referenced using
 - ◆ Dot notation (for strings)
 - ◆ Bracket notation
- Array indices are referenced using
 - ◆ Bracket notation
- Object declaration using object literals


```
...
var obj = { };
var obj = new Object();
...
```

→ Array declaration using array literals

...

```
var arr = [];  
var arr = new Array();
```

...

→ Reference to array value

...

```
arr[1]
```

...

→ Reference to object value

...

```
obj.name = "";  
obj["name"] = "";
```

...

→ Arrays always reference to numeric keys called indexes

→ Objects reference to keys that are always strings

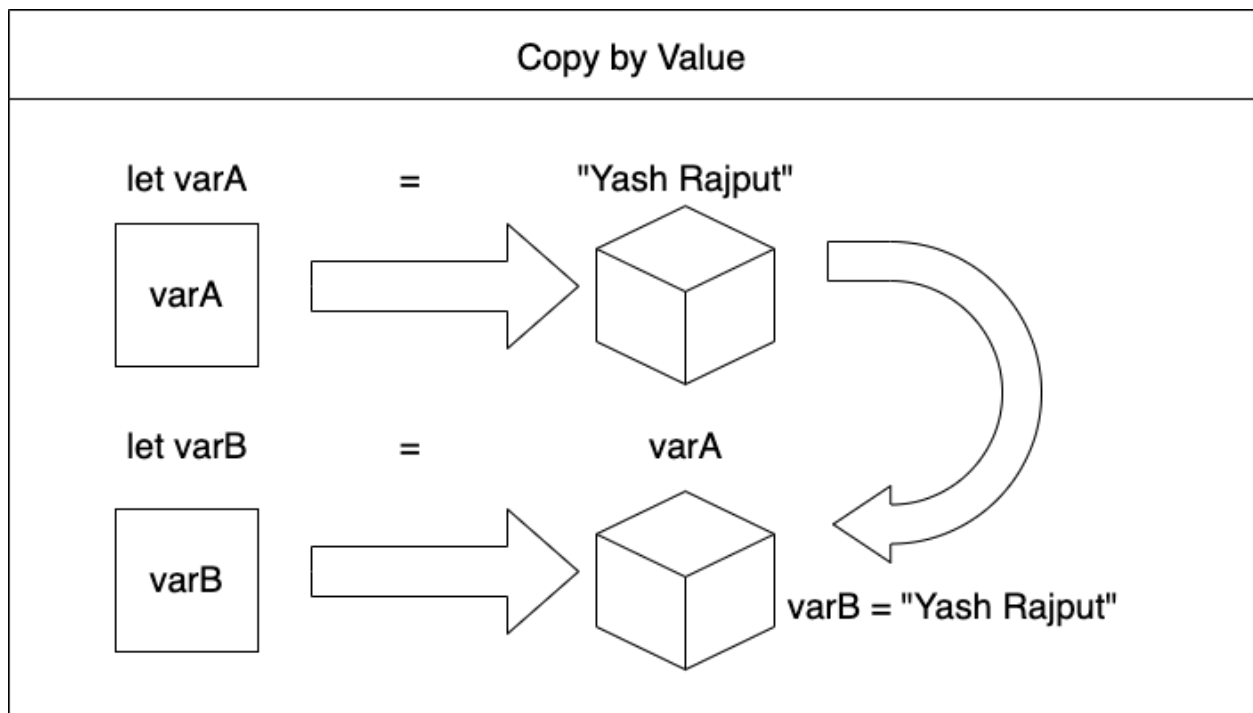
WEEK #2

Copy/Pass by Value

- All primitive data types are copy/pass by value
- When we create a variable(varA) a memory location(x1010) is created for the variable and the value("Yash Rajput") is stored in there
- When we create another variable(varB) a new memory location(x2020) is created for that variable
- If we assign the first variable(varA) to the second variable(varB) the value from the first variables' memory location(x1010) is copied to the second variables' memory location(x2020)
- varB gets a copy value of varA onto the new memory location

...

```
let varA = " Yash Rajput";  
let varB = varA;  
console.log(varB);    // prints "Yash Rajput"  
...
```



Copy/Pass by Reference

- All non-primitives are copy/passed by reference in javascript
- When a variable is created a memory location is created which holds the reference to another separate memory location which holds the values of the variable assigned to it
- If we change any value in either variable, values in all the variables get updated because reference of the memory location is passed or copied not the actual value

FUNCTIONS

- Functions are a block of code that performs the given task or condition when called and are reusable
- Functions act as a container similar to variables
- Function syntax:

```
...  
Function function-name(){  
    //block of code to be used  
}  
function-name()  
...
```

- In Javascript functions behaves same as a variable, it has same properties of a variable

FUNCTION SCOPE

- Function has acts same like a variable does, it has a local and global scope
- Local scope of a function is also referred to as functional scope because the variable declared inside the function has only scope within that function

```
let x = "Yash";  
function funcA() {  
    let x = "Rajput";  
    console.log(x);  
}  
console.log(x); //outputs Yash since it has scope only within  
function  
funcA();        // outputs Rajput it has a global scope
```

- Local scope overrides global scope hence the value of x inside the function is not overridden but it is otherwise

```
let x = "Global scope, declared outside function";//global scope
funcA(); //function called before declaration

function funcA() {
let y= "Local scope, declared inside function";    //local scope
console.log(x);
}

console.log("Outside function");
console.log(x);
```

Output:

```
yashrajput@Yashs-MacBook-Air sessions %
Inside function
declared outside function
Outside function
declared outside function
```

- Function declaration gets hoisted but function assignment does not
- Hence, a function can be called before it is declared
- A function must always return something

FUNCTION DECLARATION

```
function funcA(){
    console.log("This is a function");
}
console.log(funcA());
Output:
This is a function
undefined
yashrajput@
```

- When a function is declared we can use it anytime whenever it is called or invoked
- We use 'function' keyword to declare a function followed by function name and '()' which takes parameter that can be passed and accessed inside the function
- We must explicitly return a function, or else js engine will return undefined by default
- When we pass a value in a function calling it dynamically takes it as a parameter

FUNCTION EXPRESSION

- Function can also be assigned to as an expression which is called function expression or function assignment
- An expression is a series of statements that evaluates to a value

```
var funcA = function(){
    console.log("Function Expression");
}
funcA();
```

- Above is called as function expression or function assignment
- As everything in javascript is an object, functions are also objects, hence they are passed by reference when assigned to a variable
- funcA holds the reference of function not the actual function
- Function declaration gets hoisted but the reference does not get hoisted
- Function expression can be assigned multiple times to a variable
- This is also called as anonymous function or unnamed function

FIRST CLASS CITIZEN

- As functions in javascript behaves same and have same properties as that of a variable they are called as first class citizens
- Function can be passed as a parameter to another function, assigned to a variable and one function can return another function
- Function declaration gets hoisted not the assignment
- Everything in JS is an Object, hence function can be called as an object, objects are non-primitive and non-primitives are copy/pass by reference
- Function is an object, hence it is Non-Primitive and when Non-Primitive are assigned to a variable it holds the reference to the value. Hence function holds reference of function not the actual function
- Function declaration gets hoisted not the assignment

```

// Function assigned to a variable
const funcA = function() {
  console.log("Yash Rajput");
}
funcA(); // Invoke it using the variable
----
// Function passed as an argument
function firstName(){
  return "Yash"
}
function fullName(name,lastname) {
  console.log(name() + lastname);
}
fullName(firstName," Rajput"); // passed as a parameter
----
// Function returning a function
function sayHello() {
  return function() {
    console.log("Hello!");
  }
}

```

Higher Order Functions

- Higher-order functions are those functions that take other functions as arguments or return other functions.
- The function passed as arguments in higher-order function is known as a callback
- We can wrap some condition or code in a function (*callback*) and pass that function to another function(Higher Order Function)
- It is used to wrap another function

```
const hof=function (fnparam) {  
  return function(p1){  
    console.log('param1',p1);  
  
    return fnparam(p1)  
  }  
}  
  
const finalfN= function(p2){  
  console.log('param2', p2);  
  return 10 + p2;  
}  
  
const result =hof(finalfN);  
const value = result(20);  
console.log("value",value)
```

Output:

```
param1 20  
param2 20  
value 30
```

Pure Functions

- Pure functions are functions that will always return the same output
- This function does not depend on any data that is mutable or variables that is declared outside its scope
- Pure functions cannot modify anything outside its functional scope

```
const printName = function(name) {
    return ("Hey! My name is ",name);
}
console.log(printName('Yash'));
```

IIFE/SEF

- Immediately Invoked Function Expression or Self Executing Function it is a function invoked immediately after its declaration
- The function is enclosed within brackets '()' and a semicolon is used just before the function declaration
- IIFE allows the developers to not let the variable inside the function pollute outside its scope
- It doesn't allow to access the variable inside the inner scope of an IIFE
- IIFE can be called only once
- If we return something we can assign it to a variable which can be called once

```
const counter = (function(parameter){
    return function(step){
        let count = parameter;
        return function(){
            count = count + step;
            return{
                count: count
            }
        }
    }
})
```

```

})(100)
const step = counter(4);
const step2 = counter(8);
const step3 = counter(16);
const step4 = counter(32);
console.log(step());
console.log(step());
console.log(step());
console.log(step());

```

CLOSURE

- Closures preserve the outer scope inside an inner scope
- Closure gives you access to an outer function's scope from an inner function
- Closures are created every time a function is created, at function creation time
- Using closure we can make the private variable and methods like in java language
-

```

function funcA() {
    let varA = 5;
    function funcB() {
        console.log(varA);
    }
    funcB();
}
Let result =funcA();

```

- funcB has its functional scope within funcA, first funcB will check its scope for varA if it is not found it will look for it in its parents scope

