Version control is one of the significant challenges faced by developers since the genesis of coding. Pushing source code into a central server is what developers usually do as part of version control. Whether it is Java, .Net, Python, or Database code, when an application under development, there is minimum communication among developers. Hence, there are significant conflicts over code version controls, tracking, and retrieval, which could potentially lead to delays in project or product release. Version Control Software (VCS) are tools that help in software configuration management and help to keep track of the source code and builds application components or modules.

What is Git?

‘’Git’’ is a go-to-version control tool that allows developers to access all of the code and efficiently manage their source code and track file changes be it small, medium, or massive application development. It remains the most widely used open-source distributed version control system (https://en.wikipedia.org/wiki/Distributed_version_control) (DVCS) till date and has been in use for over a decade after its initial release. Unlike other version control systems that store a project’s full version history in one place, Git gives each developer their repository locally containing the entire history of changes and the entire application.

- Git is used to tracking changes in the source code hence tracks history
- Git is a distributed version control tool for source code management. It is free and open-source.
- Git creates backup automatically, as the developer has a version of the code on the local repository.
- Git allows multiple developers to work together; hence is scalable and supports collaboration.
- Git supports non-linear development because of its thousands of parallel branches.

This Git cheat sheet can serve as a ready reckoner to essential git commands (https://hackr.io/blog/basic-git-commands-with-examples) that you may need to use in your coding career.

Installing Git

Before you start using Git, you have to make it available on your computer. Even Though if it is installed, it is probably a good idea to update to the latest version. You can either install it as a package or via another installer or download the source code and compile it yourself.

Installing on Linux

If you want to install the essential 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 are on operating systems such as Fedora, you can use dnf.

Git Cheat Sheet: Quick Guide for Reference

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 are on operating systems such as Fedora you can use dnf.	$ sudo dnf install git-all
	If you are on a Debian-based distribution, such as Ubuntu use the following command	$ sudo apt install git-all
Installing on macOS	There are several ways to install Git on a Mac. The easiest way is to install it from the Github website	https://mac.github.com
Installing on Windows	There are also a few ways to install Git on Windows. The most official build is available for download on the Git website.	https://git-scm.com/download/win (https://git-scm.com/download/win)

Configuring Git

Configure user information for all local repositories on your computer

$ git config --global user.name "[name]"	Sets the name you want to be attached to your commit transactions.
$ git config --global user.email "[email address]"	Sets the email you want to beattached to your commit transactions.
$ git config --global color.ui auto	Enables helpful colorization of the command line output
$ git config –global alias	Creates a Git command shortcut
$ git config –system core.editor	Sets the preferred text editor
$ git config –global –edit	Open and edit the global configuration file in the text editor

Setting Up Git Repositories

$ git init [project-name]	Creates an empty repository in the project folder with the specified name
$ git clone (repo URL)	Downloads a project from a remote service such as Github and its entire version history
$ git clone (repo URL) (folder)	Clones a repository to a specific folder
$ git remote -v	Displays a list of remote repositories with URLs
$ git remote rm (remote repo name)	Removes a remote repository
$ git fetch	Fetching from a repository grabs all the new remote-tracking branches and tags without merging those changes into your own branches.
$ git pull	Retrieve the most recent changes from origin and merge

Managing File Changes

$ git add (file name)	Adds file changes to staging. Snapshots the file in preparation for versioning.
$ git add	Adds all directory changes to staging
$ git add -A	Adds new and modified files to staging
$ git rm (file_name)	Removes a file and stops tracking it. Deletes the file from the working directory and stages the deletion
$ git rm –cached (file_name)	Removes the file from version control but preserves the file locally
$ git checkout <deleted file name>	Recovers a deleted file and prepares it for commit
$ git status	Displays the status of modified files. Lists all new or modified files to be committed

$ git diff	Displays all unstaged changes in the index and the current directory. Shows file differences that are not yet staged
$ git diff --staged	Shows file differences between staging and the last file version.
$ git reset [file]	Unstages the file, but preserve its contents
$ git commit -m "[descriptive message]"	Records file snapshots permanently in version history
$ git mv [file-original] [file-renamed]	Changes the file name and prepares it for commit

REDO COMMITS

Erase mistakes: You would typically want to UNDO/REDO when you commit some changes to Git and realize that the changes need to be removed/reverted.

$ git reset [commit]	Undo all commits after [commit], preserving changes locally
$ git reset --hard [commit]	Discards all history and changes back to the specified commit

GROUP CHANGES: Commands for Git branching

You can decide how to group the changes to create meaningful commits.

$ git branch	Lists all local branches in the current repository
$ git branch [branch-name]	Creates a new branch
$ git checkout [branch-name]	Switches to the specified branch and updates the working directory
$ git merge [branch]	Combines the specified branch’s history into your current branch
$ git branch -d [branch-name]	Deletes the specified branch

$ git fetch remote <branchname>	Fetches a branch from the repository
$ git push –all	Pushes all local branches to a designated remote repository

SAVE FRAGMENTS

The Git stash command removes changes from your index and “stashes” them away for later. It is useful if you wish to pause what you are doing and work on something else for a while. You cannot stash more than one set of changes at a time.

$ git stash	Temporarily stores all modified tracked files
$ git stash pop	Restores the most recently stashed files
$ git stash list	Lists all stashed changesets
$ git stash drop	Discards the most recently stashed changeset

Review History

Browse and view the version history of your project files.

$ git log	Lists version history for the current branch
$ git log --follow [file]	Lists version history for a file, including renames
$ git diff [first-branch]...[second-branch]	Shows content differences and conflicts between two branches
$ git show [commit]	Outputs metadata and content changes of the specified commit

Git Glossary

Branch	Branches represent specific versions of a repository that “branch out” from your main project. Branches allow you to keep track of experimental changes you make to repositories and revert to older versions