A version control system is a piece of software that helps the developers on a software team work together and also archives a complete history of their work.

There are three basic goals of a version control system (VCS):

1. We want people to be able to work simultaneously, not serially.
2. When people are working at the same time, we want their changes to not conflict with each other.
3. We want to archive every version of everything that has ever existed — ever.

A filesystem is two-dimensional: Its space is defined by directories and files. In contrast, a repository is threedimensional: It exists in a continuum defined by directories, files, and time. A version control repository contains every version of your source code that has ever existed.

A consequence of this idea is that nothing is ever really destroyed. Every time you make some kind of change to your repository, even if that change is to delete something, the repository gets larger because the history is longer.

Each change adds to the history of the repository. We never subtract anything from that history.

The essential difference between a Centralized Version Control System (CVCS) and a DVCS is the notion of a *repository instance*.

In a CVCS, the repository exists in one place on a central server. Every piece of software that is used to access the repository includes a network client.

Most operations interact with a local repository instance, not a network server. The only time networking code gets involved is when the repository instances are being synchronized. Every developer has his own private repository instance.

In practice, virtually all DVCS teams have a central server. With a CVCS, a central server happens because it is inherent in the centralized model. With a DVCS, a central server happens because of the team’s decision to have one.

**Git Cheat Sheet**

git diff Show unstaged changes between your index and working directory.

**git config --global –edit** Open the global configuration file in a text editor for manual editing.

git diff HEAD Show difference between working directory and last commit.

git diff –cached Show difference between staged changes and last commit

**git log --grep=”<pattern>”** Search for commits with a commit message that matches <pattern>.

|  |  |
| --- | --- |
| Create | |
| $ git init | Fom existing data |
| $ git clone | From existing repository |

|  |  |  |
| --- | --- | --- |
| Show | | |
| $ git status | | Files changed in working directory |
| diff | $ git diff | Changes made to tracked files |
| $ git diff --staged | Shows file differences staging and the last revision |
| $ git diff C1 C2 | What changed between C1 and C2 |
| $ git log | | History of changes |
| $ git log –p File | | History of changes for file with diffs |
| $ git blame File | | Who changed what and when in a file |
| $ git show C1 | | A commit identified by ID |
| $ git show C1:File | | A specific file from a specific commit |
| $ git branch | | All local branches |
|  | |  |
|  | |  |

$ 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 between two branches

$ git show [commit]

Outputs metadata and content changes of the specified commit

Revert



$ git reset [file]

Unstages the file but preserve its contents

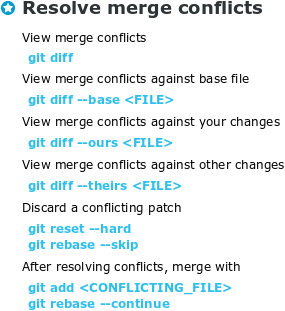
$ git reset [commit]

Undoes all commits after [commit], preserving changes locally

$ git reset --hard [commit]

Discards all history and changes back to the specified commit

Resolve merge conflicts



$ git rm [file]

Deletes the file from the working directory and stages the deletion

$ git rm --cached [file]

Removes the file from version control but preserves the file locally

$ git mv [file-original] [file-renamed]

Changes the file name and prepares it for commit

