git commands

- **git --version** = which version of git is installed .
- **git config** = set user info and preferences.
- **git init** = Initialize the git repository in the project directory or folder
- **git clone** = copy the existing repository to the system.
- **git status** = to show the status of working directory
- **git add <file.txt>** = adding to staging area or tracking area.
- **git add .** = adding all the files under the directory to staging are.
- **git commit -m "message"** = To record changes in the repository.
- **git commit -am "message"** = Add to staging area and commit .
- **git branch** = List, Create, delete the branches.
- **git checkout** = Switch branches
- **git merge** = Merge another branch into current branch.
- **git remote -v** = list the remote connections.
- **git remote add origin <repo-url>** = Add a remote repository.
- **git push** = Upload local changes to remote

git push -u origin feature

- **git pull** = Fetches and merges from the remote repo.
- **git fetch** = Only for download the changes but doesn't merge.

git fetch origin

- **git log** = to view commit history.

git log --oneline

git log --graph --all = tree view.

- **git diff** = shows the difference between working directory and staging / index.
- **git show <commit-id>** = shows details of a specific commit.
- **git reset** = To unstage the file from the staging area.

git reset --soft HEAD~1 = Moves HEAD only

git reset --mixed HEAD~1 = Moves HEAD + unstaged files

git reset --hard HEAD~1 = Resets Everything.

- **git checkout -- <file>** = Discards changes in working dir.
- **git clean -n** = Remove the untracted files.
- **git clean -f** = Force delete
- **git tag** = To mark specific point in the history.

git tag v1.0

git tag -a v1.0 -m "Release version 1.0"

git push origin v1.0

- **git rebase** = Reapplies commits on top of another base.
- **git stash** = Saves uncommitted changes temporarily.
- **git cherry-pick <commit-id>** = Apply a single commit from another branch.
- **git revert <commit-id>** = Creates a new commit that undoes the given commit.
- **git remote add upstream <repo-url>**
- **git fetch upstream.**

****

What is Version Control? Why Git?

What is Version Control?

- **Version Control System (VCS):** A tool that tracks changes in source code or any files over time.
- **Purpose:**
  - Maintain a history of file changes.
  - Collaborate with multiple developers without overwriting each other's work.
  - Revert back to previous versions when needed.

Why Git?

- **Git** is a **Distributed Version Control System (DVCS)** created by Linus Torvalds.
- Key Reasons to Use Git:
  - Distributed architecture (every developer has a full copy of the repository).
  - Fast, lightweight, and scalable.
  - Robust branching and merging features.
  - Wide industry adoption (GitHub, GitLab, Bitbucket).

Real-World Example:

- Large enterprise projects use Git to manage hundreds of microservices simultaneously.
- DevOps teams track infrastructure code changes using Git repositories (Infrastructure as Code).

Where to Implement:

- Software development projects.
- DevOps pipelines (CI/CD workflows).
- Infrastructure as Code (Terraform, Ansible).
- Configuration file versioning.

****

Installing Git (Linux/Windows)

Linux:

- Install using package manager:

```
sudo apt update
sudo apt install git
git --version
```

Windows:

- Download Git installer from https://git-scm.com/
- Follow setup instructions.
- Use Git Bash terminal after installation.

Real-World Example:

Developers in a cross-platform environment can collaborate seamlessly by installing Git on both Linux servers and Windows development machines.

Key Concepts

- **Repository (repo)** – Your project's code + history.
- **Local repo** – Exists on your machine.
- **Remote repo** – Stored on a server (GitHub, GitLab, Bitbucket, etc.).
- **Commit** – A snapshot of your changes.
- **Branch** – An isolated line of development.
- **HEAD** – Pointer to the current commit you're working on.
- **Clone** – Copy of a remote repo to local.
- **Staging area (index)** – Where you prepare changes before committing.

Git Installation & Setup

 # Install Git

sudo apt install git -y # Ubuntu/Debian

brew install git # macOS

choco install git # Windows (Chocolatey)

 # Configure identity (important for CI/CD commit tracking)

git config --global user.name "NAME"

git config --global user.email "EMAIL@example.com"

 # Check settings

git config –list

Creating & Initializing Repos

 # Initialize in an existing folder

git init

 # Clone from remote

git clone https://github.com/org/repo.git

 # Add a remote

git remote add origin https://github.com/org/repo.git

git remote -v

Staging, Committing, and Pushing

 # Stage files

git add file.txt

git add . # Stage all changes

 # Commit changes

git commit -m "Add new feature"

 # Push to remote

git push origin main

 # Pull latest changes

git pull origin main

Branching & Merging (DevOps Best Practices)

Branches are essential in **feature-based development** and **CI/CD workflows**.

 # Create a branch

git branch dev

 # Switch to branch

git checkout dev

git switch dev # Newer command

 # Create and switch in one go

git checkout -b feature/login

 # Merge into main

git checkout main

git merge feature/login

 # Delete branch

git branch -d feature/login

git push origin --delete feature/login

Working with Remotes

 # List remotes

git remote -v

 # Change remote URL

git remote set-url origin git@github.com:org/repo.git

 # Fetch changes without merging

git fetch origin

 # Sync fork

git pull upstream main

Undoing Changes

 # Unstage a file

git reset file.txt

 # Discard local changes

git checkout -- file.txt

git restore file.txt # Newer

 # Reset to previous commit

git reset --hard <commit_id>

 # Revert a commit (safe for shared branches)

git revert <commit_id>

Viewing History

git log

```
git log --oneline --graph --decorate --all
git show <commit_id>
git diff
git diff --staged
```

Git in CI/CD Pipelines (DevOps Usage)

In Jenkins, GitLab CI, GitHub Actions, etc., Git is the first step of the pipeline:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/org/repo.git'
            }
        }
    }
}
```

Git Tags (for Deployments)

```
# Create tag
git tag v1.0.0

# Annotated tag
git tag -a v1.0.0 -m "Release version 1.0.0"
```

 # Push tags

git push origin --tags

Use tags for **production releases** so CI/CD can deploy specific versions.
