Git

Git is a source code Version Control system (VCS) that helps developers track and manage changes to code over time. It's especially useful when working in teams or managing complex projects. It is useful for — collaborating in teams, tracking history, branching, merging, and backups.

Before Git, centralized systems like SVN and BitKeeper were used. Git was adopted because it's faster, distributed, open-source, and much better at branching and merging, making it ideal for modern software development.

Install Git

- Windows: Download From https://git-scm.com/
- Linux: `yum install git -y` / `apt install git -y`
  - Verify Installation

    `git --version`
  - Configure Git

    `git config --global user.name <name>`

    `git config --global user.email <email>`

    `git config --list` - Shows all Git configuration settings that apply in the current context. All settings (system, global, local).

    `git config --global --list` - Shows only the global (user-level) Git settings.

    Optional -

    - `git config --global core.editor "code"` Set VS code as editor
    - `git config --global init.defaultBranch main` This command sets the default branch name to `main` (instead of `master`) for all new Git repositories you create with `git init`

- Create Local Repository

```
mkdir my-project
cd my-project
git init
```

`git init` This creates a hidden folder: `.git` → Contains all metadata and version history (this is your local Git repository).

Git Areas

1. Working Directory
2. Staging Area
3. Commit History (Repository)

How Git Tracks Code: -

Files in Git move through three main areas:

1. **Working Directory**

Your actual project folder where you create and modify files. Contains untracked (new) or modified files that Git doesn't automatically track until instructed.

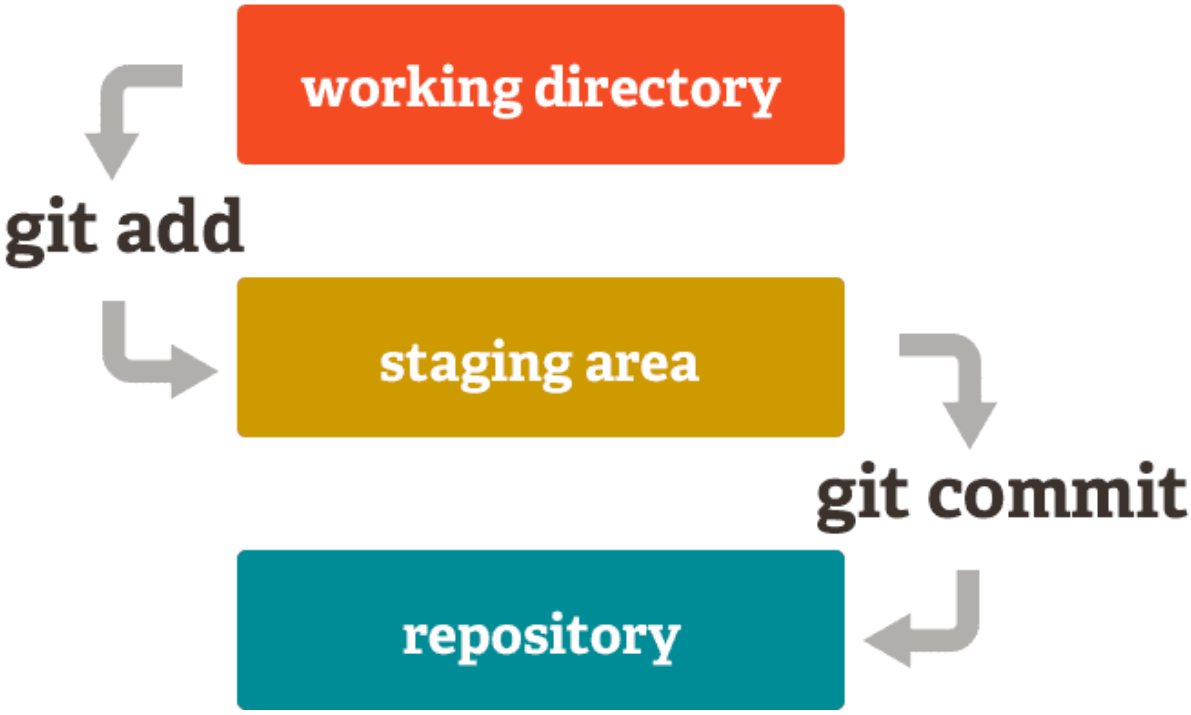
2. **Staging Area (Index)**

A preview zone where you select changes for the next commit using:
git add <filename>
Allows selective committing of specific files or changes.

3. **Local Repository**

Where committed changes are stored:
git commit -m "Your message"
Creates a project snapshot stored in the .git folder, existing only on your local machine.

