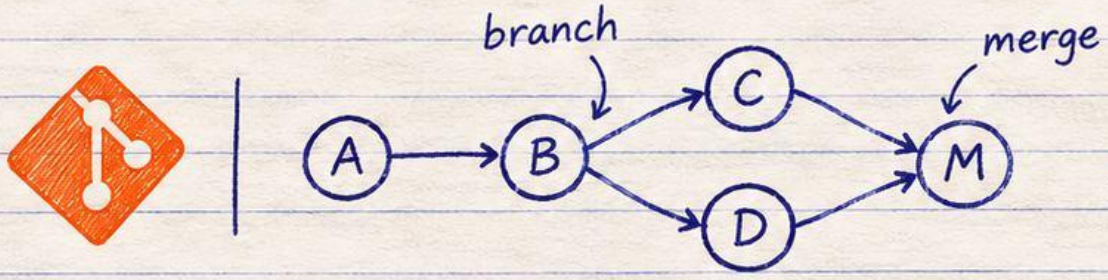
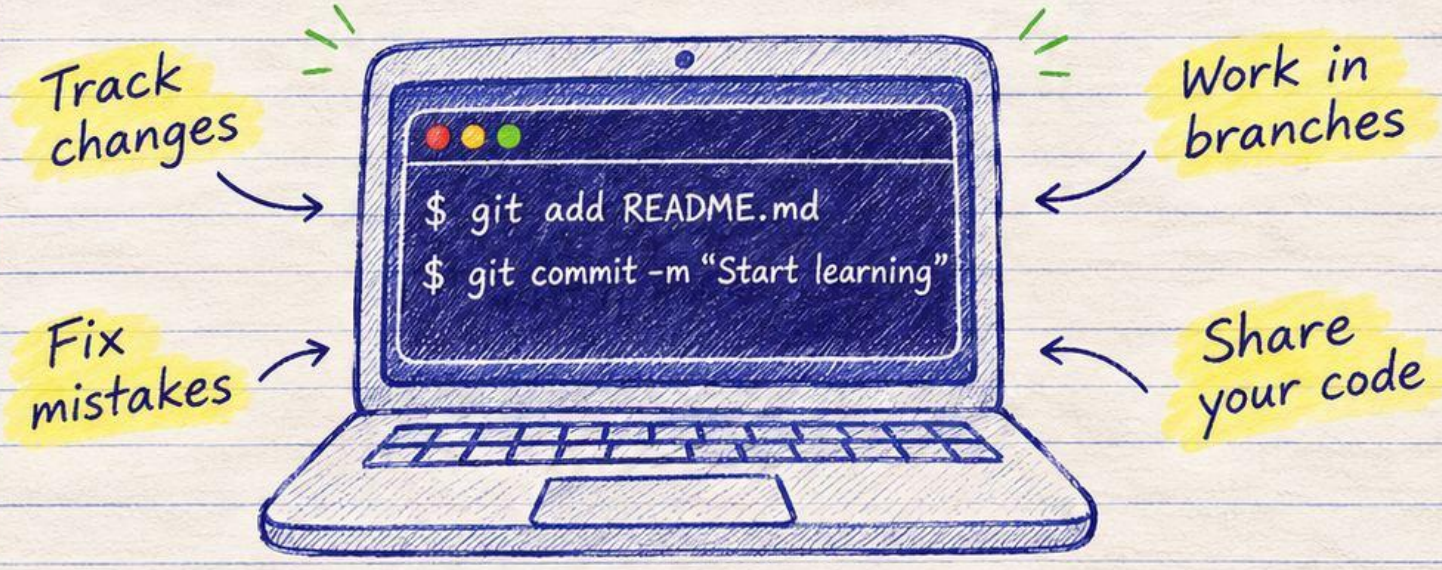
ZainSpire

GIT & GITHUB HANDWRITTEN NOTES

19 essential lessons. One clear workflow.





Edit → Review → Stage → Commit → Share

Only the essentials. Explained clearly.