INLINE FUNCTION

- Inline function is a type of anonymous function that is assigned to a variable
- It is an anonymous function that is assigned to variable and given a name to it and is created at runtime

```
const hof=function (fnparam) {
return function(p1){
    return fnparam(p1)  // inline function call from here.
}
}
const result = hof (function(l1){ //pass function inline as
parameter
    return 10 + 11;
})(20)
console.log("value",result);
```

```
// iife , HOF ,inline function (all concept)
const result = (function (fnparam) {
    return function(p1){
        return fnparam(p1)  // inline function call from here.
    }
})(function(l1){ // pass function inline as parameter.
    return 10 + 11;
})(20)
console.log("value",result);
```

ARROW FUNCTION

- Arrow function are declared using variable
- They are anonymous functions without any name
- We do not use function keyword to declare arrow function
- If the expression is in inline we don't have to use return keyword
- If there is a single parameter we don't have to use parentheses
- Arrow function can be only expression not declaration it means it cannot get hoisted

```
const location = {  
  name : "mumbai" ,  
  pincode : "743" ,  
  places : ['CST', 'dahisar' , 'juhu']  
}  
  
const getLocation = location => { //It is only  
  expression not declaration  
  return location.name + ", " + location.pincode  
}  
  
const text= getLocation(location);  
console.log(text);
```

- Different ways we can declare an arrow function

```
//If arrow function does not take parameter we have to use "()" "  
const fnArrow = () => 'something';  
  
//If function taking one parameter we write like this or below  
const fnArrow1= (param) => 'new' + param;
```



```
//If function taking 1 parameter we write like this or below
    const fnArrow2 = param => 'new' + param;

//If function take 2 or more parameter that time it is mandatory to use
parentheses
    const fnArrow3 = (param,param2) => 'new' + param;
```

CURRING

- Currying is a transformation of functions that translates a function from callable
- Currying is a process in which we can transform a function with multiple arguments into a sequence of nesting functions
- It returns a new function that expects the next argument inline

```
//Function without currying
function sum(a, b, c) {
    return a + b + c;
}
sum(1,2,3);

//Above function with currying
function sum(a) {
    return (b) => {
        return (c) => {
            return a + b + c
        }
    }
}
console.log(sum(1)(2)(3))
```

CONSTRUCTOR FUNCTION

- Adding a 'new' keyword to a constructor makes it a constructor function.
- Object is created using the constructor function when we use ' . '
- Object is derived from a prototypal object and this prototypal object derived from another prototypal object there is a continuous chain.
- 'this' keyword points to the current object.

```
const Emp = function(name,city){
  this.name = name;
  this.city = city;
  this.getEmpDetails = function(){
    return this.name + " "+this.city
  }
}

const Employee = new Emp("Yash Rajput","Mumbai");
```

BUILT-IN CONSTRUCTOR FUNCTION

- String
- Number
- Array
- Object
- Boolean
- Math
- Date

OBJECT BUILT-IN METHOD:

The constructor property returns a reference to the Object constructor function that created the instance object.

- `Object.prototype.keys = function(){`
 `}`

```
const obj = {  
  key1: 1,  
  key2: 2  
}  
  
const result = Object.entries(obj);  
console.log(result);
```

- `Object.prototype.values = function(){`
 `}`

```
const obj = {  
  key1: 1,  
  key2: 2  
}  
  
const result = Object.values(obj);  
console.log(result);
```

- `Object.prototype.entries = function(){`
 `}`

```
const obj = {  
  key1: 1,  
  key2: 2  
}  
  
const result = Object.entries(obj);  
console.log(result);
```

ARRAY BUILT-IN METHODS

- `Array.isArray()` : To check if the array is an array or not

```
var a=[3,4,5,6]
var b={}
console.log(Array.isArray(a))
console.log(Array.isArray(b))
```

- `forEach()` : Iterate all the items of the array.

```
var a=[1,2,3,4]
a.forEach(function(item,index) {
  console.log(item,index)
})
```

- `map()` : Modify the array and return a new array of the same length.

```
var a=[2,3,5,6]
var b=a.map(function(item,index) {
  return item+index;
})
console.log(b)
```

- `concat()` : Concat two or more arrays into another array and return an array.

```
var a=[1,2,3,4]
var b=[5,6,7,8]
var c = a.concat(b)
console.log(c)
```

- `push()` : Add the element at the end of the array.

```
var a=[1,2]
a.push(3)
console.log(a)
```

- `pop()` : Remove the element from the end of the array

```
var a=[1,2,3]
a.pop()
console.log(a)
```

- `shift()` : Remove the element from the beginning of the array

```
var a=[1,2,3]
a.shift()
console.log(a)
```

- `unshift()` : Add the element at the beginning of the array.

```
var a=[2,3]
a.unshift(1)
console.log(a)
```

- `slice()` : Extract the part of the array between two indexes, starting is included and ending of the slice is excluded.

```
var a=[5,6,7,4]
console.log(a.slice(0,2))
```

- `includes()` : It checks whether the element is present in array or not. If element is present it will return true if not then false.

```
var a=[1,2,3,4]
console.log(a.includes(1))
```

- `indexOf()` : If element is present in array it gives its index if element is not present in array it gives -1 that means element is not present in array.

```
var a=[2,3,4,6]
console.log(a.indexOf(5))
```

- `join()` : Joins the all element of array using delimiter, output is an array converted to string.

```
var a=[1,3,4,5]
console.log(a.join(", "))
```

STRING BUILT-IN METHODS

- `split()` : Split the text using a delimiter and return an array.

```
var a="Yash Rajput";
console.log(a.split(" "));
```

- `toUpperCase()` : Convert a character into upper case.

```
var a="yash rajput";
console.log(a.toUpperCase());
```

- `toLowerCase()` : Convert a character into lower case.

```
var a="YASH RAJPUT";
console.log(a.toLowerCase());
```

- trim() : Removes the space at beginning and at the end of text.

```
var a = " Yash Rajput ";  
console.log(a.trim());
```

- replace() : Replace a text or character with another text or character.

```
var a = "Pallavi Muneshwar";  
console.log(a.replace("Yash", "Rajput"));
```

- substring() : Extract text using index.

```
var a = "Yash Rajput";  
console.log(a.substring(2));
```

- charAt() : It will give at which index which character is present.

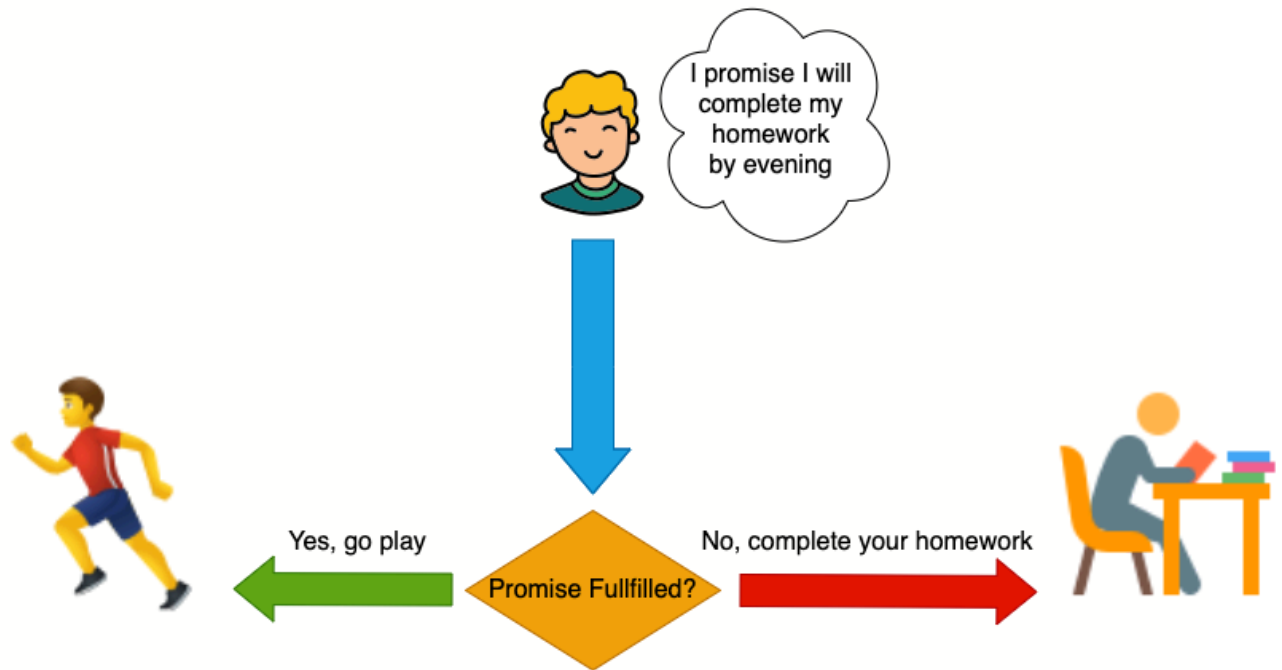
```
● var a="Yash Rajput";  
● console.log(a.charAt(5));
```

WEEK #3

- Anything in a code that takes time to execute leads to asynchronous execution
- We used callbacks to perform and handle asynchronous executions, but it used to create something called a `callback hell`.
- A callback hell is where each callback is nested inside another callback, and each inner callback is dependent on its parent.
- In ES6, `Promise` was introduced. Using promises, we can avoid the `callback hell` and make our code cleaner, easier to read, and easier to understand.