Flow: Working Directory → (git add) → Staging Area → (git commit) → Local Repository
To push to GitHub: git push origin <branch>



Git Config Scope Summary

Command	Scope	Applies To
`git config --system user.name "name"`	System	All users, all repos
`git config --global user.name "name"`	Global	Current user, all repos
`git config user.name "name"` *(no flag)*	Local	Current repository only

local > global > system - Config Priority

Git Workflow

Step	Command	What It Does
Add files	`git add filename` or `git add .`	Stages files to be committed
Commit	`git commit -m "Message"`	Saves snapshot of staged files
Check status	`git status`	Shows file changes (tracked/untracked)
History	`git log`	Shows commit history

Git Commands

```
# Check Git Version:
git --version

# Add Files:
git add <file-name>        # Add a specific file
git add .                  # Add all changes (new, modified)

# Commit Changes:
git commit -m "Your Commit Message"

# Git Configuration:
git config                         # View general config help
git config --system                # Use system-wide config (less common)
git config --global                # Use user-level config
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
git config --global init.defaultBranch main   # Set 'main' as default for new rep
os

# View Config Settings:
git config --list                  # List all Git settings (merged view)
git config --global --list         # Only global config
git config --system --list         # Only system config

# Pull Repository:
git pull <git-repo-url>            # Rarely used this way; URL pulls from specific
repo
git pull origin <branch>           # Common way to pull latest from a remote br
anch

# Connecting to Remote Repository:
git remote add origin <git-repo-url>   # Connect local repo to remote
```

```
# Verify Remote:
git remote                    # List remote names (like 'origin')
git remote -v                  # Show remote names with URLs

# Push Changes to Repository:
git push origin main           # Use 'main' if that's your branch
# (Use 'master' only if the branch is still named master)

# Rename Branch:
git branch -m main             # Rename current branch to 'main'
git branch -m master main      # Rename 'master' to 'main' (if you're not o
n it)

# Clone a Remote Repository:
git clone <git-repo-url>

# Check Status:
git status                     # See what's changed and staged
git status -s                  # in short

# View Commit History:
git log                        # List past commits
git log --oneline
git log -p
git log --since="1 week ago"

# View Changes
git diff              # Show unstaged changes (working directory vs staged)
git diff --staged     # Show staged changes (staged vs last commit)
git diff HEAD         # Show all changes (working directory vs last commit)

# Difftool
git difftool          # View unstaged changes using difftool
git difftool --staged # View staged changes using difftool
git difftool HEAD     # View working area and commit area using difftool
```

```
# Undoing Changes
git reset
git reset --soft HEAD~1  # Unstages files (moves them from staged to working area). Doesn't change file content.
git reset --mixed HEAD~1 # (default.)  (default)  Undo last commit but keep all changes staged.
git reset --hard HEAD~1  # Completely undo the last commit and all changes. Warning: changes are lost.
# HEAD~1 means “one commit before the current one.”

# Working with reflog
git reflog               # Shows a log of all actions that moved HEAD (commits, resets, checkouts, etc).
git reset --hard HEAD@{3}# Go back to the state the repo was in 3 actions ago.
git reset --hard <commit-id>    # Reset the repo to a specific commit ID. Be careful — it erases uncommitted changes.

# git tag lightweight & annotated tag
git tag v1.0             # Creates a lightweight tag
git tag v2.0 <commit-id> # Tags a specific commit.
git tag -a v1.0 -m "Version 1.0"  # Creates an annotated tag
git tag -d v2.0          # Deletes a local tag named v2.0

# Comparing branches & pulling changes
git diff main..origin/main      # Shows the difference between your local main and the remote origin/main.
git pull <link-or-remote>       # Fetches and merges changes from the remote repository.
git fetch origin                # Copies changes from remote to local repo without merging.
git merge                       # Usually followed by 'git merge' to apply the changes.

# Cloning and merging
git clone <repo-url>            # Copies the entire repository to local machine and initializes it.
```

```
git merge <branch-name>          # Merges the specified branch into the curre
nt branch.

# Working with branches
git branch <branch-name>         # Creates a new branch with the specified na
me.
git branch                       # Lists all local branches in the repository.
git checkout <branch-name>       # Switches to the specified branch.
git checkout -b <branch-name>    # Creates a new branch and switches to it.
                                 # If the branch already exists, it just switches to it.
git switch <branch-name>         # An alternative to 'git checkout' for switchin
g branches (newer and more user-friendly).

# Working with stash
git stash -u   # Include untracked files
git stash -a   # Include untracked and ignored files

git stash                        # Temporarily saves your changes (that are not commi
tted) and cleans the working directory.
git stash list                   # Shows a list of all stashed changes.
git stash show                   # Displays the changes in the most recent stash.
git stash pop                    # Applies the most recent stash and removes it from
the stash list.

# Deleting branches
git branch -D <branch-name>      # Forcefully deletes the specified local bran
ch.

# Renaming remotes
git remote rename origin origin2  # Renames the remote named 'origin' to 'ori
gin2'.
```

.gitignore

A `.gitignore` file is used in Git to tell the version control system which files or directories to **ignore**—that is, not track or commit to the repository.

```
# .gitignore file example
secret.txt
config.json
*.log                  # Ignores all log files
!output.log               # dont ignore !
/directory/
.gitignore
```

Files already tracked by Git won't be ignored even if you add them to `.gitignore`. You need to untrack them first:

```
git rm --cached <filename>
```

Fork

A fork is a personal copy of someone else's repository, created under own account.

It allows to freely make changes to the code without affecting the original project. Forking is commonly used when we want to:

- Contribute to an open-source project without having direct access.
- Experiment with new features or fixes safely.
- Customize a project for own use.

Forking is especially useful in open-source collaboration, where contributors work on their own versions of a project and suggest improvements to the maintainers of the original repository.