GIT & GITHUB: THE BASICS

WHAT GIT DOES

- Git records versions of your project.
- A commit is a saved snapshot of tracked files.
- Compare changes, revisit old work and collaborate.
- Git works locally; you can commit offline.

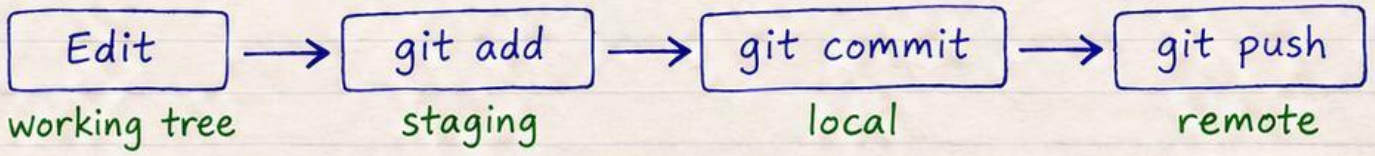
GIT VS GITHUB

- Git: the version-control tool on your computer.
- GitHub: an online host for Git repositories.
- GitHub adds pull requests, reviews and issues.
- You can use Git without GitHub.

FOUR PLACES TO KNOW

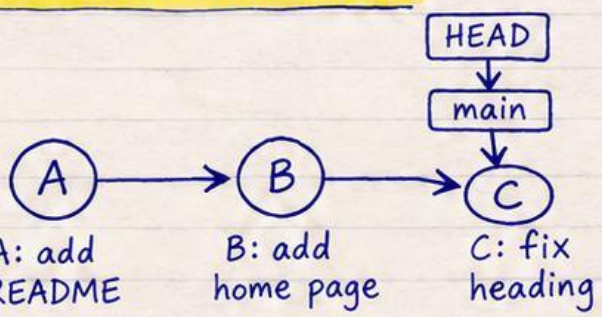
- **Working tree**: files you edit.
- **Staging area**: the next snapshot you prepare.
- **Local repository**: commits on your computer.
- **Remote repository**: shared history on a server.

THE BASIC FLOW



! A commit does not upload your work.

A SMALL TIMELINE



- Each circle represents a commit.
- A branch name points to a commit.
- HEAD usually points to your current branch.

REMEMBER

- Git is distributed: a normal clone has local history.
- main is a common branch name, not a special rule.
- A local repo alone is not an off-device backup.

03 / 20 — INSTALL & SET YOUR IDENTITY

INSTALL GIT

- Download for your system: git-scm.com/install/
- Windows: use the official Git installer.
- macOS: use Xcode tools or Homebrew.
- Debian / Ubuntu: use the system package manager.
- Open a new terminal after installation.

```
git --version
```

↓

Expected: a Git version number.

IDENTITY FOR NEW COMMITS

```
git config --global user.name "Zain"
```

```
git config --global user.email "zain@example.com"
```

- Replace the example email with your own verified GitHub email or your GitHub-provided no-reply email.
- The name is an author label, not your login.

CHOOSE A DEFAULT BRANCH

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

- This affects newly created repositories.
- It does not rename an existing branch.

CHECK THE SETTINGS

```
git config user.name
```

```
git config user.email
```

```
git config --list
```

↓

Expected name: Zain

GLOBAL VS LOCAL

- --global: a default for your user account.
- Inside a repo, omit --global for a local override:

```
git config user.name "Zain"
```

- Local settings override global settings for that repo.

REMEMBER

- Commit identity and GitHub authentication are separate.
- Use a recent Git version for switch and restore.

04 / 20 — YOUR FIRST REPOSITORY

START A NEW PROJECT

```
mkdir zain-notes
cd zain-notes
git init -b main
```

- This creates a repository with main as its branch.
- Git stores repository data in the hidden .git folder.

CREATE A FILE

- In your editor, create README.md containing:

```
# Zain's Git Notes
```

- Save the file before checking Git.