Promise

- A promise in JavaScript is similar to a promise in real life. When we make a promise in real life, it is a guarantee that we are going to do something in the future
- Promises can only be made for the future. A promise has two possible outcomes: it will either be kept when the time comes, or it won't.
- When we define a promise in JavaScript, it will be resolved when the time comes, or it will get rejected.
- Promise is an object that holds the future value of an async operation

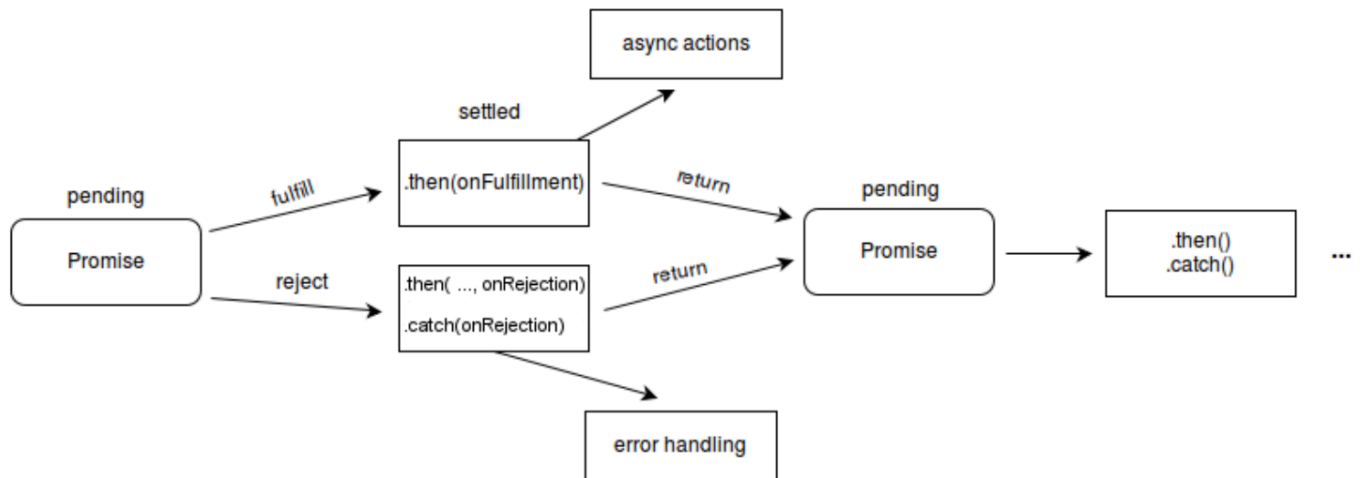


- Promises are one of the ways we can deal with asynchronous operations in JavaScript.
- The promise is commonly defined as a proxy for a value that will eventually become available.
- It is a way of defining a function in such a way that we can synchronously control its flow.

Creating a Promise:

- To create a promise in JavaScript, we use the `Promise` constructor:
 - The Promise constructor accepts a function as an argument.
- This constructor accepts two functions with the names: `resolve()` and `reject()`.
- When you call the new Promise, the function is called automatically.
- Inside the promise, you manually call the `resolve()` function if the promise is completed successfully and invoke the `reject()` function in case an error occurs.
- The Promise object supports two properties: `state` and `result`
- A promise has three states:
 - `pending`: initial state, neither fulfilled nor rejected.
 - `fulfilled`: meaning that the operation was completed successfully(Promise successful)
 - `rejected`: meaning that the operation failed.(Promise failed)

```
const myPromise = new Promise((resolve, reject) => {  
  
  // "Producing Code" (May take some time)  
  
  let condition;  
  
  if(condition is met) {  
  
    resolve('Promise is resolved successfully.');//when  
successful  
  
  } else {  
  
    reject('Promise is rejected'); // when error  
  
  }  
  
});
```



Consuming a Promise:

- We can use three methods to consume a promise:
 - then()
 - The then() method is used to schedule a callback to be executed when the promise is successfully resolved.
 - The then() method takes two callback functions

```
myPromise.then(onFulfilled, onRejected);
```

- The onFulfilled callback is called if the promise is fulfilled. The onRejected callback is called when the promise is rejected.

- catch()
 - If you want to schedule a callback to be executed when the promise is rejected, you can use the catch() method of the Promise object.
 - Internally, the catch() method invokes the then(undefined, onRejected) method.
- finally()
 - Will be executed no matter if the promise is fulfilled or rejected

```
const promise = new Promise(function(resolve, reject) {
  const x = "YASH RAJPUT";
  const y = "yash rajput"
  if(x.toLowerCase() === y.toLowerCase()) {
    resolve();
  } else {
    reject();
  }
});

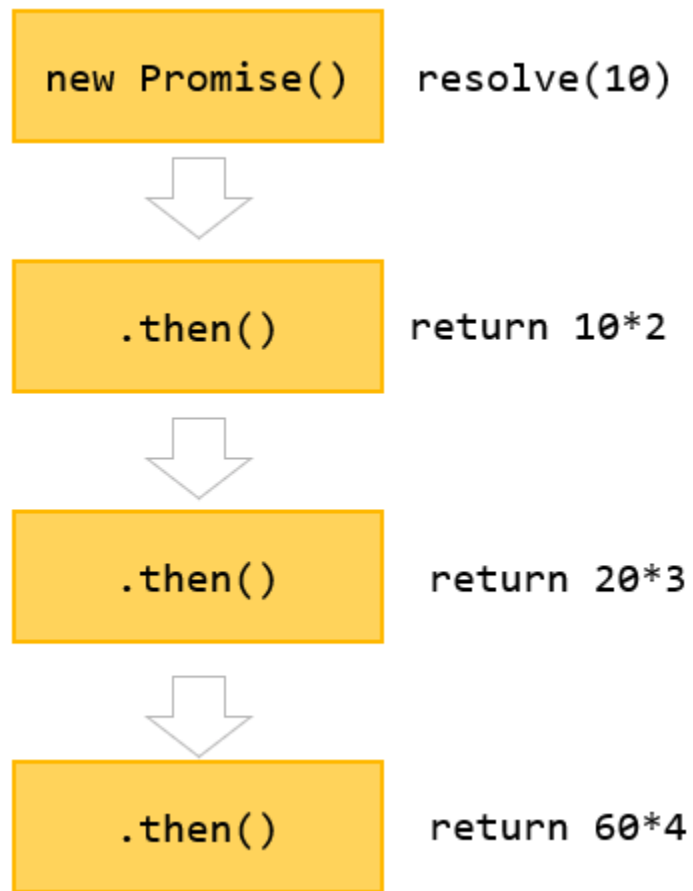
promise.
  then(function () {
    console.log('Success, You are Yash Rajput');})
  .
  catch(function () {
    console.log('Identity error');})
  .
  finally(function() {
    console.log('Promise ended');
  })
```

Promise Chaining

- The instance method of the Promise object such as then(), catch() or finally() returns a separate promise object.
- Therefore, you can call the promise's instance method on the return Promise.
- The successively calling methods in this way are referred to as the promise chaining

```
let pro = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve(10);
  }, 3000);
});

pro.then((result) => {
  console.log(result); // 10
  return result * 2;
}).then((result) => {
  console.log(result); // 20
  return result * 3;
}).then((result) => {
  console.log(result); // 60
  return result * 4;
});
;
```



Working of above promise chaining code

Async/Await

- Async/Await is a type of Promise.
- Promises are objects that can have multiple states.
- Promises do this because sometimes what we ask for isn't available immediately, and we'll need to be able to detect what state it is in.
 - pending: when you first call a promise and it's unknown what it will return.
 - fulfilled: meaning that the operation completed successfully
 - rejected: the operation failed
- The `async` and `await` keywords help us in handling Promises more conveniently and it is more readable than regular promise handling using `then()` and `catch()`

```
//Promise using then() and catch()

let age = 24;

let promise = new Promise((resolve, reject) => {

  setTimeout(() => {

    if(age > 18) resolve('I can apply for driving license');

    else reject(new Error('I am not eligible yet'));

  }, 1000);

});

promise.then(res => console.log(res)).catch(err => console.log(err));
```

```
//Using async/await

async function license() {

  let promise = new Promise((resolve, reject) => {

    setTimeout(() => {

      resolve('I can apply for licence');

    }, 1000);

  });

  let res = await promise;

  console.log(res); // I am resolved

}

license();
```

- The async keyword:
 - If we use async before any function then the function will return a Promise.
 - If the function is not returning any Promise explicitly then it will wrap a Promise around the return value implicitly
- The await keyword:
 - The await keyword waits for a Promise and pauses the code in that line until the Promise gets resolved.

- The await keyword should always be used inside the async function otherwise, it will throw an error.
- Handling errors:
 - We can use try...catch block to handle errors in async/await

```
async function foo() {  
  let promise = new Promise((resolve, reject) => {  
    setTimeout(() => {  
      resolve(new Error('I am rejected'));  
    }, 1000);  
  });  
  try {  
    let res = await promise;  
    console.log(res);  
  }  
  catch (e) {  
    console.log(e);  
  }  
}  
foo(); //ERROR i am rejected
```

NODE

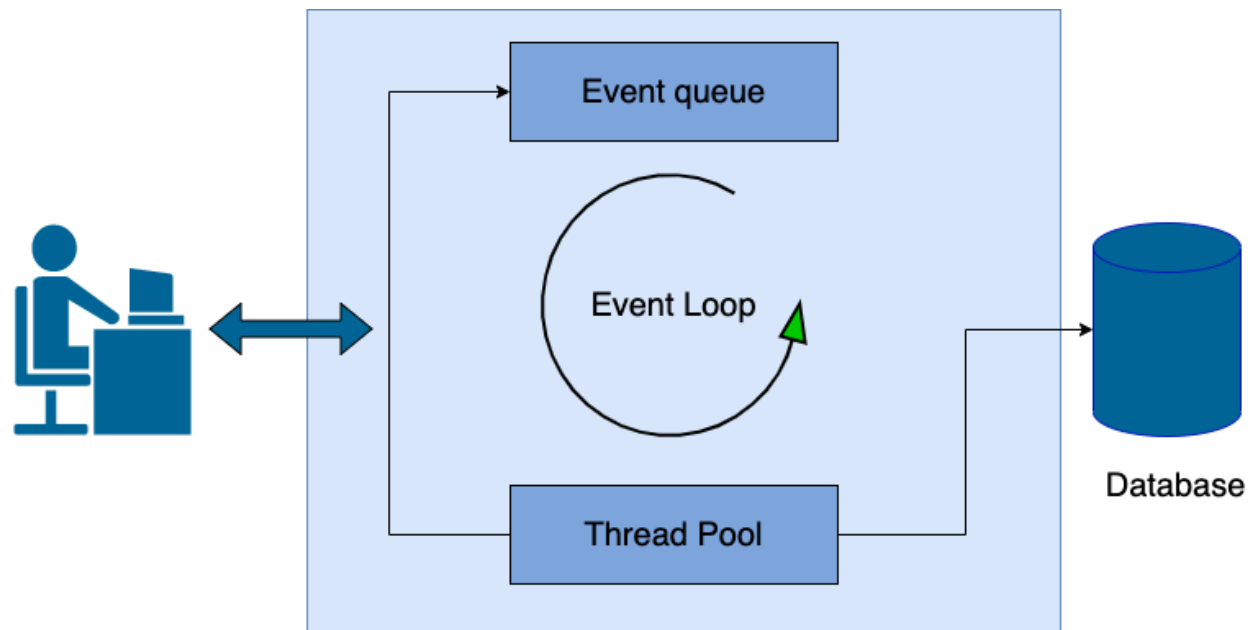
What is node.js?

