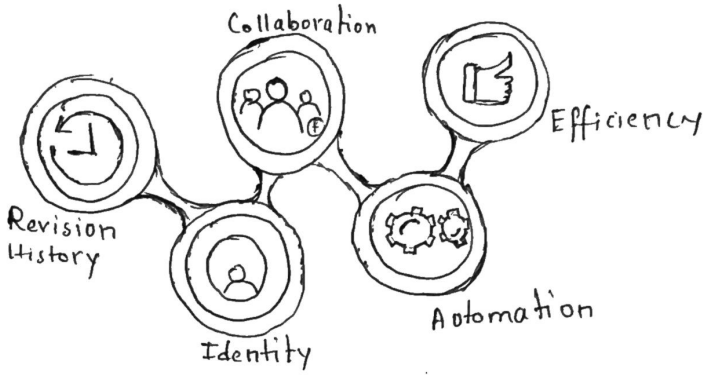
Version Control :- (Source Control, Source Code Mangement

A system that records all changes and modifications to ~~find~~ files for tracking purposes.

Chages in files

- Adding files
- Modifying / Updating files
- Deleting files

There are many benefits related to version control



Peer Reviews :- aims to get other developers on your team to review the code and provide feedback where necessary.

DevOps (Development Operations) :- is a set of practices, philosophies, and tools that increase an organisation's ability to deliver applications or services to a high quality or velocity

→ version control is key tool in this process

Agile Methodology

In Agile process, a team normally plan & execute two week of work to complete, which is called an iteration. Each iteration has a list of tasks to complete before the two week ends.

→ It used to achieve efficiency.

Microservices

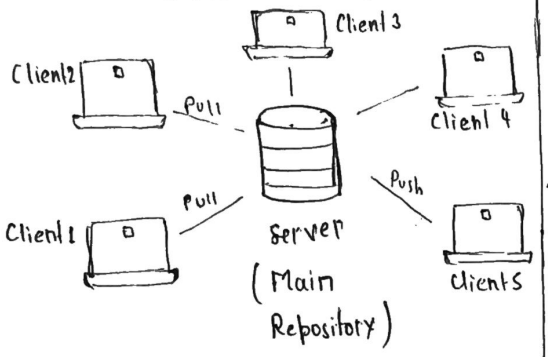
Git blame

Different Versions control systems :-

- Subversion
- Perforce
- AWS code Commit
- Mercurial
- Git

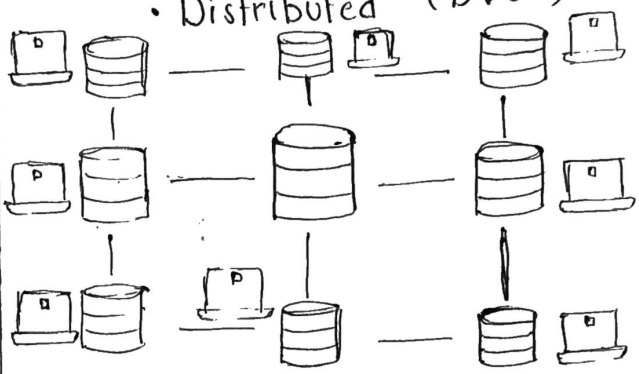
Version Control Systems can be split into two types or categories

- Centralized (CVCS)



- Contains all the history of the changes
- Every Operation needs to have a connection to the server itself.
- Server is the central copy of the project.
- advantage of CVCS is that they are considered easier to learn than their distributed Counter parts.
- Disadvantage is that they can be slower given that you need to establish a connection to the server to perform any action.

- Distributed (DVCS)



- Every User Essentially a server not a client
- You have entire history of changes on your local system
- It allows user to work in an offline state
- Speed and performance are also better than its CVCS counterpart.
- DVCS is a key ~~ope~~ factor in improving developer operations and software development lifecycle
- No server & client system exists

Procedures of Version Control in professional software Development :-

Work flow :-

- A good workflow will have a process for resolving conflicts.
- It should have a peer review system (where another developer must review code before it can be merged in)

Continuous Integration :-

CI is used to automate the integration of code changes from multiple developers into a single main stream

This process is widespread in test-driven software development strategies. It is often used to automatically compile the project and run tests on every code change to ensure that the build remain stable & prevent regression in functionality.