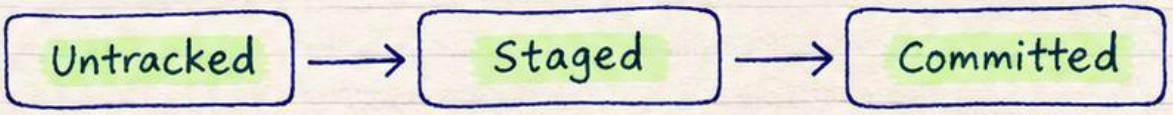
MAKE THE FIRST COMMIT

```
git status
git add README.md
git commit -m "Add project README"
git log --oneline
```

- Expected: one commit with your message.
- The commit ID will be different on your computer.

WHAT CHANGED?

- Before add: README.md is untracked.
- After add: README.md is staged.
- After commit: its snapshot is in local history.



CHECK YOUR LOCATION

- Run Git commands inside the project folder.

```
git status
```

- A clean working tree means no pending tracked edits or untracked files are being reported.

EXISTING PROJECT?

- Use git clone REPO_URL for an existing remote repo.
- Replace REPO_URL with its real address.
- Do not run git init again after cloning.
- Leave the .git folder alone.

05 / 20 — STATUS, STAGE & COMMIT

SEE WHAT IS READY

```
git status
```

```
git status --short
```

- Short status uses two columns: first = staging area; second = working tree.
- ?? means an untracked file.
- A in the first column means a staged addition.
- M means modified in the relevant column.

CHOOSE THE NEXT SNAPSHOT

`git add README.md`	→ Stage one file.
`git add .`	→ Stage changes under the current folder.
`git add -A`	→ Stage additions, modifications and deletions repo-wide.
`git add -p`	→ Choose changed sections interactively.

REVIEW, THEN COMMIT

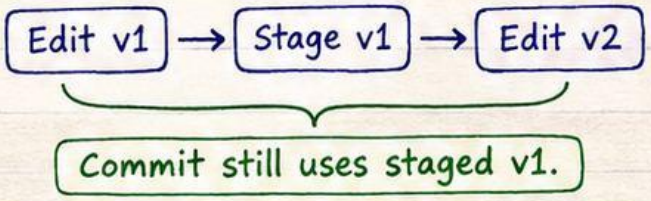
```
git diff --staged
```

```
git commit -m "Fix README heading"
```

- One clear change deserves one clear commit.
- A commit saves staged content, not every visible edit.

IMPORTANT DETAIL

- If you edit a file after staging it, stage it again to include the newer edits in the next commit.



SHORTCUT WITH A LIMIT

```
git commit -am "Fix tracked files"
```

- Automatically stages modified/deleted tracked files.
- It does not add new untracked files.

GOOD MESSAGE

✓ `"Add login validation"` is clearer than ✗ `"Update files"`.

06 / 20 — REVIEW CHANGES & HISTORY

THREE USEFUL COMPARISONS

- `git diff` → Unstaged tracked edits: working tree vs staging.
- `git diff --staged` → Staged changes vs the latest commit.
- `git diff HEAD` → Tracked changes in your working tree vs HEAD.

! New untracked files are not shown by these diffs.

READ A DIFF

- A line starting with - was removed.
- A line starting with + was added.
- Example:

```
- Hello
+ Salam, Zain!
```

- These signs are diff markers, not file content.

EXPLORE COMMITS

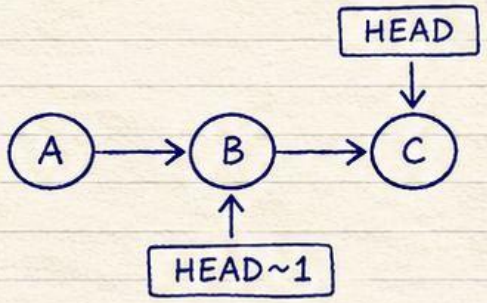
- `git log --oneline -5`
  → Show five recent commits.
- `git log --oneline --graph --all`
  → See branch history as a graph.
- `git log --oneline -- README.md`
  → Show commits affecting one file.