- Node.js is an open-source and cross-platform JavaScript runtime environment.
- Developed around 2009 based on google's v8 engine for javascript and npm was created
- In 2010 express, a framework for node, was launched
- Larger companies started adopting Node.js: LinkedIn, Uber,etc
- We can say that Nodejs is:
 - Event driven
 - I/O driven
 - Non-Blocking
 - Asynchronous,that runs on an event loop.



- Free License
- Multi-Tasking
- High Performance
- Cross Platform
- Fast Processing

Parts of the Node.js Architecture:



- Requests
 - Incoming requests can be blocking (complex) or non-blocking (simple), depending upon the tasks that a user wants to perform in a web application
- Node.js Server
 - Node.js server is a server-side platform that takes requests from users, processes those requests, and returns responses to the corresponding users
- Event Queue

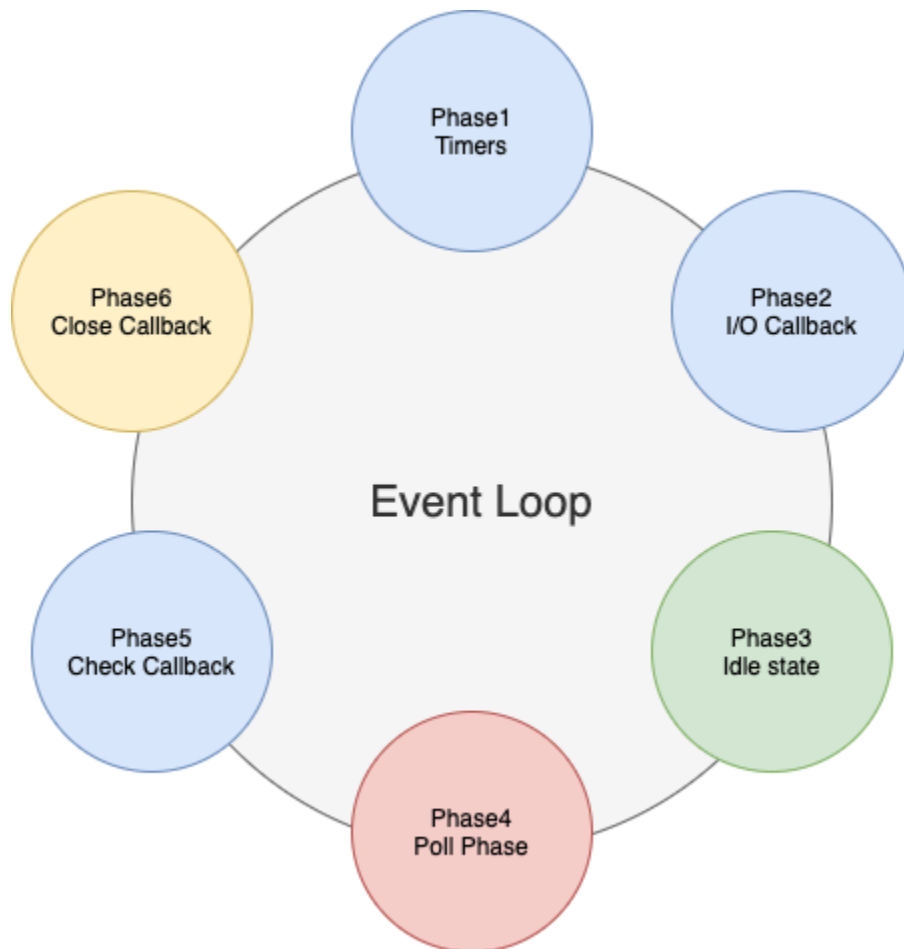
- Event Queue in a Node.js server stores incoming client requests and passes those requests one-by-one into the Event Loop
- Thread Pool
 - Thread pool consists of all the threads available for carrying out some tasks that might be required to fulfill client requests
- Event Loop
 - Event Loop indefinitely receives requests and processes them, and then returns the responses to corresponding clients
- External Resources
 - External resources are required to deal with blocking client requests. These resources can be for computation, data storage, etc.

Event Loop

- Event loops allow Nodejs to perform non-blocking operations.
- These 6 phases create one cycle.
 - Timer:
 - execute callbacks after a set period of time are provided by a timer module.
 - I/O callback(pending callback):
 - This phase executes callbacks for some system operations such as errors.
 - Idle:
 - In this phase, the event loop does nothing .It is in an idle state and preparing to go to the next phase.
 - Poll phase:
 - In this phase where all the javascript code we write is executed
 - During this phase the event loop is managing the I/O workload,calling the function in the queue until the queue is empty.
 - Check callback:
 - This phase runs when the poll phase becomes idle.
 - The event loop will never come in this phase when there is no callback remaining to be executed in the poll phase
 - Closing callback:
 - In this phase the event loop executes the callback.

NextTick :

- NextTick Allow users to handle errors, cleanup any then unnecessary resources, or perhaps try the request again



Blocking and Non-Blocking

In the example below there is a queue to buy a ticket only after the first person gets his task done, the second person gets to the counter and so on.

We can say they are in a blocking state as unless one task isn't completed second task would be executed

Longer time would be taken for all the people to get their ticket as only single counter is present

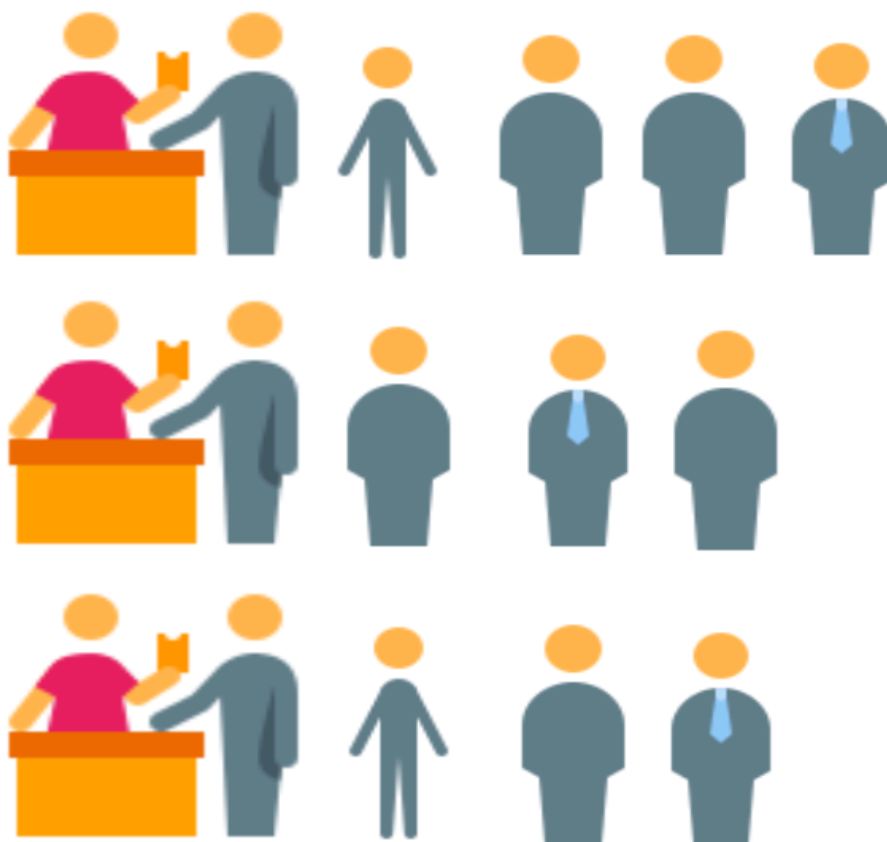


Comparing with the previous example as there are more than one counter more tickets can be sold

It would take less amount of time and much more tickets can be sold to the larger amount of people