Continuous Delivery :-

Continuous delivery is an extension of Continuous Integration Once the changes have been ~~made~~ merged into main stream, a CD system automatically packages the application and prepares it for deployment.

This helps avoid human error when packaging the application

Continuous Deployment :-

Conti. Deploy. is an extension of Continuous Delivery. The goal of Conti. Deploy. is to deploy and release software to customer frequently & safely.

The strategy commonly involves automatically deploying to a test (also known as staging) environment first to validate the deployment package and software changes.

Once validated, it can automatically deploy to the live (also known as production) environment for ~~to~~ customer.

Staging Vs. Production

- Development Environments
    - Developer Environment
    - UAT or QA Environment
    - Staging Environment
- Staging
- New Features
- Testing
- Migrations
- Configuration Changes
- Production
- Downtime
- Vulnerabilities
- Reputation

⇒ Revision history is a ~~document~~ record of all changes within a project. It allows you to pinpoint who made the changes, when they were made and what was changed

⇒ Centralise Version Control System

- ✓ Concurrent Version System
- ✗ Mercurial
- ✓ Subversion
- ✗ ~~GIT~~ Git

→ Staging Environment should mimic your production environment

The Command Line :-

cd
touch
multi
mkdir
history

`Code example.js` → (to open example.js in vs code)

Helper commands

Manual (man)

flag or (option)
ls -l → flag

Vim (Visual Editor improved)

Uses modes to determine commands you can work with

- Normal mode : Default mode
- Insert mode : Allows contents of files to be edited.
- Command line mode : Normal commands begin with:

Commands in vim

```
#!/bin/bash
echo "Hello world!"
```

→ save with .sh extension

to run
`./test-shell.sh`

---

link file

lrwxrwxrwx temp → temp

directory

drw------

standard file

-rw-r----

---

cd ssh

`wc file1.txt -w` - word count

~~`wc -c`~~

`wc file1.txt -l` - line count.

`mkdir -p dir1/dir2` - p flags for making parent directory

Pipes — first output will be input of other

`ls | wc -w` -

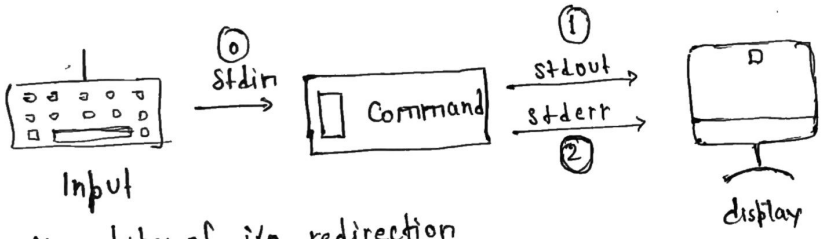
`cat file1.txt | wc -w` — for no. of words in file1.txt

`cat file1.txt file2.txt | wc -w` — for both files concatenated word count

`ls file1.txt | wc -w`

Redirection

- Shell keeps refrence of std. input, output & error by numbering system



three types of i/o redirection

Standard Input

- `<` - Input sign
- cat command

`cat > input.txt` (ctrl + D) → to come out of cat

`cat < input.txt`

Standard Output

- `>` - output sign

`ls -l > output.txt`

Standard Error

- `2>`
- `2>&1`

```
ls -l /bin/usr > error.txt
ls: cannot access '/bin/usr': No such file or directory
ls -l /bin/usr 2> error.txt
less error.txt
```

```
ls -l /bin/usr > error_output.txt 2>&1
less error_output.txt
```

Grep (Global Regular Expression)

```
grep som names.txt
grep -i sam names.txt      - case insensitive
grep -w sam names.txt      - partial match ignored
ls /bin | grep zip
```

Git - Version control system for tracking changes to projects.

- Fast
- Reliable
- Open-source
- Accessible

GitHub - GitHub is a Cloud-based hosting service that lets you manage Git repositories from a user interface.

