Basic Workflow -

To create a version

1.) git init - Create a new empty git repository.
or reinitialize existing one

file1.js

working directory

> directory which contains the working directory along with metadata will be in

- .git
  - objects
  - HEAD
  - info
  - refs
  - hooks

> git status - Get the current status of repository.

> git status -s - Get status with short style (?? file.js)

git add → ?? → The file is untracked (the file is not known to the repository or the repository has not yet created any version of this file.

Staging Area - A temporary collection before you add your file to the repository.

> git add ____ → to staging area

> git status → working directory + staging area → A file.js → No ??
the changes to the file are added to the staging area.

to repository : git commit -m " "

→ new version will be recorded by the repository

- any version in repository will contain
  - author
  - date and time
  - message while committing
  - actual changes

> git commit → saves all currently staged changes of the project.

M_ file.js → file modified and present in staging area.

_M file.js → file in working directory

> git log → get all the commit logs

only works on the committed history.

-- online

-- colour

-- graph

get the diff" of current version of a file with last recorded version.

> git diff

> git checkout <file> → Remove all the changes till the last commit.

You cannot retrieve the changes

If file is in staging area, we can't use checkout

to use

Unstage

Unstaging → > git reset → git staging area to working directory.

> git checkout <file>

Unstage the changes and remove them immediately.

> git reset --hard = unstage + checkout (along with all the changes in the working directory)

Version ← Author, committer, Version info

Date Page

Git Repository -

- HEAD
- config
- description
- hooks
- info
- objects
- refs
  - heads
  - tags

For git → file
for us → object

Every commit is identified by a unique identifier.

eg: 02 2ddf2

02 → Directory name (To see internal)

2ddf2 → file name

> git cat-file -p <file obj id>

→ On commit, git creates the objects (file) in this directory.

4 types of object:

- commit object → created per commit → tree object id, author, committer, commit message.
- tree object → created per commit per directory → 100644: the entry is pointing to a file blob object id, file name: which have contents
- blob object → created per file → it contains content of a file in encrypted format.

During git commit [At the time of git add ←

everytime we commit, three objects are created.

→ id's here are in the form of hash value.

After you make changes to the file and commit, the commit object contains parent id which is the previous version object id.

ort strategy

.git
-refs
-heads
-main → points to latest commit id.

Branching

- Branch is a file under .git/refs/heads with branch name
- which stores the latest commit id.
- The default branch name is either main or master.
  ↓ new version of git

Get list of branches → git branch.

Create a branch → git branch <branchname> → main and another branch would us have the same id.

git branch

b1 → The branch you are, at is stored under .git/HEAD.
* main
↳ you are on the branch main

> git checkout -b <branch>

Switch Branch – git switch / git checkout <branch name>
↓ multipurpose

Merging → ek branch ka changes dusre mai loona.

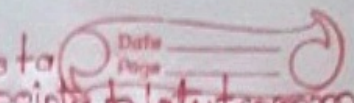
1.) Merge b1 in main
> git checkout main → we are in main branch → git status → show the branch name
> git merge b1

Delete a branch –
> git branch -d <branch name>

git push origin --delete <branch-name>

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

man git



→ Head can also directly to a commit instead of a branch.

> git checkout ____
commit id

> git branch b2

> git ~~branch~~ checkout main

conflict in merging → two branches modified the same file of the same line. conflict must be resolved manually.

Monolithic Architectures:

→ traditional model of a software program which is built as a unified unit.

everything contained in APP.

→ is a singular, large computing network

→ when code changes, you will have build the entire code base.

→ used when code base is small. This allows everything to be rebased at once.

Micro Services

→ series of independently deployable services.

→ every single service has own logic and database

→ every service has separate repository.

→ Updating, testing, deployment and scaling occurs within each service.

→ They do not reduce complexity. for management and scaling.

container runtime : for running the containers

containerd : Default container runtime

→ worker node

host pod → runs container : wrapper around one or more container

→

Kubernetes works only on Linux.

Cluster → Master, Worker

present in Both Master and Worker

Kubelet → agent that talks with master about what's going on in the worker.

→ for communication.

worker (kubelet) ↔ Master (Kubelet)

Kube Api Server

↓

exposes all the REST Api's

Database: etcd → cluster information
eg: master info

Kube-proxy → manages network communication b/w nodes.

Kube-scheduler

→ scheduling the pods created to the worker

Kube-controller-manager

→ control pods, node, service

* Common to both master and node

→ Kubelet, Kubeproxy, container Runtime

Services – provide a stable endpoint for accessing a set of pods. It routes traffic to the appropriate pods but does not create containers

containers managed by container runtime

Pod runs in Kubernetes
container runs in docker system

Create a single Node cluster

↓

download minikube executable.

add the line in ~/.bashrc
↓
alias ...
↓
Reload the rc
↓
source ~/.bashrc

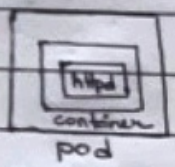
> minikube start
> minikube stop
> minikube destroy → delete the cluster
> minikube status

open Kubernetes GUI

minikube dashboard

Node:

list of node - kubectl get nodes
details of node - kubectl describe node <node id>



→ Pods runs a application inside a container.
→ represents a process running.

single container pod

Multi-container pod
→ to run multicontainer in pod.
initcontainer
sidecar

YAML

YAML Ain't Markup language

→ data serialization format.
→ readable by humans.

key-value : Mapping
Single value : Scalar
list of values : sequence → #pet
- Tiger
- cat

SOAP

YAML → just a syntax

name : sudh
age : 28
address : pune

Kubernetes object
↳ apiVersion → v1
↳ kind → type of object
↳ metadata →
↳ spec →

To create a POD -

apiVersion : v1
kind : Pod
metadata :
  name : pod1
  labels :
    type : mypod
spec :
  containers :
    - name : ~~nginx~~

↓

create a replica (services)

pod :

Ready ← > kubectl get pods
column shows the no. of containers > kubectl create -f ——— pod yaml file

> kubectl delete pod <podname>

ReplicaSet → For horizontal scaling.
↳ it ensures that
create pods

two pod ko manage krta hai

→ scaling of pods

↓

features	docker swarm	K8s
Load Balancer	service	service
Replication behaviour		

> kubectl get pod --watch

kubernetes.io → replicaSet

Replicaset → scaling pods

copy the syntax

rs1.yaml

- apiVersion: apps/v1
- kind: ReplicaSet
- metadata:
  name: rs1
  spec:
  replicas: 10
  selector:
  matchLabels:
  type: mypod
  template:
  metadata:
  name: pod1
  labels:
  type: mypod
  spec:
  containers:
  - name: containers1
  image: httpd.

pod creation

> kubectl get replicaset

> kubectl get rs

> kubectl create -f ____ file name

>