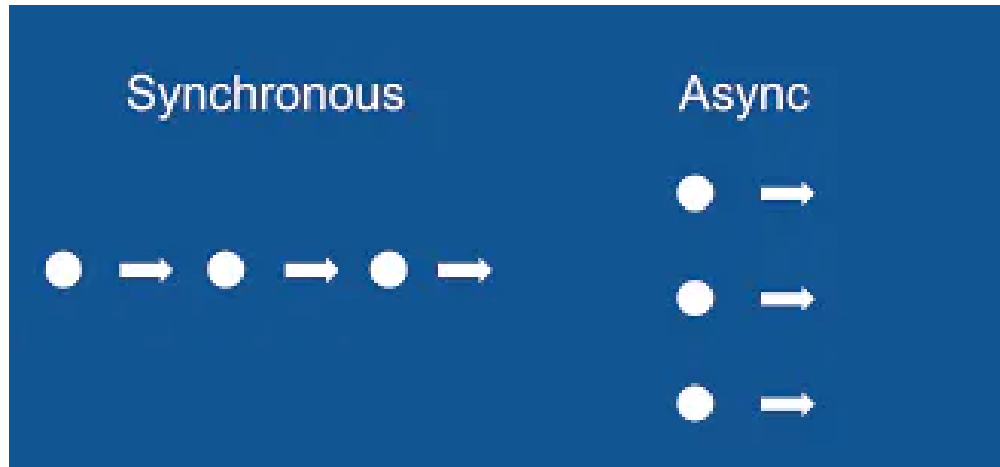
Even if one counter gets closed other two counter would still be working and selling the tickets

Would take more cost to serve multiple counters at once compared to a single counter but can perform more heavy task of selling tickets to huge amount of people where a single counter might end up choking

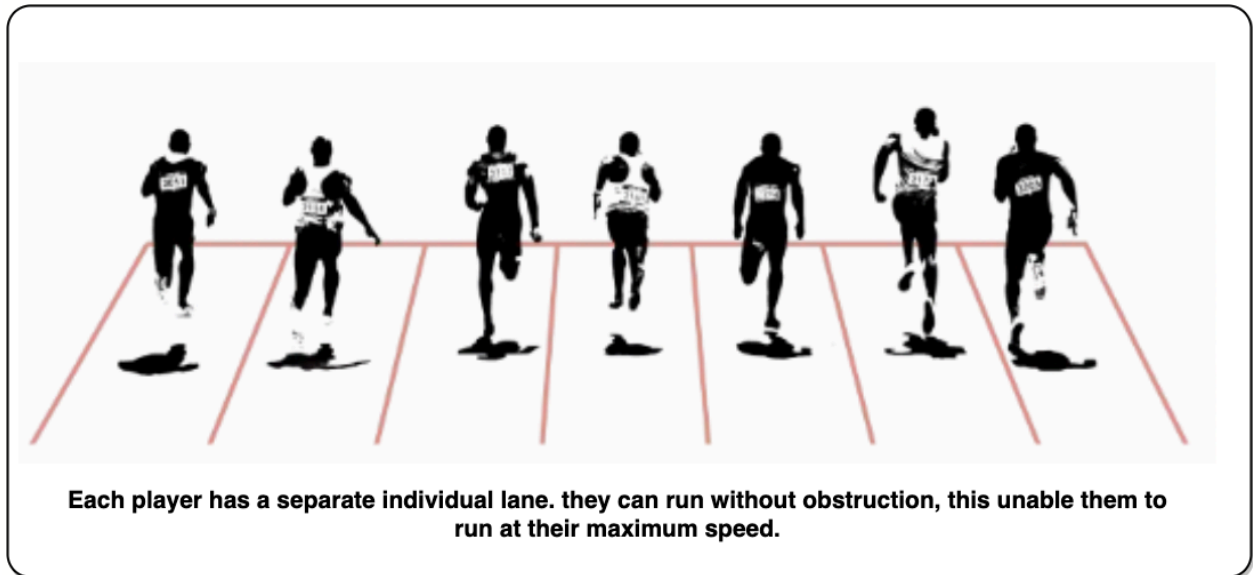


Synchronous Vs Asynchronous

- Synchronous and Asynchronous code are used for handling concurrency in JavaScript
- Synchronous are blocking and Asynchronous are non-blocking
- Nodejs uses asynchronous calls to execute the code



Single Threaded VS Multi Threaded



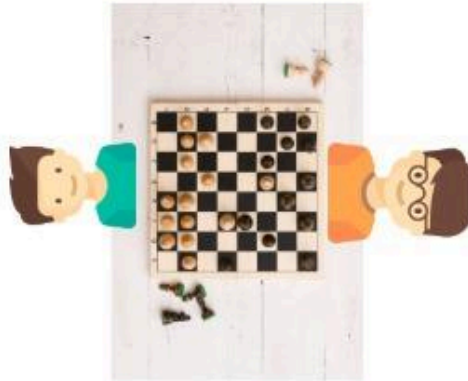
Non-blocking Single thread

Multithread Example

CARROM



CHESS



DART THROW



TABLE TENNIS



All games played in one room

I/O driven/ Event Driven



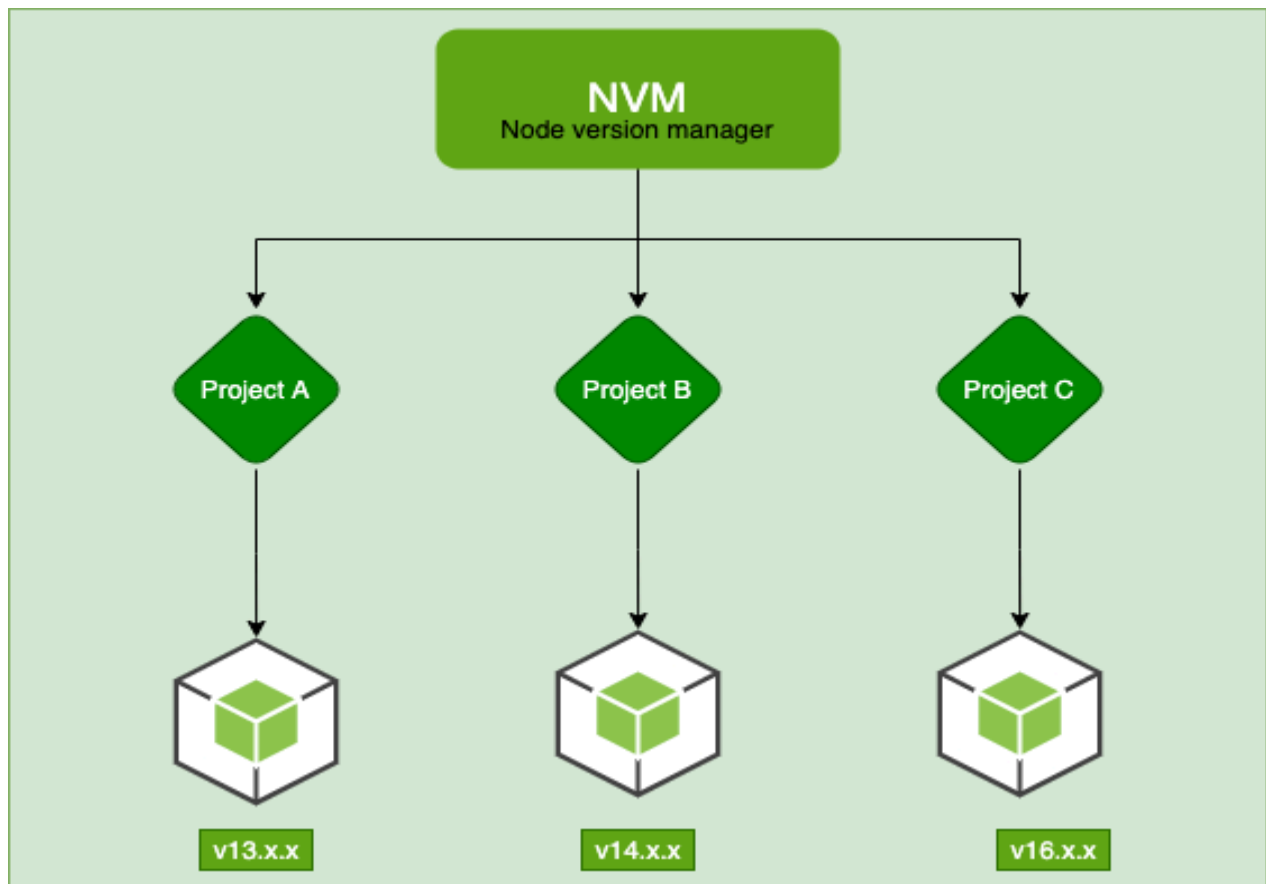
Cash is debited after you put the card in the machine



Match stops at when the referee blows the whistle

NVM

- Node Version Manager helps us to manage the versions of nodes that we need to use for our projects.
- The Node.js platform and community of tools and libraries are fast moving targets, and what might work under one Node.js version will not be guaranteed to work for another version of Node.js.
- For this reason, you need a way to not only install Node.js easily, but also switch between Node.js versions.
- NVM enables us to install different versions of Node.js and switch between them seamlessly



NVM:

1) `curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh`
`| bash`

2) `wget -qO-`
`https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash`
install nvm using first or second
after that

`cd ~`

2. `ls -la`

check for file called `.profile` then copy below code in `.profile` file
(optional if doesn't exist)

3. `touch .profile` (if `.profile` doesn't exist creates a file called `.profile`)

4. `cat > .profile`

```
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s  
"${HOME}/.nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
```

NPM

- Node Package Manager is used to install node packages into our projects.
- Node packages are basically a folder containing the code you need and you can either install it locally or globally
- Local installation:
 - Locally installed packages are added to the project directory
 - When we install new packages a new folder called “node_modules” is created and the packages are installed in it
 - To install local packages

```
$ npm install [package-name]
```

- Global installation:
 - Globally installed packages are packages that can be accessed into any project
 - They are installed outside project folder in the node directory
 - To install global packages

```
$ npm install [package-name] -g
```

- There are over a million packages available to use
- This saves time and helps to create an application in much lesser time

File system modules in node js:

Node provides an in-built module called fs (filesystem module.)it used for handle file operation

Like creating,reading,deleting files etc.