Most Common Features of GitHub on top of Git

- access control
- pull requests
- automation

Installing Git on Windows

https://gitforwindows.org/

to check

```
$ git version
```

Connecting Github via https :-

Go to token.txt in documents/application on your ubuntu

Connecting Github via ssh

Go to sshkey.txt in _____ on your computer

$P = mv$

$V = u + at$

$s =$

$V = \frac{dx}{dt}$

$a = v\frac{dv}{dx}$

$a = \frac{dv}{dt}$

$v\,dt = dx$

$F \propto \frac{\Delta P}{\Delta t}$

$F = k\frac{\Delta P}{\Delta T}$

$= \frac{P_f - P_i}{\Delta T}$

$= \frac{mv - mu}{\Delta T}$

$= m\left(\frac{v-u}{\Delta T}\right)$

$= ma$

$V = \frac{dx}{dt}$

$\boxed{a = \frac{dv}{dt}}$

$F = \frac{\Delta P}{\Delta T}$

$0 = \frac{\Delta P}{\Delta T}$

$\Delta P = 0$

$P_f - P_i = 0$

$\boxed{P_f = P_i}$

Creating & Cloning a repository

1. create a repository (on the site)
2. Go to code
3. copy path in https
4. Go to cmd
5. Give command

   git clone <path> → paste it

try same with ssh path

How Git Works

After cloning four files will generate
two of them will be

1. .git - hidden folder to track all the changes
& 2. Readme.md

- .git folder will be initialized by following command

git init .

Git Workflow

three steps

Workflow:-

Working directory → staging area → committed files → remote repo

Add & Commit

~~to check~~

git status - to check any change or commit currently occured

~~gi~~

Git add

- Git Command that marks a file to be included as part of a code commit.

to stage a file

```
git add test.txt
git status
```

```
git restore --stage test.txt
```

→ to revert those changes

↓ to unstage

Git commit command

```
git commit -m "Adding a new file for testing"
```

↳ for message in double inverted commas

```
git config --global user.email "your@example.com"
git config --global user.name "your name"
```

→ for identity authentication (use only one of them)
→ omit --global for only this repository
→ ✲ use before commit

Push Commad — to add your ^commits to original repository (local)

To add new branch

```
git checkout -B feature/lesson
```

↳ name of ~~directories~~ new branch

another way

```
git branch feature/lesson
```

`git branch` — to check branches

⇒ now main branch has no indication or knowlegde of any change even if we push code to main repository

because

Git branches exists in isolation
Changes need to be merged back to main branch

Pull requests — for peer review / to validate that changes are correct.

~~when coding~~

```
touch test2.txt
git add test2.txt
git commit -m "Add test2.txt"
git push -u origin feature/lesson
```

↳ to only get update from main branch

password authentication has been removed so use Access token or ssh key.

[git checkoul main] -> to check or change directory

[git status]

[git pull] - to recieve latest changes

[ls]

Remote vs local

```
mkdir my-second-repo
cd my-second-repo
git init
git remote -v
```

```
cd ..
cd my-first-repo
git remote -v
origin git@github.com:Gil.
cd ..
ls -la
cd my-second-repo
git remote add origin git@github.com:GitTutorials101/my-first-repo.git
```

Push & Pull

```
gil status
git push origin main
```

⇒ When you perform a git Push command, it copies content from your local repository and then uses it to merge all of the content in the remote repository

to get the latest ~~repository~~ changes from repository

```
cat test.txt
git pull
cat test.txt
```

Error, fatal: bad boolean config value 'pink' for 'color.ui'

