FREE · LEARN.AIWITHPRIYABRAT.COM

The Git & GitHub Cheat Sheet

From your first commit to pull requests, rebasing, undoing mistakes and Git for AI projects. Every terminal session was run for real (Git 2.50.0) and shows its actual output.

By **Priyabrat Dalbehera** · GenAI engineer

11 pages | **10** parts | **50** real terminal sessions | **10** "oh no" fixes

What's inside

1 **Setup & mental model** p. 2
config, the four areas, init, clone, .gitignore, inside .git

2 **Everyday workflow** p. 3
status, add, commit, diff, log, show, good commit messages

3 **Branches & merging** p. 4
switch, fast-forward and 3-way merges, conflicts, detached HEAD

4 **Remotes & GitHub** p. 5
push, fetch vs pull, rejected pushes, PRs, forks, SSH

5 **Undo anything** p. 6
restore, amend, reset modes, revert, reflog, lost branches

6 **Rebase & history** p. 7
rebase, interactive rebase, cherry-pick, squash, force-push

7 **Stash, tags & detective tools** p. 8
stash, tags, blame, bisect, worktrees, aliases

8 **Collaborating on GitHub** p. 9
PR workflow, reviews, branch rules, Actions CI, gh CLI

9 **Git for AI & Python projects** p. 10
leaked secrets, LFS, notebooks, pre-commit, AI assistants

10 **Oh no! & interview answers** p. 11
10 accident fixes, 12 quick answers, how Git stores commits

How to read it
Lines starting with `$` are commands; the grey lines under them are the real output.

Tip & Watch out
Green boxes are good habits. Amber boxes are traps that lose work or leak secrets.

The example
Every session uses the same small Python chatbot project, and a GitHub repo for it.

learn.aiwithpriyabrat.com

© 2026 Priyabrat Dalbehera · Free to download and share unchanged, with credit · Not for resale

1.1 First-time setup (once per computer)

```
$ git config --global user.name "Asha Rao"
$ git config --global user.email "asha@example.com"
$ git config --global init.defaultBranch main
$ git config --global core.editor "code --wait"
$ git config --global pull.rebase true    # optional
$ git config --list --show-origin         # check
```

Use the same email as your GitHub account so commits link to your profile. Line endings: `core.autocrlf true` on Windows, `input` on macOS and Linux.

1.2 The mental model: four places your code lives

```
working folder      your files
  | git add             ▲ git restore
  ▼
staging area        the next commit
  | git commit          ▲ git restore --staged
  ▼
local repo          history in .git/
  | git push            ▲ git pull · git fetch
  ▼
GitHub (origin)     the shared copy
```

Working folder	the files you edit
Staging area	what goes into the next commit (also called the index)
Local repo	the commit history in `.git/`, on your computer
Remote	a copy on GitHub, usually called `origin`

1.3 Start a new repository

```
$ git init
Initialized empty Git repository in ~/chatbot/.git/
$ git status
On branch main

No commits yet

nothing to commit
```

1.4 Copy a repository from GitHub

```
$ cd ..
$ git clone https://github.com/asha/chatbot.git chatbot-copy
Cloning into 'chatbot-copy'...
done.
$ cd chatbot-copy
$ git log --oneline
12aa948 Add requirements
8f89541 Add app
8c4414e Add README
```

1.5 .gitignore: files Git should never track

```
$ cat .gitignore
.venv/
__pycache__/
.env
*.log
$ touch .env debug.log notes.md
$ git status --short
?? .gitignore
?? notes.md
$ git check-ignore -v .env
.gitignore:3:.env       .env
```

Tip: `.gitignore` only affects **untracked** files. Already committed one by mistake? `git rm --cached file`, then commit.

1.6 Key words

commit	a snapshot of the whole project, with a message, an author and a parent
hash	the commit's ID, e.g. `3f2a9c1` (the first 7 characters are enough)
branch	a movable label pointing at a commit
HEAD	where you are now: usually the current branch
origin	the default name of the remote you cloned from
upstream	the remote branch your branch tracks (or the original repo of a fork)

1.7 Install and check

```
$ winget install --id Git.Git -e    # Windows
$ brew install git                  # macOS
$ sudo apt install git              # Ubuntu / Debian
$ git --version
```

On Windows, Git Bash and the Git Credential Manager come with the installer.

1.8 Where settings live (and which one wins)

```
$ git config user.email
asha@example.com
$ git config --local user.email "asha@work.com"
$ git config user.email
asha@work.com
```

`--system`	every user on the computer
`--global`	you: `~/.gitconfig`
`--local`	this repo only: `.git/config` (wins)

Tip: use `--local` for a work email in work repos and keep your personal one global.

