**GIT NOTES (Version Control System)**

**GIT** is one of the most popular and widely adopted distributed version control systems. It provides excellent support for branching, merging, and distributed development workflows. Git is known for its speed, flexibility, and robustness. It is commonly used in both small and large-scale projects.

**A version control system** is a system, a software tool that helps developers and team members to:

* track changes made on the files (source code file, assets…)
* manage multiples revisions of the same unit of information.
* collaborate easily on projects.

**BENEFITS**:

* Revert the code files back to their previous state.
* Recall and revert the entire project back to its previous state.
* Know who introduced a particular issue and when.
* Allows you to track changes made to files over time. It keeps a complete history of all modifications, including who made the changes, when they were made, and what was changed.
* Git enables collaboration among multiple developers. It provides mechanisms for sharing and synchronizing code changes between team members, facilitating concurrent work on the same codebase.
* Git makes it easy to create branches, independent lines of development. With git we can work on new features and experiment without making changes to the main code. It also allows engineers to merge the new features to the main code.
* Git repositories can be hosted on various platforms like GitHub, GitLab, and Bitbucket, allowing for easy remote collaboration. Each developer has a local copy of the entire repository, including the complete history.

**Most use VCS:**

Git, GitHub, GitLab, Bitbucket, AWS Code Commit, Azure Devops.

**GitLab** is another popular web-based hosting platform for Git repositories. It offers a complete DevOps platform **with integrated CI/CD pipelines**.

**Bitbucket** is a web-based hosting platform **that supports both Git and Mercurial repositories**. It provides features like code review, pull requests, issue tracking, **and continuous integration (CI) pipelines**. Bitbucket is often used for private repositories and offers integrations with other Atlassian products like Jira and Confluence.

**GitHub** is a widely used web-based hosting platform for Git repositories. It provides a rich set of features for collaboration, code review, issue tracking, and project management.

**AWS CodeCommit** is a fully managed, highly scalable version control service provided by Amazon Web Services (AWS). **It supports Git repositories and integrates well with other AWS services**, making it suitable for cloud-based development workflows.

**Azure Repos** is a version control service provided by Microsoft Azure. It supports both Git and Team Foundation Version Control (TFVC) repositories.

**Azure Repos** is a version control service provided by Microsoft Azure. It supports both Git and Team Foundation Version Control (TFVC) repositories.

**Difference between centralized VCS and distributed VCS**

**CVCS:** relies on a central server where developers commit changes, and the copy of the code is only stored on the remote server.

**Users need to be online** to interact with the remote server. Changes cannot not be done offline**. Users also need to be in network** to push changes, limiting where and when developers can commit.

**There is a single point of failure** because if the only copy of a project goes down, developers are unable to access the code or retrieve previous versions.

**DVCS:** distributed version control **doesn’t have a single point of failure** because developers clone repositories on their DVCS workstations and create multiples backup copies. If the source code is corrupted, teams can use any developer’s clone as a backup.

**Users can commit offline** and doesn’t necessarily need to be in network because there are local copies.

**REPOSITORY definition**

Git repositories store and manage the history of changes to a project's source code and other files.

a repository can be either **local or remote**:

* **Local Repository**: A local repository is stored on your local machine. It is created when you initialize a Git repository in a directory using the git init command or by cloning an existing repository.
* **Remote repositories** store the complete history and branches of a project, just like local repositories. It is where developers can push their local commits and pull the latest changes to/from the remote repository to share their changes with others.

It can also be **private or public.**

**A private repository:** it is a repository that belongs to a specific group or organization and whoever accesses it has the appropriate permissions. It provides privacy and confidentiality and is used for professional software development.

**A public repository** is accessible and available to anyone allowing individual to view, clone, and contribute to the code base.

**DIFFERENT COMMANDS**

**Git init** creates a new Git repository.

**.git directory:** it is hidden to prevent accidental deletion or modification of the folder. **It contains details of every single change made to the code base.** which makes it possible to undo the changes and rollback to the desired version of the code.

**git config --global** command allows you to configure settings that will apply to all repositories on your machine and are specific to your user account.

**git add** command is used to add changes or files to the staging area in Git before committing them.

**git status** command is used to display the current status of the Git repository. It helps to understand the state of the repository and identify any pending changes or untracked files.

**git commit** command records changes to the repository. It creates a new commit with a unique identifier and a commit message describing the changes.

**git commit --amend --no-edit** is used to make the additional changes to your **commit files.**

**git push** uploads local commits to a remote repository, typically used to share changes with others.

**git pull r**etrieves and merges changes from a remote repository into the current branch.

**git branch** lists, creates, or deletes branches in the repository.

**git checkout** switches between branches or restores files from a specific commit.

**git merge** integrates changes from one branch into another.

**git log** displays a log of commits in reverse chronological order.

**git remote** manages connections to remote repositories.

**git restore** command is used with the wildcard (\*) to restore multiple files or even all files in your working directory to a previous state.

**git stash** command is used to save changes that are not ready to be committed yet. It's particularly useful when you need to switch to a different branch or pull in changes from a remote repository without committing your local changes.

**AUTHENTICATION**

In git is not advice to use username and password to authenticate because is not safe. The best way is to use **https** (use a token to authenticate) or **ssh.**

**NB:** after getting the job when we will receive the company computer, to authenticate on their GitHub account, we need to: **2 things to authenticate, SSH KEY and the Git config.**

1. **SSH**

* **Run ssh-keygen:** it will generate a key and show the location where that key is store for exp **c:/users/ludivine/.ssh/**
* **Cd:** we need to cd into the path **cd c:** **/users/ludivine/.ssh/**
* **Ls:** it will display the private and public key
* **Cat the public key:** cat id\_rsa.pub and copy the content.
* **Go into GitHub 🡺 settings🡺 ssh and gpg keys 🡺 click on new key 🡺 paste the key 🡺 give it a name and save.**

1. **Git CONFIG**

* Git config - -global user.name “username”
* Git config - -global user.email email address (your work email address)

**NB: always create your repository from GitHub and choose the option add a ReadMe file that way, git will initialize my repository.**

**CONTINUOUS INTEGRATION, DELIVERY AND DEPLOYMENT**

**Continuous Integration (CI**) is the practice of frequently merging code changes from multiple developers into a shared repository. where automated build and test processes are triggered. The main goal of CI is to identify and resolve integration issues early in the development process.

**Continuous Deployment (CD**) is an extension of CI that automatically deploys code changes to a staging or production environment after passing the build and test stages. CD is an automate process ensuring that software updates are delivered to end-users quickly and frequently.

**Continuous Delivery (CD**) is like continuous deployment, but with an additional **step of manual approval before deploying to production.** In continuous delivery, once the code changes have passed the automated build and test stages, they are deployed to a staging environment. A manual approval process is then triggered before the changes are released to the production environment.