INSPECT A SNAPSHOT

- `git show HEAD` → Show the latest commit and its changes.
- `git show HEAD:README.md` → Read README.md from the latest commit.
- `git diff HEAD~1 HEAD` → Compare the previous commit with the latest.

! This needs at least two commits.

READ THE GRAPH



- Each node is a commit; connecting lines show ancestry.
- HEAD~1 means the first parent of HEAD.
- Commit IDs shown in examples are not universal.

07 / 20 — IGNORE, RENAME & REMOVE

CREATE .gitignore EARLY

- Ignore generated files and local secrets.
- Example .gitignore content:

```
.env
.venv/
__pycache__/
node_modules/
*.log
!important.log
```

→ The last pattern keeps important.log despite *.log.

ALREADY TRACKED?

- Adding a tracked file to .gitignore does not untrack it.
- For a previously tracked .env:

```
git rm --cached .env
```

- Then add .env to .gitignore and commit the change.
- --cached removes it from tracking, not from disk.
- Old commits still contain the previous file.
- If a real secret was exposed, replace that secret.

RENAME A TRACKED FILE

```
git mv old.txt new.txt
```

- Stages the rename.
- Git can also detect renames made in your editor.

REMOVE A TRACKED FILE

```
git rm old.txt
```

- Removes it from disk and stages the deletion.
- Use only when you intend to delete the file.

EMPTY FOLDERS

- Git tracks files, not empty folders.
- A placeholder such as logs/.gitkeep keeps a folder.
- .gitkeep is a naming convention, not a Git feature.

CHECK

```
git status
git diff --staged
```

→ Review exactly what you will commit.

08 / 20 — BRANCHES: SEPARATE YOUR WORK

WHAT IS A BRANCH?

- A branch is a movable name pointing to a commit.
- Use one branch for a focused feature or fix.
- Committing moves the current branch forward.

CREATE + SWITCH

`git switch -c feature/readme` → Creates a branch and switches to it.

`git switch main` → Returns to an existing branch.

`git branch feature/readme` → Creates the branch without switching.

- Use this only if that branch does not already exist.

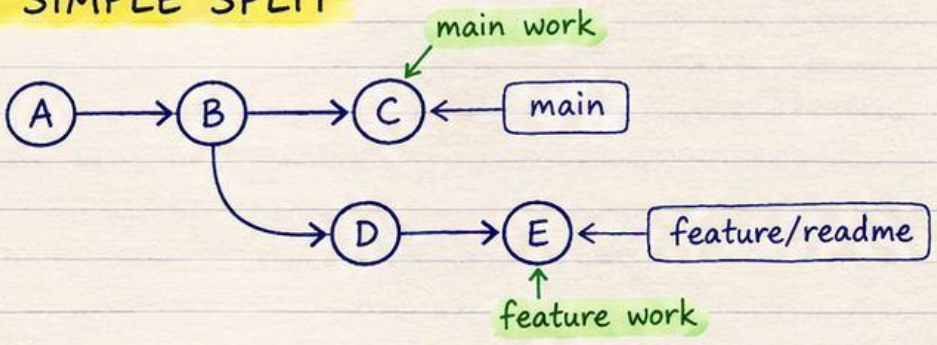
SEE YOUR BRANCHES

`git branch` → Lists local branches; * marks the current one.

`git branch --show-current` → Prints the current branch name.

`git branch -a` → Includes remote-tracking branches.

A SIMPLE SPLIT



- HEAD follows the branch you are using.

- A → B → C on main, and B → D → E on feature/readme.
- C: main work. E: feature work.

BEFORE SWITCHING

- Check git status.
- Commit or stash unfinished work when needed.
- Git blocks a switch if it would overwrite edits.
- Some uncommitted edits can follow you across branches.