1.9 What's inside .git

```
$ ls .git
COMMIT_EDITMSG
HEAD
config
description
hooks
index
info
logs
objects
refs
$ cat .git/HEAD
ref: refs/heads/main
```

Watch out: `.git/` **is** the history. Delete it and the folder is no longer a repository; never edit files in it by hand.

1.10 Get help fast

```
$ git switch -h          # short summary of options
$ git help switch        # full manual page
$ git help everyday      # the essential commands guide
```

learn.aiwithpriyabrat.com

2.1 What changed? git status

```
$ git status
On branch main
Changes not staged for commit:
        modified:   app.py

Untracked files:
        notes.md

no changes added to commit
$ git status --short     # M = modified, ?? = new
 M app.py
?? notes.md
```

2.2 Stage and commit

```
$ git add app.py
$ git commit -m "Greet the user by name"
[main 04b62e0] Greet the user by name
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git status --short
?? notes.md
```

`git add .` stages everything · `git add -p` picks hunks one by one · `git commit -am "msg"` stages tracked files and commits in one step.

2.3 See the exact changes: git diff

```
$ git diff              # working folder vs staging area
diff --git a/app.py b/app.py
index 2f9a147..3dff4a2 100644
--- a/app.py
+++ b/app.py
@@ -1 +1 @@
-print("Hello")
+print("Hello, Asha")
$ git add app.py
$ git diff --staged --stat
 app.py | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

`git diff`	not yet staged
`git diff --staged`	staged, not committed
`git diff main..feature`	between two branches
`git diff HEAD~1`	since one commit ago

2.4 History: git log

```
$ git log --oneline
12aa948 Add requirements
8f89541 Add app
8c4414e Add README
$ git log --format="%h %ad %an  %s" --date=short -2
12aa948 2026-10-01 Asha Rao  Add requirements
8f89541 2026-10-01 Asha Rao  Add app
$ git log --oneline -- app.py   # one file only
8f89541 Add app
```

2.5 One commit in detail

```
$ git show --stat HEAD
commit 12aa9487892019fff750661b51a75bb9c0c61a98
Author: Asha Rao <asha@example.com>
Date:   Thu Oct 1 09:04:00 2026 +0530

    Add requirements

 requirements.txt | 1 +
 1 file changed, 1 insertion(+)
```

2.6 Rename and remove

```
$ git mv app.py main.py
$ git rm --cached notes.md   # untrack, keep the file
rm 'notes.md'
$ git status --short
R  app.py -> main.py
D  notes.md
?? notes.md
```

2.7 Commit messages that help

- Imperative and specific: *Add retry to the LLM client*, not *fixed stuff*.
- Subject up to ~50 characters; a blank line; then **why**, wrapped at 72.
- One logical change per commit, so it's easy to review and revert.
- Conventional Commits: `feat:` `fix:` `docs:` `refactor:` `test:` `chore:`

2.8 Stage part of a file: git add -p

`y` / `n`	stage / skip this hunk
`s`	split it into smaller hunks
`e`	edit the hunk by hand
`q`	quit; staged hunks stay staged

Use it when one file holds two unrelated changes: commit them separately. `git restore -p` discards hunk by hunk the same way.

2.9 Find commits: log filters

```
$ git log --oneline main..feature/search   # only on the branch
1b1468c Add search
$ git log --oneline --grep="licence"
6f7daf6 Add licence
$ git log --oneline --author="Asha" -2
6f7daf6 Add licence
12aa948 Add requirements
```

More filters: `--since="2 weeks ago"` · `-n 5` · `--stat` · `-p` (with diffs) · `--merges` · `--no-merges`.

2.10 Point at commits: HEAD~ and friends

```
$ git show -s --oneline HEAD~1
8f89541 Add app
$ git show -s --oneline HEAD~2
8c4414e Add README
```

`HEAD`	the current commit
`HEAD~1` / `HEAD~3`	1 / 3 commits back
`HEAD^2`	the second parent of a merge commit
`main@{1}`	where main was one move ago (reflog)
`v1.0`, `3f2a9c1`	a tag or a (short) hash

learn.aiwithpriyabrat.com

3.1 Create, switch and list branches

```
$ git switch -c feature/login      # create + switch
Switched to a new branch 'feature/login'
$ git branch
* feature/login
  main
$ git switch main
Switched to branch 'main'
$ git branch -m feature/login feature/sign-in    # rename
```

Older equivalent: `git checkout -b name`. Delete a merged branch: `git branch -d name` (`-D` forces). All branches including remote ones: `git branch -a`.

3.2 Fast-forward merge (main didn't move)

```
$ git switch main
Switched to branch 'main'
$ git merge feature/login
Updating 12aa948..891f817
Fast-forward
 login.py | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 login.py
```

3.3 Three-way merge (both branches moved)

