GIT

Git is an example of a distributed version control system (DVCS) commonly used for open source and commercial software development. DVCSs allow full access to every file, branch, and iteration of a project, and allows every user access to a full and self-contained history of all changes. Unlike once popular centralized version control systems, DVCSs like Git don’t need a constant connection to a central repository. Developers can work anywhere and collaborate asynchronously from any time zone.

How to install GIT?

Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the package management tool that comes with your distribution. If you’re on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use dnf:

-$ sudo dnf install git-all

If you’re on a Debian-based distribution, such as Ubuntu, try apt:

-$ sudo apt install git-all

Process for initializing GIT

To use Git on the command line, you'll need to download, install, and configure Git on your computer.

If you want to work with Git locally, but don't want to use the command line, you can instead download and install the GitHub Desktop client. For more information, see "Getting Started with GitHub Desktop."

Setting up Git

1. Download and install the latest version of Git.

2. Set your username in Git.

3. Set your commit email address in Git.

Next steps: Authenticating with GitHub from Git

When you connect to a GitHub repository from Git, you'll need to authenticate with GitHub using either HTTPS or SSH.

Connecting over HTTPS (recommended)

If you clone with HTTPS, you can cache your GitHub password in Git using a credential helper.

Connecting over SSH

If you clone with SSH, you must generate SSH keys on each computer you use to push or pull from GitHub.

Basic GIT Commands

* git init :initializes a brand new Git repository and begins tracking an existing directory. It adds a hidden subfolder within the existing directory that houses the internal data structure required for version control.
* git clone:creates a local copy of a project that already exists remotely. The clone includes all the project’s files, history, and branches.
* git add:stages a change. Git tracks changes to a developer’s codebase, but it’s necessary to stage and take a snapshot of the changes to include them in the project’s history. This command performs staging, the first part of that two-step process.
* Git commit:saves the snapshot to the project history and completes the change-tracking process. In short, a commit functions like taking a photo. Anything that’s been staged with git add will become a part of the snapshot with git commit.
* Git status: shows the status of changes as untracked, modified, or staged.
* Git branch: shows the branches being worked on locally.
* Git merge: merges lines of development together. This command is typically used to combine changes made on two distinct branches. For example, a developer would merge when they want to combine changes from a feature branch into the master branch for deployment.
* Git pull:updates the local line of development with updates from its remote counterpart. Developers use this command if a teammate has made commits to a branch on a remote, and they would like to reflect those changes in their local environment.
* Git push: updates the remote repository with any commits made locally to a branch.

Commands of GIT

1. CONFIGURE TOOLING

Configure user information for all local repositories

• git config --global user.name "[name]"

Sets the name you want atached to your commit transactions

• git config --global user.email "[email address]"

Sets the email you want atached to your commit transactions

• git config --global color.ui auto

Enables helpful colorization of command line output

2. CREATE REPOSOTORIES

Start a new repository or obtain one from an existing URL

• git init [project-name]

Creates a new local repository with the specified name

• git clone [url]

Downloads a project and its entire version history

3. MAKE CHANGES

• git status

• git diff

• git add [file]

• git diff –staged

• git reset [file]

• git commit -m "[descriptive message]"

4. GROUP CHANGES

• git branch

• git branch [branch-name]

• git checkout [branch-name]

• git merge [branch]

• git branch -d [branch-name]

5. REFACTOR FILENAMES

6. SUPPRESS TRACKING

7. SAVE FRAGMENTS

8. REVIEW HISTORY

9. REDO COMMITS

10. SYNCHRONIZE CHANGES

• git fetch [bookmark]

• git merge [bookmark]/[branch]

• git push [alias] [branch]

• git pull

Creating new GIT repo

1) Initialize the repository

Command:

Git init

2) Add all the files to the repository

Command:

Git add . (To add all the files)

Git add filename(To add the one file)

3) Save the changes

Git commit

Some Advantages:

1) Git won’t duplicate the data, Instead, it only maintains changes in a logical way.

2) Git is simple to use if we understand the concept of it.

3) Branching is the one best thing, Using this we can divide the work later we can merge the work into master. It suits well for the collaborative environment