(fs) module can be imported using the following syntax

```
var fs = require('fs')
```

1. fs.readFile() -

- The fs.readFile() method which is used to read the file.
- This method reads the entire file into a buffer.
- The method accepts three parameters.
- Syntax:

```
fs.readFile( filename, encoding, callback_function )
```

2. fs.readFileSync()

- The fs.readFileSync() method which is used to read the file and return its content.
- In the fs.readFile(). fs.readFileSync() method, we can read files synchronously.
- When the fs.readFileSync() method is called the node program stops executing, and node waits for the fs.readFileSync() function to get executed
- After getting the result of the method the remaining node program is executed.
- Syntax:

```
fs.readFileSync( path, options )
```


3. fs.writeFile()

- The fs.writeFile() method is used to asynchronously write the specified data to a file.
- By default, the file would be replaced if it exists.
- The 'options' parameter can be used to modify the functionality of the method.
- Syntax:

```
fs.writeFile( file, data, options, callback )
```

4. WriteFileSync

- The fs.writeFileSync() is a synchronous method.
- The fs.writeFileSync() creates a new file if the specified file does not exist.
- Syntax:

```
fs.writeFileSync( file, data, options )
```

5. appendFile()

- The fs.appendFile() method is used to asynchronously append the given data to a file.
- A new file is created if it does not exist.
- The options parameter can be used to modify the behavior of the operation.
- Syntax:

```
fs.appendFile( path, data[, options], callback )
```

6. fs.rename():-

- The fs.rename() method is used to asynchronously rename a file at the given old path to a given new path.
- Syntax:

```
fs.rename( oldPath, newPath, callback )
```

7. fs.readdir():-

- The fs.readdir() method is used to asynchronously read the information of a given directory.
- The callback of this method returns an array of all the file names in the directory.
- Syntax:

```
fs.readdir( path, options, callback )
```

8. fs.readdirSync()

- The fs.readdirSync() method is used to synchronously read the information of a given directory.
- The method returns an array with all the file names or objects in the directory.
- Syntax:

```
fs.readdirSync( path, options )
```

Express framework

- Express JS is the pre-built Node JS framework
- Express helps developers build faster and smarter websites and web apps
- Steps to create an Express application:
 - Change to your project directory.
 - Install express-generator

```
npm install express-generator -g
```

- Create a project using express <project-name>eg:
express first-api
- Navigate to your project using

```
install npm
```

- This will install all the required packages and modules for the project to run inside node_modules directory
- We can view the project running in the browser locally hosted

```
http://localhost:3000
```

- Response:
 - `res.json()`
 - It sends a json response. This method sends a response that is the parameter converted
 - `res.send()`
 - Send the response ,the parameter can be string, object, boolean or an array.

- Query parameter:

- The query string portion of the url is the part of the URL after the question mark “?”.

`localhost:3000/yash.in/home?value=42`

- Each key and value is called a query parameter.
- If query string has multiple query parameters, they're separated by “&”.

`localhost:3000/yash.in/home?name=age&value=23`

- The string has 2 query para age and 23.
- Express automatically parses query parameters for us store them on request object as `res.query`
- If query parameters appear multiple times in a query string, express will group the values into an array.

`localhost:3000/yash.in/home?name=yash&name=pallavi`

Express will set `req.query.name=["yash", "pallavi"]` into an array.

- “ ? “ initialize the query string key/value.
- “ = “ assign the key to the value.
- “ & “ separate each key/value pair.

- Importing a Module:
 - Node Platform has provided a function call “require()” to import one module into another.

```
const <variable-name> = require("module path");
```

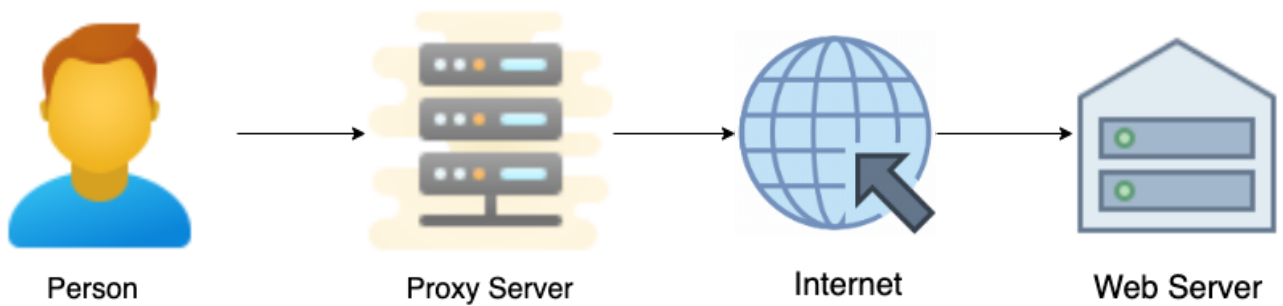
- Exporting a Module:
 - Node “exports” object is used to export a Node module so that other modules can use it.

```
module.exports;= <function name>.
```

```
exports.<function-name> =<function-name>
```

Proxy server:

- A proxy server acts as a gateway between you and the internet.
- A proxy server is a bridge between you and the rest of the internet. Normally, when you
- use your browser to surf the internet, you connect directly to the website you're visiting.
- Proxies communicate with websites on your behalf.



Deploying a project

- In our project first initialize git repo (using `git init`)
- Create a `.gitignore` file (in our project). → in that file add data . this is a file that we don't want to push in the server(`.*` , `~*` , `node_module*` , `data*`)
- Delete the contained of data folder and (
- Create `.gitkeep` file (solve problem of Git not pushing empty folders to remote)
- In gitlab create new project (black project)
- Add remote in local machine (terminal)
- `.gitignore` and `data/.gitkeep` forcefully add and commit and push
- In our project in terminal first check git status -> `git add (file)` -> `commit` -> `push`
- If you get access denied (then add ssh key in local)
- In server site (item2) login to your server account
- And clone the project in server
- If you get access denied then you have to generate key first and add that key
 - Go to `.ssh` (`cd .ssh`)
 - Create key (`ssh-keygen`)
 - Then enter the name of the key (like:gitlab, server etc.)
 - In the `ssh` folder `pub` and `private` key are generated .
 - Copy the public key (`cat file_name`)
 - Go to gitlab -> go to edit profile -> then add that copy(ssh key) \
 - In server side -> for adding ssh key we need to follow -> `ssh-agent /bin/bash` ->after that we can add ssh key (`ssh-add name key`)
- In server side -> clone the data .json file(country.json) → move the data json file in data folder
- For running your project → you need to install `nvm` →

curl -O-

<https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh> |
bash

- Copy the export 2 line and paste into (.profile) ->
- Exit the server (exit)
- After login again → check the nvm version(nvm ls)
- Then install node version using (nvm install v14.17.3)
- In our project (node-module are get added)
- To run the project we have to install forever package (npm install forever -g) -g means globally
- Then run the project (DATA-DIR=data forever bin/www)
- For checking the status we use (forever list)

Do's and Don'ts

- Code should be expressive, i.e. variables have meaningful names. (to avoid confusion.)
- use `.toString()` as below before using any String method on variables.(to avoid errors.)name variables in lowercase, if they have multiple words use camelCase naming.
- use underscore (`_`) for private variables.
- const variables that are hard coded or who's value won't change using capital letters.
- avoid using reset commands.
- Use the literal method to create an array. use `[]` instead of `new Array()`.
- Use `push()` method for adding elements, adding elements using indexes can create holes in an array.
- always use `+` sign rather than using `parseInt` and `Number` constructor.
- always convert to boolean for safeguard.
- Always save the file before committing.
- If function is passed as parameter check function and also create empty function incase if function is not passed as parameter.
- always do git status before committing and adding a file.(especially for conflict).
- if variable value will be updated in the future use `let` keyword otherwise use `const` keyword always.
- Give the commit proper message for better understandability.
- use git reset commands only if you have not pushed the changes. Otherwise make a new commit.
- `===` used to check `typeof`
- Use `null` to assign an “empty” or “unknown” value.
- whenever we do an if statement explicitly typecast all the values inside the if condition to make sure your code works in all the engine.
- Whenever we pass a parameter to function we need to make a local copy of it, if it is NP so that we don't pollute the NP variable.
- Always write code in proper formatting.(prettier).

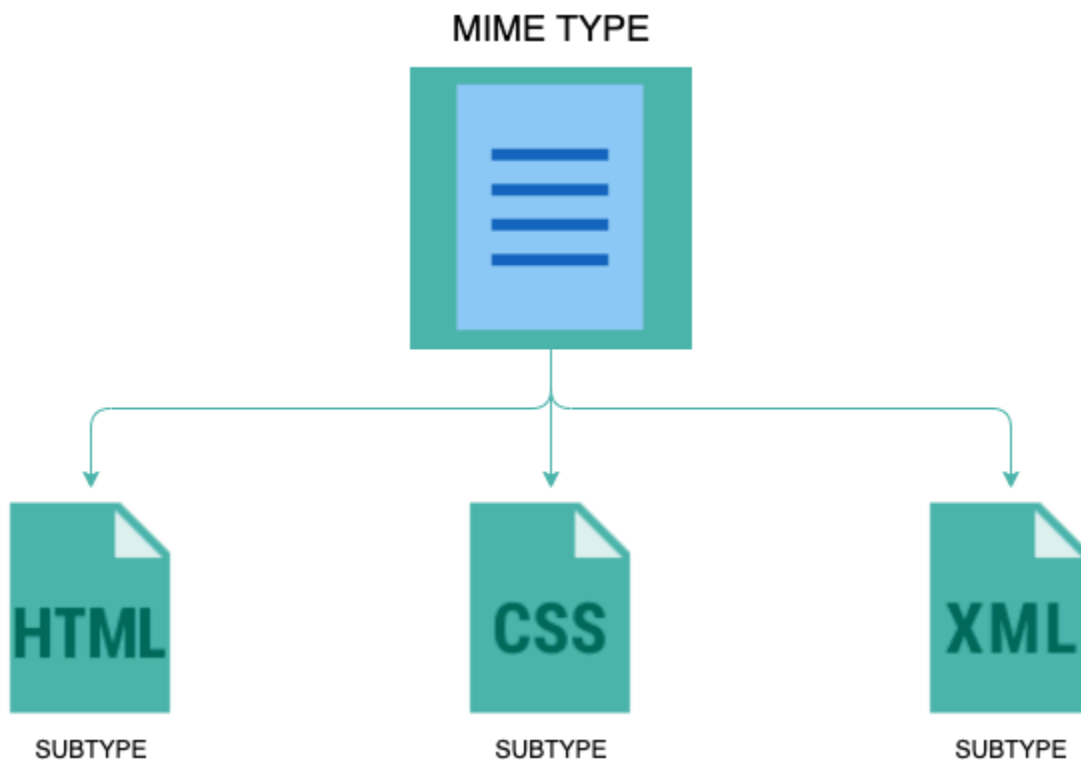
- Constant names should be all UPPER_CASE, separated by underscores.(environment var).
 - put all declarations at the top.
 - if data is dynamic always use an array and if data is static use an object(lookup).
 - Use console.log for Debugging Purposes.
 - Always use CLI
 - Always send merge requests on UI rather than locally (GIT).
 - Always use forEach instead of for loop.(avoid using for loop).
 - Put semicolon ; before IIFE
 - Use spread operator
 - Use cat
-
- Avoid using git add . Always give the name which file you want to add
 - Conflict should not be solved in the main branch or the branch that is shared with others as it will pollute the shared branch.
 - Don't skip the SemiColon ';'.
 - == only used to check values.
 - We can assign undefined values to variables but it is not recommended to do so. Undefined is used internally by the javascript engine.
 - Don't use Boolean,Function,Number,constructor.
 - don't use keywords as variable names.
 - we don't add a data folder into the git code repo.
 - Avoid using the UI.
 - Avoid coding with complex nested callbacks.