```
$ git merge --no-edit feature/search
Merge made by the 'ort' strategy.
 search.py | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 search.py
$ git log --oneline --graph
*   e82a525 Merge branch 'feature/search'
|\
| * 1b1468c Add search
* | 6f7daf6 Add licence
|/
* 12aa948 Add requirements
* 8f89541 Add app
…
```

3.4 Resolve a merge conflict

```
$ git merge feature/greeting
Auto-merging app.py
CONFLICT (content): Merge conflict in app.py
Automatic merge failed; fix conflicts and then commit the result.
$ cat app.py
<<<<<<< HEAD
print("Hello, world")
=======
print("Hi there")
>>>>>>> feature/greeting
$ code app.py    # keep the right line, delete the markers
$ git add app.py
$ git commit --no-edit
[main 7a7db77] Merge branch 'feature/greeting'
```

Tip: stuck? `git merge --abort` puts everything back as it was before the merge.

3.5 Detached HEAD: looking at an old commit

```
$ git switch --detach HEAD~2
HEAD is now at 8c4414e Add README
$ git status
HEAD detached at 8c4414e
nothing to commit, working tree clean
$ git switch -c experiment    # keep work done here
Switched to a new branch 'experiment'
```

Detached means HEAD points at a commit, not a branch. New commits there are lost when you switch away unless you create a branch.

3.6 Branch naming and strategy

`feature/…`	new work
`fix/…`	bug fixes
`chore/…`, `docs/…`	maintenance, docs

Trunk-based: short-lived branches merged into `main` daily (most teams today). **Git Flow**: long-lived `develop` and release branches (versioned products).

3.7 Which branches are merged?

```
$ git branch --merged
  feature/login
* main
$ git branch --no-merged
  feature/wip
$ git branch -d feature/login
Deleted branch feature/login (was 891f817).
```

`-d` refuses to delete unmerged work; `-D` forces it.

3.8 Delete remote branches and tidy up

```
$ git push origin --delete feature/login
$ git fetch --prune          # forget deleted remote branches
$ git config --global fetch.prune true    # always prune
```

3.9 Conflict toolbox

```
$ git merge feature/greeting
Auto-merging app.py
…
$ git status --short           # UU = conflict in both
UU app.py
$ git checkout --theirs app.py # take their version
Updated 1 path from the index
$ cat app.py
print("Hi there")
```

`--ours` / `--theirs`	take one side for a whole file
`git diff`	show both sides of each conflict
VS Code	Accept Current / Incoming / Both, or the merge editor

Tip: fewer conflicts: pull often, keep PRs small, and don't reformat whole files in a feature branch.

learn.aiwithpriyabrat.com

4.1 Put a local project on GitHub

```
$ git remote add origin https://github.com/asha/chatbot.git
$ git push -u origin main
branch 'main' set up to track 'origin/main'.
To https://github.com/asha/chatbot.git
 * [new branch]      main -> main
$ git remote -v
origin  https://github.com/asha/chatbot.git (fetch)
origin  https://github.com/asha/chatbot.git (push)
```

Create the empty repo on github.com first, or in one step: `gh repo create chatbot --private --source=. --push`.

4.2 fetch (look) vs pull (look + merge)

```
$ git fetch
From https://github.com/asha/chatbot
   12aa948..ba42eea  main       -> origin/main
$ git status -sb
## main...origin/main [behind 1]
$ git pull
Updating 12aa948..ba42eea
Fast-forward
 requirements.txt | 1 +
 1 file changed, 1 insertion(+)
```

Tip: `fetch` is always safe: it only updates `origin/main`. `pull` = `fetch` + `merge` (or `rebase`).

4.3 Push rejected? Someone pushed first

```
$ git push
To https://github.com/asha/chatbot.git
 ! [rejected]        main -> main (fetch first)
…
$ git pull --rebase
From https://github.com/asha/chatbot
   12aa948..ba42eea  main       -> origin/main
…
$ git push
To https://github.com/asha/chatbot.git
   ba42eea..d3b3f8d  main -> main
```

The rejected push ends with *error: failed to push some refs* and a hint to pull first. `pull --rebase` puts your commit on top of theirs, so the history stays a straight line.

4.4 Share a feature branch, then open a pull request

```
$ git switch -c feature/faq
Switched to a new branch 'feature/faq'
$ git commit -q --allow-empty -m "Add FAQ answers"
$ git push -u origin feature/faq
branch 'feature/faq' set up to track 'origin/feature/faq'.
To https://github.com/asha/chatbot.git
 * [new branch]      feature/faq -> feature/faq
```