OLDER COMMAND

`git checkout -b feature/readme`

- is the older create-and-switch form.
- Prefer switch when learning branch navigation.

09/20 — MERGE BRANCHES

MERGE INTO THE CURRENT BRANCH

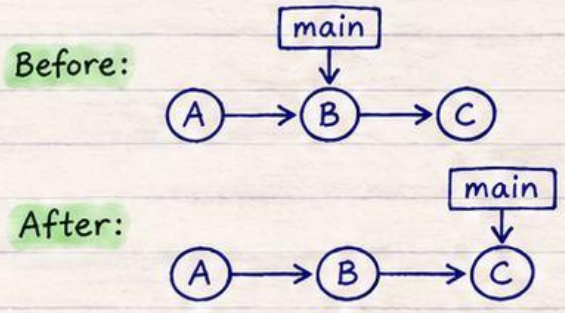
- Start with committed work and a clean working tree.

```
git switch main
git merge feature/readme
```

- This brings feature/readme into main.
- Always check which branch is receiving the merge.

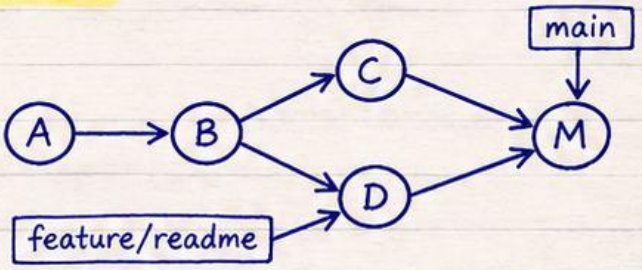
FAST-FORWARD

- If main is an ancestor of the feature branch, Git can move main forward without a merge commit.
- No extra merge commit is needed.



WHEN BOTH SIDES HAVE NEW COMMITS

- A normal merge combines both lines of history.
- M is a merge commit with parents C and D.
- If Git cannot combine the edits, resolve conflicts.



REVIEW THE RESULT

```
git status
git log --oneline --graph --all
```

- Run your project's checks before sharing.

CLEAN UP AFTER MERGING

```
git branch -d feature/readme
```

- Deletes the local branch name after it is merged.
- Run this from a different branch.
- Merged commits stay reachable through main.
- Deleting a branch is not the same as deleting files.

REMOTE CLEANUP

```
git push origin --delete feature/readme
```

- Only remove a shared branch when the team is done.

RESOLVE A MERGE CONFLICT

WHY IT HAPPENS

- Git cannot always combine competing edits.
- Conflicts can involve text, deleted files or renames.
- A conflict needs a decision, not a random choice.

EXAMPLE: MERGING feature/readme INTO main

```
<<<<<<< HEAD
Hello
=======
Salam, Zain!
>>>>>>> feature/readme
```

← The top block is the current branch in this merge.

← The bottom block comes from feature/readme.

FIX THE FILE

1. Read both changes.
2. Write the correct final content.
3. Remove every conflict marker.
4. Save and test the file.

Possible final content:

```
Salam, Zain!
```

FINISH THE MERGE

```
git status
```

```
git add README.md
```

```
git diff --staged
```

```
git commit -m "Resolve greeting conflict"
```

- Stage every resolved file, not just one.

CANCEL AN IN-PROGRESS MERGE

```
git merge --abort
```

- Best used when you started with a clean working tree.

REMEMBER

- Accept Both is not always the correct solution.
- For a rebase conflict, use `git rebase --continue` after resolving and staging, not a merge commit.
- Current/incoming labels can differ during rebase.

STASH: PAUSE UNFINISHED WORK

WHEN TO USE IT

- You need to change tasks before your edits are ready.
- A stash temporarily stores uncommitted changes.
- It is local; it is not a pushed backup.

SAVE YOUR WORK

`git stash push -m "WIP README"`

- Saves tracked changes and cleans them from your work.

`git stash push -u -m "WIP with new files"`

- Also includes untracked files.
- Ignored files are still excluded with -u.

