## What is Version control system ?

Version control, also known as source control, is the practice of tracking and managing changes to software code.

Version control systems are software tools that help software teams changes to source code over time.

## Why you want to use version control system ?

1. Managing versions
2. Maintaining a collaborative environment where multiple people can club up together and actually start contributing to the code.

## What is GIT ?

Git is distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers who are collaboratively developing source code during software development.

Goals - Speed / Data integrity / and support for distributed, non-linear workflows.

## What is GitHub ?

GitHub is a platform and cloud-based service for software development and version control, allowing developers to store and manage their code.

## Getting started

### To check version of git

The first thing we need to do, is to check if Git is properly installed :

```
git --verison
```

## Configure Git

Now let Git know who you are. This is important for version control systems, as each Git commit uses this information :

```
git config --global user.name <User_Name>
git config --global user.email <Email_Id>
```

Change the user name and E-mail address to your own. You will probably also want to use this when registering to GitHub later on.

## Creating Git Folder

let's create a new folder for project

```
mkdir <Folder_name>
cd <Folder_name>
```

mkdir - makes a new directory.
cd - changes the current working directory.
clear - clear all instructions from command line interface.

touch - creates a file

```
touch <file_name>
```

## Git init

The git init command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository.

Most other Git commands are not available outside of an initialized, so this is usually the first command you'll run in a new project.

```
git init creates a .git (subdirectory) in the current working    directory
(contains all of the necessary Git metadata for
 the new repository.)

Metadata includes subdirectories for objects, refs, and   template files.
```

### Initialize Git

Once you have navigated to the correct folder, you can initialize Git on that folder:

```
`git init` -> Powers your folder to be managed by git, and initialises a new empty
repository. It also creates a .git folder that has all the all the relevant logic
to manage versions of your project.

git init
Initialized empty Git repository in /Users/user/myproject/.git
```

- created your first git repository !.

## Git status

- We check the Git status and see if file is a part of our repo:

  `Working Area` -> `There can be bunch of files that are not currently handled by git. It menas that changes done or to be done in those files are not managed by git yet. A file which is in working area is considered to be not in the staging area. When we do 'git status' and we see bunch if 'untracked files' then these are actually called to be in the working area.`

  Files in your Git repository folder can be in one of 2 states:
- Tracked - files that Git knows about and are added to the repository
- Untracked - files that are in your working directory, but not added to the repository

  `git status`
  `On branch master`

  `No commits yet`

```
untracked files:
use "git add ..." to include in what will be committed
index.html

nothing added to commit but untracked files present (use "git add" to track
```

Git is aware of the file, but has not added it to our repository !

```
`Staging area` -> What all files are going to be part of the next versions that we
will create. This staging area is the place where git knows what changes will be
done from the last version to the next version.
```

`Repository Area` **->** `This area actually contains the details of all you previous registered version. And the files in this area, git already manages them and knows their versions history`

`git add <file_name> -> moves file from working area to staging area`

- If you want to remove file staging area ( unstaging area ) :
  `git rm --cached <file_name>`

`git rm --cached <file_name> -> moves file back from staging area to working area`

## Git Commit

Adding commits keep track of our progress and changes as we work. Git considers each `commit` change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

```
`commit -> Commit is particular version of the project. It captures a snapshot of
the Project's staged changes and creates a version out of it.`

`git commit` -> registers staging changes to a commit

    `git commit -m "First release of Hello Wordl!"`
    [master (root-commit) 221ec6e] First release of Hello World!
        3 files changed, 26 insertions(+)
        create mode 100644 README.md
        create mode 100644 bluestyle.css
        create mode 100644 index.html
```

**Note:** Short status flags are:

- ?? - Untracked files
- A - Files added to stage
- M - Modified files
- D - Deleted files

`git log -> To view the history of commits for a repository, you can use the log command:`

```
`git log`
commit 09f4acd3f8836b7f6fc.........  (HEAD -> master)
Author: w3schools-test
```

```
Data: Fri Mar 26 09: 23: 45 2023 +010

        updated index.html with a new line

commit 221ec6e10.........................
Author: ............
Date: ............

        First release of Hello World!
```

`If you want to exit out of git log prompt press 'q'`

### Git Restore

`git restore <file> -> It remove all files changes from the staging area to be committed. This can be useful, if we did some dirty piece of code and now no more want it. Instead of deleting every change line by line, we can restore it or you can say restore last clean version of the file.`

```
        `git restore <file>`
```

`git restore --staged <file_name> -> It removes file from changes from staging area to the working area.`

```
        `git restore --staged <file_name>`
```

## Difference between git rm and git restore

If you want to move the whole file back to the untracked state, then we do git rm, otherwise if we just want the changes to be moved in working area or staging area then we git restore.

-

`git commit -m <your commit> -> If we want to avoid opening a text editor like vim/nano to add commit message we can use this following command`

```
 `git commit -m <your commit message`
```

### Git Remote

`git remote -> List down all the remote connection names`

```
 `Remote connection -> It helps you to link two git repositories for uploading and
 downloading changes from each otherwise`
```

`git remote add <name of remote> <link of the remote>` : `This command helps us to add a new link to the remote repo and give name to it.`

```
 `git remote em <name of remote>` : this command delete a remote connection
```

Note: The name of the remote connection is always used to establish communication between the repositories.

### Adding multiple files in repo

`git add <file_1> <file_2> <file_3>` : This command will add multiple file changes together in the staging area.

`git add .` : This command will add all files from working repo to staging area

## Git pull

`git pull <remote name> <branch name>` : downloads latest changes from the branch of the mentioned remote in your local repo.

# Recommended practice to do

- make changes
- git add < file >
- git commit
- git pull
- git push

Merge conflicts can occur if multiple people try to make changes to the same file, and then collaborate