```
$ gh pr create --fill          # opens the pull request
$ gh pr view --web
```

Without the `gh` CLI, GitHub prints a link after the push: open it and click **Create pull request**.

4.5 Contribute to someone else's project (fork)

```
$ gh repo fork owner/project --clone   # fork + clone
$ git remote -v       # origin = your fork, upstream = theirs
$ git fetch upstream
$ git rebase upstream/main            # catch up
$ git push --force-with-lease         # update your PR
```

4.6 Sign in: HTTPS or SSH

```
$ gh auth login                       # easiest: HTTPS + browser
$ ssh-keygen -t ed25519 -C "asha@example.com"
$ gh ssh-key add ~/.ssh/id_ed25519.pub
$ ssh -T git@github.com               # test SSH
```

Watch out: GitHub doesn't accept account passwords for Git. Use `gh auth login`, a credential manager with a personal access token, or SSH keys.

4.7 What does each branch track?

```
$ git branch -vv
  feature/faq 12aa948 Add requirements
* main        a901515 [origin/main: ahead 1] Add usage
```

`ahead 1` = one commit to push · `behind 2` = two to pull · no brackets = no upstream yet (push with `-u`).

4.8 Bring your branch up to date with main

```
$ git fetch
From https://github.com/asha/chatbot
   12aa948..ba42eea  main       -> origin/main
$ git rebase origin/main
Rebasing (1/1)
Successfully rebased and updated refs/heads/feature/faq.
$ git log --oneline -3
fe8d3a0 Add FAQ
ba42eea Add flask
12aa948 Add requirements
```

Shared branch? Use `git merge origin/main` instead, so nobody's history is rewritten.

4.9 Settings worth knowing

`pull.rebase true`	pull rebases your local commits instead of merging
`pull.ff only`	pull refuses to create merge commits
`push.autoSetupRemote true`	first push sets the upstream (no `-u`)
`fetch.prune true`	drop remote branches deleted on GitHub
`rebase.autoStash true`	stash and restore local edits around a rebase

Set each with `git config --global name value`.

learn.aiwithpriyabrat.com

5.1 Throw away changes in a file

```
$ git status --short
 M app.py
?? notes.md
$ git restore app.py
$ git status --short
?? notes.md
```

Watch out: `git restore` discards the edits for good: they were never committed.

5.2 Unstage (keep the edits)

```
$ git add app.py
$ git restore --staged app.py
$ git status --short
 M app.py
?? notes.md
```

5.3 Fix the last commit (not pushed yet)

```
$ git add notes.md              # forgot this file
$ git commit --amend --no-edit
[main 28ef590] Add requirements
 Date: Thu Oct 1 09:04:00 2026 +0530
 2 files changed, 2 insertions(+)
…
$ git commit --amend -m "Add requirements and notes"
[main e005538] Add requirements and notes
 Date: Thu Oct 1 09:04:00 2026 +0530
 2 files changed, 2 insertions(+)
…
```

5.4 git reset: move the branch back

```
$ git reset --soft HEAD~1
$ git status --short
A  requirements.txt
```

`--soft`	undo the commit, keep changes **staged**
`--mixed` (default)	undo the commit, keep changes **unstaged**
`--hard`	undo the commit **and delete** the changes

5.5 git revert: undo a pushed commit safely

```
$ git revert --no-edit HEAD
[main 96314b3] Revert "Add requirements"
 Date: Thu Oct 1 09:05:00 2026 +0530
 1 file changed, 1 deletion(-)
 delete mode 100644 requirements.txt
```

Revert adds a **new** commit that reverses the old one, so shared history isn't rewritten. Reset rewrites history: use it only on commits nobody else has.

5.6 The safety net: git reflog

```
$ git reset --hard HEAD~2          # oops
HEAD is now at 8c4414e Add README
$ git reflog -3
8c4414e HEAD@{0}: reset: moving to HEAD~2
12aa948 HEAD@{1}: commit: Add requirements
8f89541 HEAD@{2}: commit: Add app
$ git reset --hard HEAD@{1}        # back again
HEAD is now at 12aa948 Add requirements
```

5.7 Recover a deleted branch

```
$ git branch -D feature/login
Deleted branch feature/login (was 891f817).
$ git branch feature/login 891f817   # hash from above
$ git log --oneline -1 feature/login
891f817 Add login
```

5.8 Remove untracked files

```
$ git clean -n      # dry run: what would go
Would remove debug.log
Would remove notes.md
$ git clean -fd     # really delete files and folders
Removing debug.log
Removing notes.md
```

5.9 Get one file back from an old commit

```
$ git restore --source=HEAD~1 app.py
$ cat app.py
print("Hello")
$ git status --short
 M app.py
```

5.10 Undo a merge