LOOK BEFORE RESTORING

`git stash list`

`git stash show -p "stash@{0}"`

- The newest stash is stash@{0}.
- Quote stash references for shell compatibility.

APPLY VS POP

`git stash apply`

- Restores the newest stash and keeps its entry.

`git stash pop`

- Restores it and removes the entry if successful.
- If pop conflicts, the stash is kept.

A SHORT WORKFLOW

Save work (stash) → switch task → return → apply stash

- Commit or review restored work before discarding a stash.

REMOVE ONE YOU NO LONGER NEED

`git stash drop "stash@{0}"`

- Only after confirming the work is safely kept elsewhere.
- Stashes are temporary storage, not project milestones.

CHECK AFTER RESTORING

`git status`

- Resolve any conflicts before continuing.

12 / 20 — UNDO: CHOOSE THE RIGHT TOOL

UNSTAGE, KEEP YOUR EDITS

```
git restore --staged README.md
```

- Moves the staged version back to HEAD.
- Your working-file edits remain.
- Use in a repo that already has a commit.

DISCARD UNSTAGED EDITS

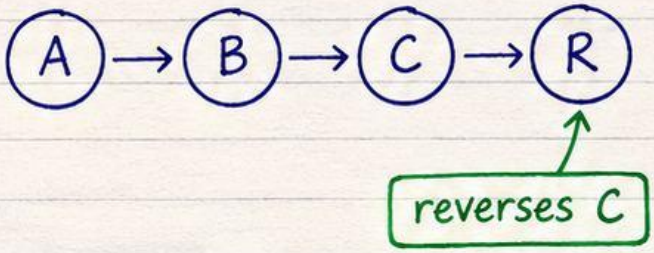
```
git restore README.md
```

- Replaces the working file with its staged version.
- Unstaged edits to that file are lost.
- It does not remove new untracked files.

UNDO A SHARED COMMIT

```
git revert HEAD
```

- Creates a new commit reversing the latest commit.
- The original commit stays in history.
- Best for undoing a normal commit already shared.



REWORK YOUR LAST LOCAL COMMIT

```
git reset --soft HEAD~1
```

- Moves the branch back one commit.
- Keeps file contents and the staging area.
- Use for unshared work with a parent commit.

DANGEROUS RESET

```
git reset --hard HEAD~1
```

- Moves the branch back and resets tracked files.
- Uncommitted tracked work can be lost.
- Do not use as a routine fix on shared work.

RECOVERY CLUE

```
git reflog
```

- Shows recent local movements of HEAD.
- It may help find a moved-away commit.
- It cannot reliably recover edits never saved by Git.

REBASE: REPLAY YOUR COMMITS

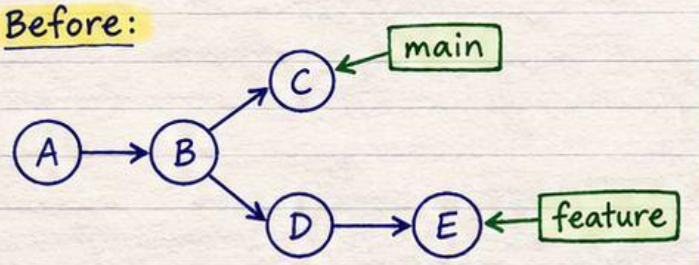
WHAT REBASE DOES

- Rebase replays branch commits on a new base.
- Replayed commits get new IDs.
- Use it mainly for your own unshared branch.

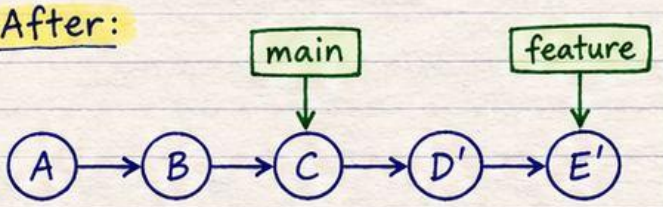
BEFORE AND AFTER

- Before: main has A → B → C.
- Feature has A → B → D → E.
- After: feature has A → B → C → D′ → E′.
- main still points to C.

