GIT DAY-1

Thursday, 23 October 2025 6:41 PM

What is Git?

Git is a **Version Control System (VCS)** used to track changes in files and coordinate work among multiple people on a project. It helps developers manage code versions efficiently, collaborate without overwriting each other's work, and maintain a complete history of all changes made to the project.

What Problems Does Git Solve?

Before version control systems like Git, teams faced major issues such as:

- **Overwriting code:** Multiple developers working on the same file could accidentally overwrite each other's changes.
- **Lost progress:** Without version history, it was easy to lose work or be unable to recover an older version of a file.
- **Difficult collaboration:** Sharing updates and merging changes was manual and error-prone.
- **No traceability:** It was hard to see who changed what, when, and why.

Git solves all of these by providing:

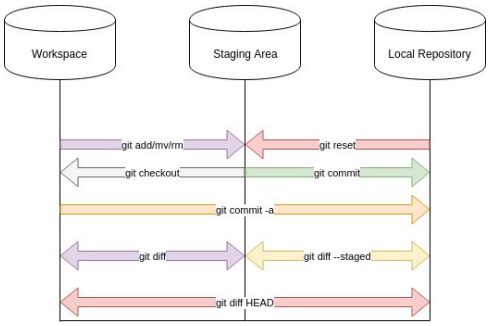
- **Branching and merging:** Developers can work on separate branches (for new features or bug fixes) without disturbing the main codebase. These branches can later be merged seamlessly.
- **Change history:** Every commit is recorded, making it easy to roll back to previous versions or review the project's evolution.
- **Collaboration:** Git enables multiple people to work on the same project efficiently, even across different locations.
- **Backup and recovery:** Since every copy of a Git repository contains the entire history, it acts as a distributed backup system.

Why DevOps Professionals Need Git

Git is an essential tool in the **DevOps lifecycle** because it serves as the **single source of truth** for all code, configuration, and automation scripts. DevOps engineers use Git for:

- **Continuous Integration and Continuous Deployment (CI/CD):** CI/CD pipelines automatically pull code from Git repositories to build, test, and deploy applications.
- **Infrastructure as Code (IaC):** DevOps teams use Git to version-control infrastructure definitions (e.g., Terraform, Ansible, Kubernetes configs).
- **Collaboration and automation:** Git integrates with tools like Jenkins, GitHub Actions, GitLab CI, and Azure DevOps for automated workflows.
- **Auditing and compliance:** Git provides full traceability of changes, which is crucial for security and compliance audits.

Each commit has its own unique ID, it is a hash of its containers and its metadata. It must have a perfix of at least 4 chars (unique in repo).



Git pull

How Git Works

In Git, files can exist in three main states:

1. **Modified** – The file has been changed but not yet saved (committed) to the Git database.
2. **Staged** – The file has been marked to be included in the next commit. This means Git knows that the changes are ready to be recorded.
3. **Committed** – The file and its changes are safely stored in Git's database. This becomes part of the project's permanent version history.

Git allows developers to move files between these states — modifying, staging, and committing — to keep track of their progress and organize their work before saving a final version.

.gitignore = file in the main folder which contains a list of files that are ignored by git (so they're not affected by git adds and git statuses). It can contain patterns (*.pyc) and paths to files.

Install:
sudo apt-get install git
Set up:
    git config --global user.name "User Name"
    git config --global user.email "email@email.com"

Create a New Project:
    mkdir project
    cd project
    git init (initialise empty git repo in my folder (based on path) aka .git folder)
    ls -la (check my folder)
Check "world status":
git status

Add files
    git add . = add all on current branch
    git add -p <param=file> = add part of file to staging area, ask for each change (if no param => all files) so we have more control and cleaner commits. After any git add, we need a git commit, either a file or a pattern (e.g. *.txt)

Delete a file
    git rm <filename> = deletes a file, updates git and then commit!
    git rm --cached <filename>" = delete a previously tracked file
Move a file
git mv <old path> <new path> should be followed by:
git rm <old path>
git add <new path>

What does git commit do?
    commit a file = create a snapshot of the current world state (files, folders & their contents)

What is a branch?
It is a version of our code. Branches have a name and are pointing to a commit (there's a different history+past commits depending on our branch, but some commits may be common).
One branche per feature (the smaller the better) so changes happen to the branch, not the master workflow until the final merge. Afterwards, we merge and delete the branch.
Commands
    git branch <name> = creates the branch, it's an exact duplicate of our current/previous branch (they point to the same commit)
    git branch = returns my current branch
    git checkout <name> = changes current (HEAD), <name> points to HEAD now
    git branch -d <name> = deletes this branch (NOT the commits also)
    git checkout -b <name> = creates a new branch and makes this new branch as our current working one = git branch <name> + git checkout <name>
    git merge <branch> = merges 's history with my current branch + try to merge changes in files from both the branches => 2 parents in new commit. (Afterwards we find the most recent parent of those two parents => commits of the new branch = commits of parent1 + commits of parent2 => updates master, master in new commit - see schema (1))

contains an explanatory message
automatically stores metadata (creator, date etc)
has a unique (hex) id number
e.g.: git commit -m "Added README file

Combinations
git commit -a = git add + git commit (not desirable due to lack of control)
git pull = git fetch + git merge (very useful)
Check difference
git diff= displays what will be added if i git add, so what changed in the folder and hasn't been updated yet
git diff <filename> = displays the alterations of a file (the modified and the commited versions of it)
git diff --staged = displays what has already been added and thus what changed will be recorded
git diff HEAD = displays changes since last commit

Display history
git-log = displays the history, the chronologival order of commits (based on their IDs), who did them, what was their description

git show <id> = displays what the commit did = git log + git diff
Make an alias
git config --global alias.<aliasname> "command(s)"
e.g.:
git config --global alias.lg "log --color --graph --pretty=format: '%(red%h%(green(%cr)%((bold blue)<%
an>%(reset' --abbrev -commit"
git config --list - displays our aliases