```
$ git log --oneline -1
e82a525 Merge branch 'feature/search'
$ git reset --hard ORIG_HEAD     # merge not pushed yet
HEAD is now at 6f7daf6 Add licence
```

Already pushed? `git revert -m 1 <merge-hash>` undoes it with a new commit (`-m 1` keeps the main-line side).

5.11 Which undo do I need?

Situation	Command
Edits in a file, not staged	`git restore file`
Staged by mistake	`git restore --staged file`
Last commit wrong, not pushed	`git commit --amend`
Unpushed commits, want them gone	`git reset --hard HEAD~n`
…but keep their changes	`git reset --soft HEAD~n`
Commit already pushed	`git revert <hash>`
Everything is a mess	`git reflog`, reset to a good entry

learnaiwithpriyabrat.com

6.1 Rebase a branch onto the latest main

```
$ git switch feature/search
Switched to branch 'feature/search'
$ git rebase main
Rebasing (1/1)
Successfully rebased and updated refs/heads/feature/search.
$ git log --oneline --graph
* cd5579d Add search
* 6f7daf6 Add licence
* 12aa948 Add requirements
* 8f89541 Add app
* 8c4414e Add README
```

Rebase replays your commits on top of `main` : same changes, new hashes, a straight line.

6.2 Merge or rebase?

	merge	rebase
history	true, with merge commits	straight line, rewritten
safe on shared branches	yes	no: only your own
conflicts	solved once	may repeat per commit
typical use	merging PRs into main	updating your branch

Watch out: the golden rule: never rebase commits that others have already pulled.

6.3 Clean up commits before a PR: interactive rebase

```
pick   1a2b3c4 Add FAQ        # the editor opens a list like this:
fixup  5d6e7f8 wip            # fixup = melt into the commit above
fixup  9a0b1c2 fix typo       # also: squash, reword, edit, drop
```

```
$ git log --oneline -3
7423e17 fix typo
6504811 wip
b103974 Add FAQ
$ git rebase -i HEAD~3
Rebasing (2/3)
Rebasing (3/3)
Successfully rebased and updated refs/heads/feature/faq.
$ git log --oneline -2
fbbbb6c Add FAQ
12aa948 Add requirements
```

6.4 Copy one commit to another branch: cherry-pick

```
$ git cherry-pick 7341ad7
[main fa58023] Fix greeting punctuation
 Date: Thu Oct 1 09:06:00 2026 +0530
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log --oneline -2
fa58023 Fix greeting punctuation
12aa948 Add requirements
```

6.5 Squash a whole branch into one commit

```
$ git merge --squash feature/faq
Updating 12aa948..7423e17
Fast-forward
Squash commit -- not updating HEAD
 FAQ.md | 3 +++
…
$ git commit -m "Add FAQ page"
[main 354261c] Add FAQ page
 1 file changed, 3 insertions(+)
 create mode 100644 FAQ.md
```

GitHub's **Squash and merge** button does the same: one tidy commit per pull request.

6.6 Rewrote history? Push safely

```
$ git push --force-with-lease    # refuses if someone else pushed
```

Watch out: plain `--force` overwrites teammates' commits without asking. Never force-push `main` .

6.7 Conflicts during a rebase

```
$ git status                  # see the conflicted files
$ git add app.py              # after fixing them
$ git rebase --continue       # replay the next commit
$ git rebase --skip           # drop this commit
$ git rebase --abort          # back to before the rebase
```

Tip: rebase replays commits one by one, so the same conflict can come back. `git config --global rerere.enabled true` remembers your fixes.

6.8 Fix an earlier commit: fixup + autosquash

```
$ echo "A: Yes, it's free." >> FAQ.md
$ git commit -qa --fixup 39e6e30
$ git log --oneline -3
981c147 fixup! Add FAQ
eee450c Add install guide
39e6e30 Add FAQ
$ git rebase -i --autosquash HEAD~3
Rebasing (2/3)
Rebasing (3/3)
Successfully rebased and updated refs/heads/feature/faq.
$ git log --oneline -2
2d8bc6f Add install guide
3610855 Add FAQ
```

The editor opens with the `fixup!` commit already moved under its target: save and close.

6.9 Interactive rebase commands

`pick`	keep the commit
`reword`	keep it, edit the message
`edit`	stop to change the commit
`squash`	merge into the one above, combine messages
`fixup`	merge into the one above, keep its message
`drop`	delete the commit
reorder lines	reorder the commits

learn.aiwithpriyabrat.com

7.1 Park unfinished work: git stash

```
$ git stash push -m "half-done greeting"
Saved working directory and index state On main: half-done greeting
$ git stash list
stash@{0}: On main: half-done greeting
$ git stash pop
On branch main
Changes not staged for commit:
        modified:   app.py

no changes added to commit
Dropped refs/stash@{0} (875a03f773c2d49ae5f562b0f3cf7bc9211dc836)
```

`git stash -u` also stashes new files · `git stash apply` keeps the stash · `git stash drop` deletes it.

7.2 Tag a release