12 principles of application development.

1. Version control
2. Configuration
3. Dependency
4. Concurrency
5. Log
6. Separate data
7. Boot shutdown
8. Build release run
9. Monitoring
10. Independent process
11. Development and deployment env should be same
12. Port should not be hard coded

MimeType:

- Mime type is a standard that indicates the nature and format of a document or file.
- Mime(Multipurpose Internet Mail Extension) type is also referred to as media type.
- A mime type consist of a type and a subtype
type/subtype
- Type refers to the general category of the data type.
- Subtype refers to the specific type of data the type represents.

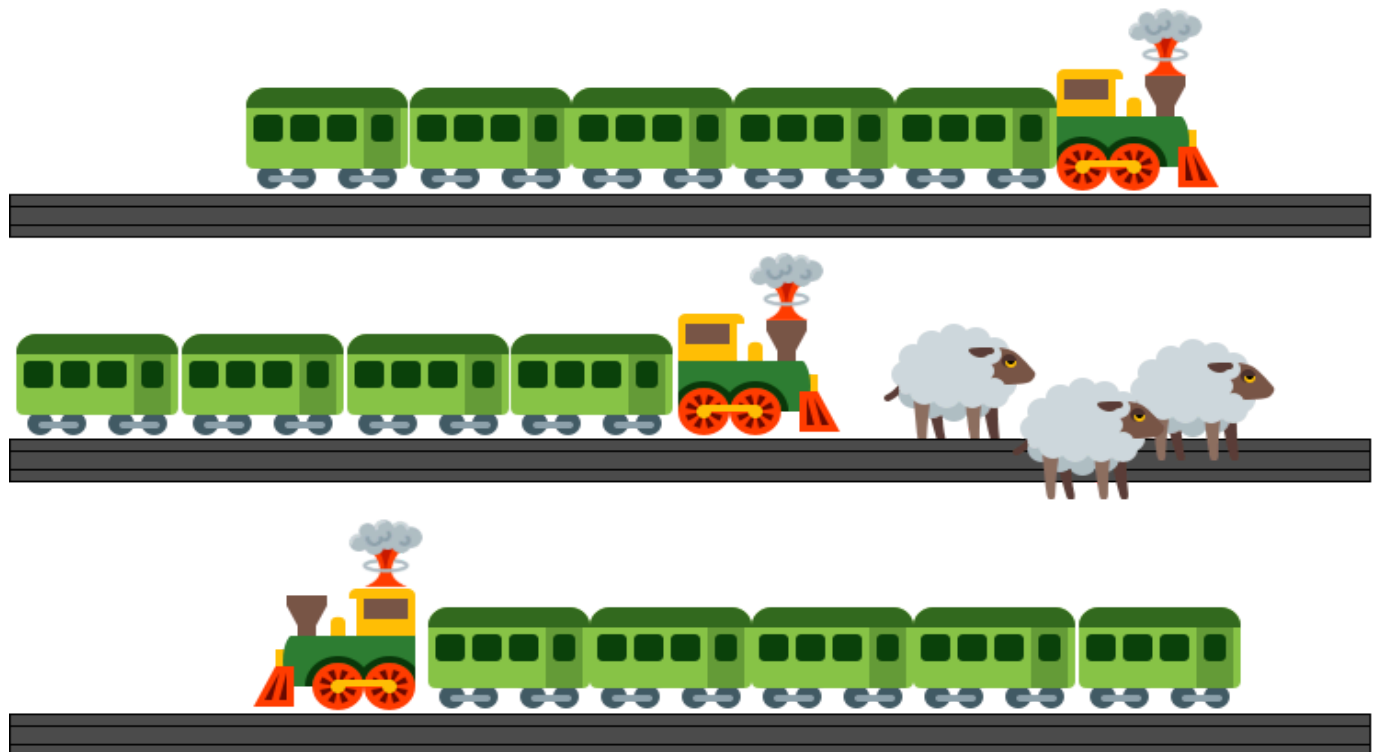


Monolithic and Microservice Architecture

- Monolithic architecture refers to building one large system or an application with a single code base.
- All the services are dependent on each other for their functionality
- If one service falls or crashes every other service might end up crashing or stopping to work
- It has a shared database for the entire system
- It is difficult to change technology or language or framework because everything is tightly coupled and depend on each other
- Below is an example of how monolithic services works:
 - In a single track, if the engine of the trains stops or crashes due to some reason all rest of the boogeys end up stopping as well, as they are dependent on each other to move ahead only after the bogey in front or the engine in front moves ahead.



- Microservice architecture refers to an architecture where services are built as individual services that run independently
- It is built as small independent module based system on business functionality
- As the services are independent of each other it is easy to upgrade technology or language or framework
- In microservices each module or service has their own separate database
- As each service are independent it is easier and takes much less time to upgrade and develop



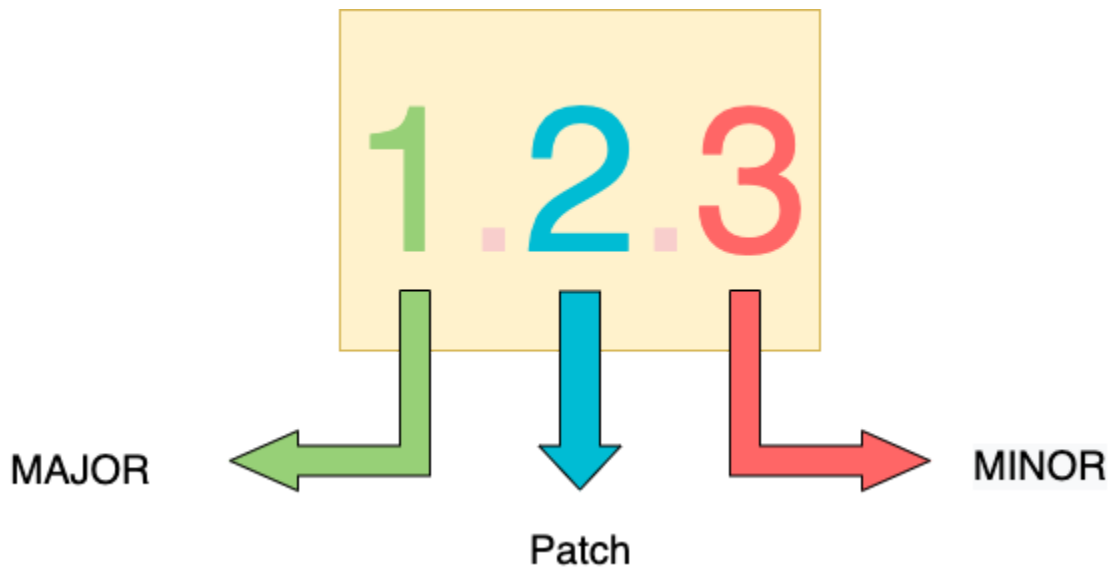
Middle-wares

- MULTER:
 - Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files
 - Multer adds a body object and a file object to the request object
 - Multer will not process any form which is not mimetype (type/subtype)
 - `$ npm install --save multer`
 - Usage:
 - `var multer = require('multer');`
 - `var upload = multer({ dest: 'path/' });`

- APPLICATION-LEVEL MIDDLEWARE:
 - This middleware function is bound to the instance of the app object by using `app.use()` or `app`.
 - METHOD functions where the method is the HTTP method of the request
 - The snippets above are all application-level middleware since they are bound to instances of the app object.
- ROUTER-LEVEL MIDDLEWARE:
 - It is very similar to the application level middleware except that
 - it is bound to an instance of the `express.router()` function.

- ERROR-HANDLING MIDDLEWARE:
 - ExpressJs has default error-handling parameters, these are error-handling functions defined in the same way we define normal Application/Router level middleware functions except that they have four arguments instead of three.
 - The fourth one being the “error” arguments! Error-handling middleware must have this fourth function as it helps to identify it as an error-handling middleware.