sol.-

```
git config --list
git config --global color.ui false
```

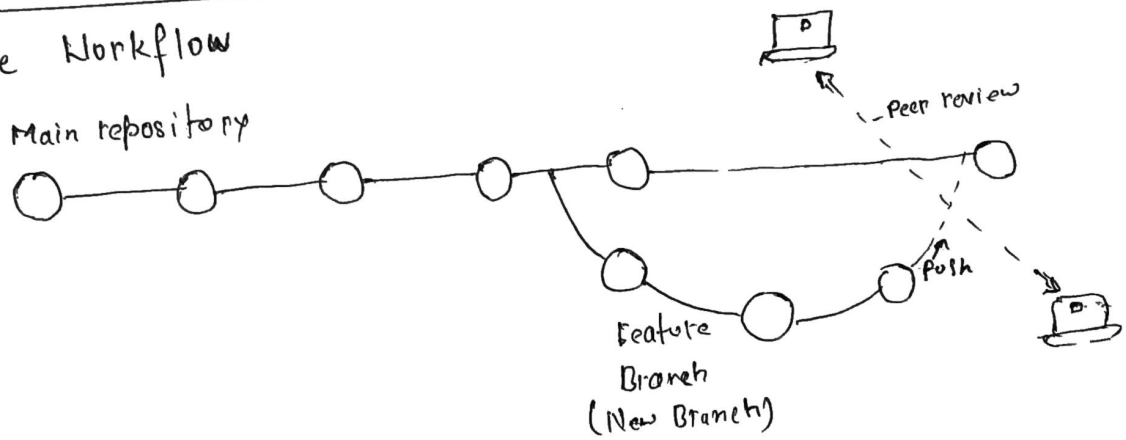
Github using Github CLI

1. gh auth login

? What account do you want to log into Github.com
? Github.com ——— HTTPS
? ——— Yes
? ——— authentication token

2. gh repo clone AshishAshishA/my-first-repo
3. cd my-first-repo
4. touch result.txt
5. ~~git~~ status
6. git add result.txt
7. git status
8. ~~git commit -m "success"~~
8. git config --global user.name "Ashish---A" (may email will not work here)
9. git commit -m "success"
10. git push

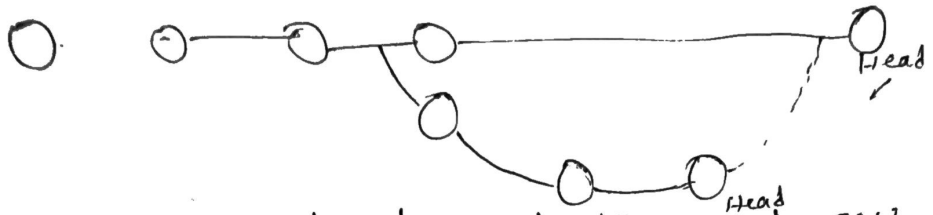
Example Workflow



git pull
git checkout -b feature/my-new-feature
git add .
git add README.md
git commit -m 'chore: created new branch with new README file'
git push -u origin feature/my-new-feature

Head

HEAD



After creating new branch Head will move to new branch from main branch.

Commands to demonstrate it

1. Cno to repository.
2. git checkout main (moving to main branch)
3. git branch (checking branch)
4. ~~git~~ cat .git/HEAD (checking HEAD)
5. git checkout feature/my-new-branch (moving to new branch)
6. cat .git/HEAD (again checking HEAD)
7. git checkout main (again moving to main)
8. cat .git/refs/heads/main (checking hash code) ~~if~~
9. nano README.me (~~changing~~ Rea Modifying README files)
10. cat .git/refs/heads/main (again ~~moving to main~~ checking has code)
    - will be same as previous
11. git status (checking status)
12. git add .
13. git commit -m "minor update"
14. cat .git/refs/head/main (again checking hash code)
    - will be diff than previous.
15.

Diff Commands :-

Git status ——— > which files were changed

Git diff ——— > what changes were made

→ Individual files, branches & commits can all be compared with Git diff

① Cro to repository

② git diff

③ git diff HEAD README.md

④ git log --pretty=oneline

⑤ git branch

⑥ git diff main feature/my-new-branch

Blame

→ git blame command is used to look at changes of a specific file and show the dates, times, and users who made the changes

git blame <date> <file>

git log

git blame Feature.js

Message format

90129342 (Developer1 2022-01-29 16:48:30 +0000 1) let add = (a,b) => {

Commit ID/ Hash | Author | Date | Time | Line number | Content

Mk docs → public repository

git blame setup.py (click q to exit)

git blame -L 5,25 setup.py

start Line, end Line

git blame -l setup.py

git log -p od7df639 - gives actual commit that occured (Commit ID)

Commit ID

municipal waste - Household organic Hazardous

Chemical Medical

The order in which the change information will display in each line as follows :

<ID><Author><Date><Time><Line number><Content>

git blame gives the points where there is change but
git log gives actual change .

Pull ~~Push~~ command is used to download the latest change from a remote repository.

* log command is used to show revision history of a repository