**Git & GitHub Complete Guide for AI/ML Developers**

**1. What is Git and GitHub?**

**Git**: A version control system (VCS). It keeps track of your code history (who changed what, when, and why).

Think of it like a time machine for your code.

**GitHub**: A cloud platform where Git repositories are hosted, shared, and collaborated on.

Think of it like Google Drive for Git repos, but with teamwork features (issues, PRs, CI/CD).

Why important for ML developers?

You’ll work on multiple experiments → Git helps you track versions.

Team collaboration → GitHub helps sync work.

Open-source ML → Most ML libraries (TensorFlow, PyTorch) are on GitHub.

**2. Setting Up Git**

1. Install Git (Linux/Mac/WSL):

**sudo apt update**

**sudo apt install git**

**git --version**

(On Windows → install from [git-scm.com](https://git-scm.com/)).

2. Configure identity (important! This info shows up in commits):

**git config --global user.name "Your Name"**

**git config --global user.email** [**your\_email@example.com**](mailto:your_email@example.com)

Example:

git config --global user.name "Ashish Sharma"

git config --global user.email "ashish.dev@gmail.com"

3. Set default branch name to `main` (modern practice):

**git config --global init.defaultBranch main**

4. Check configuration :

**git config --list**

**3. Creating a Repository (Your First Project)**

There are 2 ways:

1. Local repo → push to GitHub (start on your PC, then upload).

2. Clone repo from GitHub (start on GitHub, then pull to PC).

Step 1: Create project folder

**mkdir ml\_project**

**cd ml\_project**

Step 2: Initialize Git

**git init**

Creates a hidden `.git` folder → this is the database tracking changes. Your folder is now a Git repository .

Step 3: Create a file

**echo "print('Hello ML')" > app.py**

Step 4: Check status

**git status**

Shows untracked files. (Untracked = Git doesn’t know them yet).

Step 5: Stage files (tell Git what to track)

**git add app.py**

To add all files:

**git add .**

Step 6: Commit (save snapshot)

**git commit -m "Initial commit: added app.py"**

To remove a file from git tracking but keep it on disk:

**git rm --cached secret.key**

**echo "secret.key" >> .gitignore**

**git add .gitignore**

**git commit -m "chore: stop tracking secret.key"**

**4. Git’s Three Stages**

1. Working Directory → your actual files.

2. Staging Area (Index) → files added with `git add`.

3. Repository (History) → files saved with `git commit`.

Command to visualize:

**git status**

**5. Connecting to GitHub**

1. Create repo on GitHub (no README for now).

Example: `ml\_project`

2. Link local repo to GitHub remote :

**git remote add origin** [**https://github.com/your-username/ml\_project.git**](https://github.com/your-username/ml_project.git)

3. Push code to GitHub :

**git push -u origin main**

**6. Cloning Repos (Start from GitHub)**

To download an existing repo:

**git clone https://github.com/username/project.git**

**cd project**

**7. Basic Daily Workflow**

1. Check changes

**git status**

2. See differences

**git diff**

3. Stage changes

**git add file.py** or

**git add .**

4. Commit changes

**git commit -m "Added training function"**

5. Push changes

**git push**

**8. Branching (Working on Features)**

Branches = independent workspaces.

Create branch:

**git branch feature-login**

Switch to branch:

**git checkout feature-login**

Shortcut (create + switch):

**git checkout -b feature-login**

Push branch to GitHub:

**git push -u origin feature-login**

This allows you to work on features without breaking main code .

**9. Merging Branches**

1. Switch to `main`:

**git checkout main**

2. Merge branch:

**git merge feature-login**

**10. Handling Conflicts**

If two people edit the same line, Git will ask you to resolve conflicts :

Git marks conflicts in file with `<<<<<<<`, `=======`, `>>>>>>>`.

You edit file manually → keep correct version.

Then:

**git add fixed\_file.py**

**git commit -m "Resolved merge conflict"**

**11. Collaboration Workflows**

Fork & PR (open source) → Copy someone’s repo → make changes → submit PR.

Feature branch (team) → Each dev works in separate branch → PR → review → merge.

Git Flow → Special branches (`main`, `develop`, `feature`, `hotfix`).

**12. Undoing Mistakes**

Undo staged file:

**git reset file.py**

Undo last commit (keep changes):

**git reset --soft HEAD~1**

Undo last commit (discard changes):

**git reset --hard HEAD~1**

Restore file:

**git checkout -- file.py**

**13. GitHub Issues & Project Boards**

Issues = tasks, bug reports, feature requests.

Labels = tags (bug, enhancement).

Assignees = who will do the task.

Project Boards (Kanban) = organize workflow (To-Do → In Progress → Done).

Great for ML teams:

Track experiments (each experiment = issue).

Assign dataset cleaning / model training tasks.

Manage deadlines.

**14. ML-Specific Git Skills**

**Git LFS (Large File Storage)**

Normal Git is bad with big files (>100MB).

For datasets/models, use Git LFS :

**git lfs install**

**git lfs track " .csv"**

**git add .gitattributes**

**git commit -m "Track large files"**

**git push**

Now big files like datasets/models are stored efficiently.

**Ignore files**

Use `.gitignore` to skip files you don’t want in Git (checkpoints, logs, data).

Example `.gitignore` for ML:

\_\_pycache\_\_/

.pyc

.pkl

.h5

.csv

.json

.log

data/

models/

venv/

**15. Best Practices for Developers**

Commit often, small meaningful commits.

Write descriptive commit messages.

Always pull before pushing (`git pull origin main`).

Use branches for features/bugs.

Never commit datasets directly without LFS.

Use `.gitignore` to keep repo clean.

Use PRs for team code review.