Semver versioning



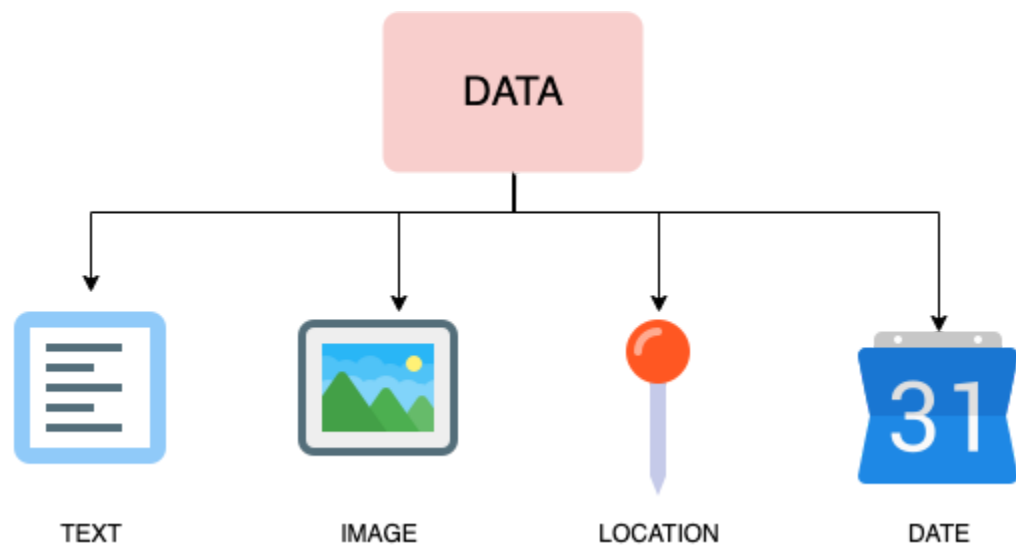
- Versions are like age it increases with as every day passes. Version is a way to categorize the software and its build
- Different versions of an applications might use different version of modules as well
- As new modules and services are developed and upgraded everyday it is difficult to keep track of what software built was suitable for the application
- That's where version comes into picture
- Every software or applications that are released, comes out with a specific version number
- With every update or upgrade the version number is incremented
- A simple set of rules and requirements that dictate how version numbers are assigned and incremented based of common practices used in both closed and open source software

- Given a version number MAJOR.MINOR.PATCH, increment the:
 1. MAJOR version when you make incompatible API changes,
 2. MINOR version when you add functionality in a backwards compatible manner, and
 3. PATCH version when you make backwards compatible bug fixes.

DATABASE

- Data:

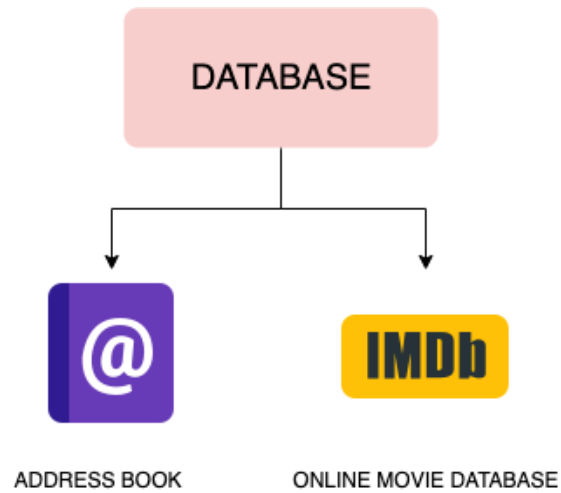
- Data can be any information that you store to access it in the future. It can be numeric, text, media, location, date, etc.
- This can be written on paper, can be saved on a computer hard-drive, or even stored in the cloud.



- Database:

- The collection of multiple data entries together creates a database.
- Databases are usually created so that users can access a large chunk of data and perform certain operations on it altogether.
- A Database makes it easy for different users to access data sets
- We can store a similar type of information for each data entry in a database
- As websites are becoming more user interactive and larger in size, data of the users, customers, orders, etc. are important

assets to the companies, thus need to have a scalable and reliable database



- SQL commands categorized into four categories as:
 - DDL – Data Definition Language
 - DQL – Data Query Language
 - DML – Data Manipulation Language
 - DCL – Data Control Language
- DDL(Data Definition Language) :
 - Data Definition Language consists of the SQL commands that can be used to define the database schema.
 - It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.
 - CREATE –
 - It is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
 - DROP –
 - It is used to delete objects from the database.
 - ALTER -
 - It is used to alter the structure of the database.
 - TRUNCATE–
 - It is used to remove all records from a table, including all spaces allocated for the records.
 - COMMENT –
 - It is used to add comments to the data dictionary.
 - RENAME –
 - It is used to rename an object existing in the database.
- DQL (Data Query Language) :
 - DQL statements are used for performing queries on the data within schema objects.
 - The purpose of the DQL Command is to get some schema relation based on the query passed to it.
 - Example of DQL:
 - SELECT – is used to retrieve data from the database.

- DML(Data Manipulation Language):
 - The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.
 - Examples of DML:
 - INSERT – is used to insert data into a table.
 - UPDATE – is used to update existing data within a table.
 - DELETE – is used to delete records from a database table.
- DCL(Data Control Language):
 - DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system.
 - Examples of DCL commands:
 - GRANT- gives users access privileges to the database.
 - REVOKE - withdraw user's access privileges given by using the GRANT command.

SQL Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

```
CREATE TABLE table_name (  
    column1 data_type constraint,  
    column2 data_type constraint,  
    column3 data_type constraint,  
    ....  
);
```

The following constraints are commonly used in SQL:

- 1) NOT NULL : Ensures that a column cannot have a NULL value
- 2) UNIQUE : Ensures that all values in a column are different. It is used to avoid duplicates while inserting the data into the table.
- 3) PRIMARY KEY :
 - A combination of NOT NULL and UNIQUE.
 - Uniquely identifies each row in a table.
- 4) FOREIGN KEY :
 - Prevents actions that would destroy links between tables.
 - It can accept duplicate values.
 - It can have null values.
 - If a column wants to become a foreign key , it should be the primary key of its own table.
 - Foreign key is also known as Referral Integrity constraint.
- 5) CHECK : Ensures that the values in a column satisfies a specific condition.
- 6) DEFAULT : Sets a default value for a column if no value is specified.
- 7) CREATE INDEX : Used to create and retrieve data from the database very quickly

ASYNC Module

- Async is a utility module which provides functions for working with asynchronous JavaScript.
- A collection of async functions for manipulating collections, such as arrays and objects.

- Methods

- map:

```
async.map(['file1', 'file2', 'file3'], fs.stat, function (err, results) {  
    // results is now an array of stats for each file  
});
```

- filter:

```
async.filter(['file1', 'file2', 'file3'], function (filePath, callback) {  
    fs.access(filePath, function (err) {  
        callback(null, !err)  
    });  
}, function (err, results) {  
    // results now equals an array of the existing files  
})
```

- groupby:

```
var ungrouped = [{  
    name: 'Alice',  
    age: 12  
},  
{  
    name: 'Bob',
```

```

    age: 12
  },
  {
    name: 'Carol',
    age: 13
  },
  {
    name: 'Carol',
    age: 45
  }
]

```

```

function getName(person, cb) {
  cb(null, person.name);
}

```

```

async.groupBy(ungrouped, getName, function (err,
results) {
  if (err) {
    return console.log(err);
  }
  console.log(results);
});

```

- mapLimit:

```

async.mapLimit(['1','2','3','4'], 2, function(num, callback){
  setTimeout(function(){
    console.log(num);
    callback(null, num);
  },
2000);

},function(err, results){

```

```
console.log(results);  
});
```

ASYNC Control Flow

- Parallel:

- Parallel tasks mean running many functions at the same time without waiting for the previous functions to complete.
- Once these tasks are completed, their results are passed to a main callback that returns an array of results.
- Syntax:

```
async.parallel(tasks, callback)
```

- Series:

- Async series is used to run a collection of task executions in a sequence.
- These tasks do not depend on the results of the previous task.
- If any function within the series functions returns an error to its callback, the whole series stops.
- No more functions will be executed. The final callback will immediately be called with the error.
- Syntax:

```
async.mapSeries(['1','2','3','4','5'], function(num, callback){
  setTimeout(function(){
    num = num * 2,
    console.log(num);
    callback(null, num);
  },
  1000);
},function(err, results){
  console.log(results);
});
```

- Waterfall :

- A waterfall runs an array of multiple asynchronous task functions in series.
- In the waterfall model the output of the first function should be the input of the second function.
- The second function is dependent on the first function.
- Syntax:

```
async.waterfall([
  function first(cb) {
    setTimeout(() => {
      console.log("first function");
      cb(null, "first function result")
    }, 2000);
  },
  function second(arg1, cb) {
    setTimeout(() => {
      console.log("second function");
      cb(null, "second function result")
    }, 2000);
  },
  function third(arg2, cb) {
    setTimeout(() => {
      console.log("third function");
      cb(null, "third function result")
    }, 2000);
  }
],
function (err, result) {
  if (err) {
    console.log("error in some function ", err);
  }
  console.log("result", result);
});
```

- Auto :
 - Async auto helps us to manage control flow of applications.
 - If we have many functions that run in a certain order and some function depends on the result of the previous function.
 - So async auto is a great solution for that .

```

async.auto({
  get_data: function (callback) {
    console.log('get data');
    callback(null,'data');
  },
  make_folder: function (callback) {
    console.log('make folder');
    callback(null,{});
  },
  write_file: ['get_data','make_folder', function
(result,callback) {
    console.log("write",result);
    callback(null,'write_file');
  }],
  email_link :['write_file',function (result,callback) {
    console.log("final task", result);
    callback(null,{file:result.write_file,
'email':'abc@ajs.com'})];
  }],
  function (err,result) {
    console.log(result);
  })
})

```

Shortcuts:

```
show create TABLE *table_name*
```