```
$ git tag -a v1.0 -m "First release"
$ git describe
v1.0
$ git commit -q --allow-empty -m "Start v1.1"
$ git describe
v1.0-1-gedab945
```

```
$ git push origin v1.0
$ gh release create v1.0 --generate-notes
```

Tags aren't pushed by `git push`: push them by name, or all at once with `--tags`.

7.3 Who changed this line? git blame

```
$ git blame -s app.py
8f895410 1) print("Hello")
$ git log --oneline -S "Hello"   # when was it added?
8f89541 Add app
```

7.4 Find the commit that broke something: bisect

```
$ git bisect start HEAD 7aaa001   # bad, good
Bisecting: 1 revision left to test after this (roughly 1 step)
[cb276cc51df140316663f629331018acaaf6f298] Shorter greeting
$ git bisect run sh check.sh
running 'sh' 'check.sh'
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[8affbeb130b8545f7bcf985c2934f66ea6064625] Add licence
running 'sh' 'check.sh'
cb276cc51df140316663f629331018acaaf6f298 is the first bad commit
…
$ git bisect reset
Previous HEAD position was 8affbeb Add licence
Switched to branch 'main'
```

Bisect does a binary search: 1,000 commits take about 10 tests. `check.sh` exits 0 for good and 1 for bad; without a script, mark each step with `git bisect good` / `bad`.

7.5 Shortcuts: aliases

```
$ git config --global alias.lg "log --oneline --graph --all"
$ git config --global alias.st "status -sb"
$ git lg
* 6f7daf6 Add licence
| * 1b1468c Add search
|/
* 12aa948 Add requirements
* 8f89541 Add app
…
```

7.6 Search the code at any point in history

```
$ git grep -n "def search"        # search tracked files
$ git log -p -- app.py            # every change to a file
$ git show v1.0:app.py            # a file as it was at v1.0
$ git shortlog -sn                # commits per author
```

7.7 Two branches at once: worktrees

```
$ git worktree add ../chatbot-hotfix -b hotfix
Preparing worktree (new branch 'hotfix')
HEAD is now at 12aa948 Add requirements
$ git worktree list
~/chatbot          12aa948 [main]
~/chatbot-hotfix   12aa948 [hotfix]
$ git worktree remove ../chatbot-hotfix
```

Tip: no stashing or switching: each folder has its own branch. Handy for a quick fix, or for letting an AI coding agent work on a branch while you keep working on yours.

7.8 Quick moves

```
$ git switch -                # back to the previous branch
$ git diff --word-diff        # changes word by word
$ git log -p -1               # the last commit with its diff
$ git restore -p app.py       # discard hunk by hunk
```

7.9 Submodules in 30 seconds

```
$ git submodule add https://github.com/org/lib libs/lib
$ git clone --recurse-submodules <url>
$ git submodule update --init --recursive
```

A submodule pins another repo at one commit. Often a package (`pip install`) is simpler.

learn.aiwithpriyabrat.com

8.1 The pull-request workflow, step by step

1	Start from fresh main	`git switch main && git pull`
2	Branch	`git switch -c fix/login-timeout`
3	Commit small steps	`git add -p` · `git commit`
4	Push	`git push -u origin fix/login-timeout`
5	Open the PR	`gh pr create --fill`
6	CI runs, teammates review	fix feedback with new commits
7	Merge	squash and merge on GitHub
8	Clean up	`git switch main && git pull` · `git branch -d fix/…`

8.2 A pull request reviewers like

- **Title** says what changes: *Retry LLM calls on 429 errors.*
- **Description**: why, what changed, how you tested it, screenshots for UI.
- Link the issue: `Closes #42` closes it automatically on merge (also `Fixes`, `Resolves`).
- Keep it small: under ~400 changed lines gets a real review.
- Open a **draft** PR early to get feedback on direction.

8.3 Reviewing someone's PR

```
$ gh pr checkout 42          # run their branch locally
$ gh pr diff 42
$ gh pr review 42 --approve
$ gh pr review 42 --request-changes -b "See comments"
```

In a review comment, a ```` ```suggestion ```` block lets the author apply your fix with one click.

8.4 Protect main (Settings → Rules)

- ☐ Require a pull request before merging
- ☐ Require 1+ approving review
- ☐ Require status checks (CI) to pass
- ☐ Block force pushes and deletions
- ☐ Optional: require linear history, signed commits

8.5 CI with GitHub Actions: test every push

.github/workflows/tests.yml

```
name: tests
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: "3.12"
      - run: pip install -r requirements.txt pytest
      - run: pytest -q
```

Tip: store API keys in **Settings → Secrets and variables → Actions** and read them as `${{ secrets.OPENAI_API_KEY }}`, never in the code.

8.6 Files GitHub understands

`README.md`	the project's front page
`LICENSE`	how others may use it
`.github/pull_request_template.md`	pre-filled PR description
`.github/ISSUE_TEMPLATE/`	bug and feature forms
`CODEOWNERS`	auto-request the right reviewers
`.github/dependabot.yml`	automatic dependency updates

8.7 gh CLI: GitHub from the terminal

```
$ gh repo clone owner/project
$ gh issue list --assignee @me
$ gh issue create --title "Bug: timeout" --body "..."
$ gh pr list · gh pr status
$ gh pr merge --squash --delete-branch
$ gh run watch                 # follow the CI run
```

8.8 The merge button: three choices

Option	What lands on main	Choose for
Create a merge commit	keeps every commit, adds a merge commit	full history
Squash and merge	one commit per PR	clean main (most common)
Rebase and merge	replays the commits, no merge commit	linear, keeps each commit

8.9 Versions and releases (SemVer)

MAJOR 2.0.0	breaking change
MINOR 1.5.0	new feature, backwards-compatible
PATCH 1.4.3	bug fix
pre-release 2.0.0-rc.1	release candidate for testing

Tag as `v1.4.3`; `gh release create --generate-notes` lists the merged PRs since the last tag.

8.10 Issues, Projects and the browser

- **Issues** with templates and labels (`bug`, `good first issue`); `gh issue develop 42 --checkout` makes a linked branch.
- **Projects**: a board or table across issues and PRs.
- Press `.` on any repo to open it in the github.dev editor; **Codespaces** gives a full cloud dev machine.
- **GitHub Pages** hosts a static site from a branch or an Actions workflow.

learn.aiwithpriyabrat.com

9.1 A .gitignore for Python and ML projects

```
# environments and secrets
.venv/  venv/  .env  *.pem
# Python
__pycache__/  *.pyc  .pytest_cache/  dist/
# notebooks
.ipynb_checkpoints/
# data, models, experiment logs
data/raw/  models/  *.pt  *.onnx  *.safetensors
mlruns/  wandb/  chroma_db/  .cache/
```

Write one pattern per line in the real file. Ready-made templates: github.com/github/gitignore (*Python.gitignore*).

9.2 Committed a secret? Untracking isn't enough

```
$ git rm --cached .env
rm '.env'
$ echo ".env" >> .gitignore
$ git add .gitignore
$ git commit -qm "Stop tracking .env"
$ git log --oneline -- .env    # still in history!
b0d5669 Stop tracking .env
907f3c6 Add config
```

Watch out: Rotate the key first: assume it's stolen the moment it's pushed. Then remove it from history with `git filter-repo --invert-paths --path .env` and force-push. Commit a `.env.example` with empty values instead.

9.3 Large files: models and datasets

```
$ git lfs install
$ git lfs track "*.safetensors"
$ git add .gitattributes model.safetensors
```

GitHub blocks files over **100 MB** and warns above 50 MB. Use Git LFS for a few big files; keep big models on the Hugging Face Hub and datasets in DVC or cloud storage, with only code and configs in Git.

9.4 Jupyter notebooks

- Outputs and execution counts make noisy diffs and leak data: strip them with `nbstripout --install`.
- Readable notebook diffs: `nbdiff` from nbdime.
- Move reusable code into `.py` modules; keep notebooks for exploration.

9.5 Catch problems before they're committed: pre-commit

.pre-commit-config.yaml

```
repos:
  - repo: https://github.com/astral-sh/ruff-pre-commit
    rev: v0.6.9
    hooks: [{id: ruff}, {id: ruff-format}]
  - repo: https://github.com/gitleaks/gitleaks
    rev: v8.21.2
    hooks: [{id: gitleaks}]
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v5.0.0
    hooks: [{id: check-added-large-files}]
```

```
$ pip install pre-commit && pre-commit install
```

Tip: enable GitHub **push protection** (Settings → Code security): it blocks pushes that contain known API key formats.

9.6 Hugging Face model repos are Git repos

```
$ git lfs install
$ git clone https://huggingface.co/BAAI/bge-small-en-v1.5
$ GIT_LFS_SKIP_SMUDGE=1 git clone <url>   # code, no big files
```

To fetch just the files your code needs, `huggingface_hub.snapshot_download()` is faster than cloning.

9.7 Version prompts, configs and experiments

- Keep prompts in files (`prompts/summarise_v3.txt`) and change them through PRs, with eval results in the description.
- One branch per experiment; merge only the winner.
- Log the commit hash with every run and model, so results are reproducible:

```
$ git rev-parse --short HEAD
12aa948
$ git status --porcelain | wc -l   # 0 = no uncommitted changes
0
```

9.8 Working with AI coding assistants

- ☐ Commit before the assistant edits: one `git restore .` undoes a bad run
- ☐ Read `git diff` before staging; stage with `git add -p`
- ☐ Small commits with clear messages, written or checked by you
- ☐ Let agents work on their own branch or worktree (7.7)
- ☐ Never paste secrets or `.env` into a prompt

9.9 .gitattributes: line endings, binaries, LFS

```
* text=auto eol=lf
*.bat text eol=crlf
*.png binary
*.safetensors filter=lfs diff=lfs merge=lfs -text
```

Fixes "every line changed" diffs between Windows and macOS/Linux teammates.

learn.aiwithpriyabrat.com

10.1 "Oh no!" → the fix

Situation	Fix
Committed to `main` instead of a branch (not pushed)	`git branch feature/x` · `git reset --hard origin/main` · `git switch feature/x`
Typo in the last commit message (not pushed)	`git commit --amend -m "…"`
Forgot a file in the last commit	`git add file` · `git commit --amend --no-edit`
Need to undo a commit that's already pushed	`git revert <hash>`
Lost commits after a reset or rebase	`git reflog` · `git reset --hard HEAD@{n}`
Deleted a branch	hash from `git reflog` · `git branch name <hash>`
A merge went wrong	during: `git merge --abort` · after (not pushed): `git reset --hard ORIG_HEAD`
Push rejected (fetch first)	`git pull --rebase` · `git push`
Work done on a detached HEAD	`git switch -c new-branch`
Pushed a password or API key	rotate it now · `git filter-repo` · force-push

10.2 Quick interview answers

Git vs GitHub	Git is the version-control tool; GitHub hosts repos and adds PRs, issues, Actions
merge vs rebase	merge joins histories with a merge commit; rebase replays commits on a new base
fetch vs pull	fetch downloads only; pull = fetch + merge (or rebase)
reset vs revert	reset moves the branch (rewrites); revert adds an undo commit (safe when shared)
What is a commit?	a snapshot of the tree + parent(s) + author + message, named by its hash
HEAD	the current position: usually a branch, sometimes a commit (detached)
Fast-forward	the branch label just moves ahead; no merge commit needed
Staging area	lets you choose exactly what goes into the next commit
origin vs upstream	your remote vs the original project you forked
--force-with-lease	force-push only if the remote hasn't changed since you fetched
cherry-pick	copy one commit onto the current branch
stash	shelve uncommitted changes to switch tasks

10.3 Revision checklist

- ☐ Draw the four areas and the commands between them
- ☐ Branch, commit, push, open a PR, merge, clean up
- ☐ Resolve a merge conflict by hand
- ☐ Choose between merge, rebase and squash for a situation
- ☐ Undo: restore, reset (3 modes), revert, reflog
- ☐ Explain why rebasing shared commits is dangerous
- ☐ What to do when a secret is pushed

10.4 Keep learning, free

- Tutorial: *Use Git every day*
- SQL Interview Cheat Sheet (21 pages)
- Python Cheat Sheet (22 pages)
- GenAI notes: RAG, agents, LLMOps

learn.aiwithpriyabrat.com

10.5 Deep dive: how Git stores a commit

```
$ git cat-file -t HEAD
commit
$ git cat-file -p HEAD
tree 265879042d3adafda971541dda0a589ff439b400
parent 8f895410b8c2d92bf12f9699816ca8fa1cc8ee58
author Asha Rao <asha@example.com> 1790825640 +0530
committer Asha Rao <asha@example.com> 1790825640 +0530

Add requirements
$ git ls-tree --abbrev HEAD     # the tree: one line per file
100644 blob 9854ab3     README.md
100644 blob 2f9a147     app.py
100644 blob f229360     requirements.txt
```

blob	a file's contents
tree	a folder: names → blobs and trees
commit	a tree + parent(s) + author + message

Every object is named by the hash of its content, so identical files are stored once and any change anywhere gives a new commit hash. That's why history can't be altered silently.

10.6 20 commands at a glance

`init` · `clone`	start / copy a repo
`status` · `diff`	what changed
`add` · `commit`	save a snapshot
`log` · `show`	read history
`switch` · `branch`	move between / manage branches
`merge` · `rebase`	combine work
`fetch` · `pull` · `push`	sync with GitHub
`stash`	shelve work
`restore` · `reset` · `revert`	undo
`cherry-pick`	copy a commit
`tag`	name a release
`reflog`	find lost commits
`blame` · `bisect`	find who / which commit

learn.aiwithpriyabrat.com