Before:



After:



- D′ and E′ replace D and E in this branch history.

UPDATE A FEATURE BRANCH

- Start with a clean working tree.

```
git fetch origin
git switch feature/readme
git rebase origin/main
```

IF A CONFLICT APPEARS

- Edit the file and remove the conflict markers.

```
git add README.md
git rebase --continue
```

- Repeat until the rebase finishes.
- To cancel:

```
git rebase --abort
```

MERGE VS REBASE

- Merge joins histories and can add a merge commit.
- Rebase rewrites the replayed commits.
- A fast-forward merge needs no merge commit.

TEAM RULE

- Do not rewrite commits others are using.
- If rewriting your already-pushed private branch is agreed:

```
git push --force-with-lease
```

- Use only with the intended upstream configured.
- The lease is a check, not permission to overwrite work.

A NAME FOR A CHECKPOINT

- Tags mark a particular commit, often a release.
- A branch moves with new commits.
- A tag does not move automatically.

CREATE AN ANNOTATED TAG

```
git tag -a v1.0.0 -m "First release"
```

- Use after checking the intended commit.
- An annotated tag stores tagger details and a message.

LIGHTWEIGHT ALTERNATIVE

```
git tag demo
```

- A simple tag pointing to the current commit.
- Use different tag names; do not create the same tag twice.

LIST + INSPECT

```
git tag
git tag -l "v1.*"
git show v1.0.0
```

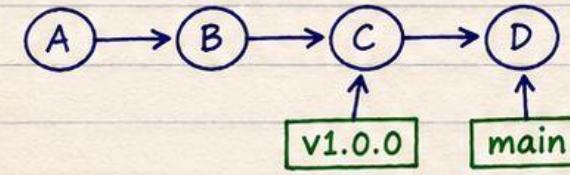
→ Inspect the tagged commit before publishing it.

SHARE ONE TAG

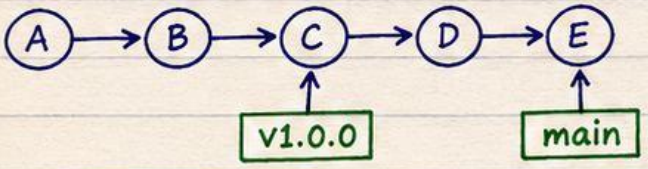
```
git push origin v1.0.0
```

- A normal branch push does not normally push all tags.
- Push the specific release tag you intend to share.

PICTURE IT



- After another commit, main moves; the tag stays at C.

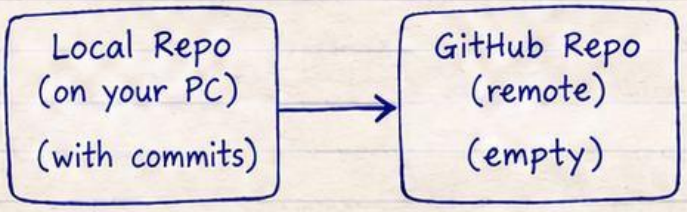


GOOD HABIT

- Tag a tested version.
- Avoid moving a published release tag.
- A GitHub release can add release notes and downloads around a tag; it is more than the Git tag itself.

CREATE THE REMOTE

- Create a GitHub repository and choose its visibility.
  - Public: anyone can view it.
  - Private: access is restricted.
- For an existing local repo, start with an empty remote to avoid a separate initial history.



CONNECT YOUR LOCAL REPO

```
git remote add origin REPO_URL
```

- Replace REPO_URL with the URL copied from GitHub.

```
git remote -v
```

- origin is a conventional nickname, not a special server.

Output:

```
origin  https://github.com/USER/REPO.git (fetch)
origin  https://github.com/USER/REPO.git (push)
```

FIRST PUSH

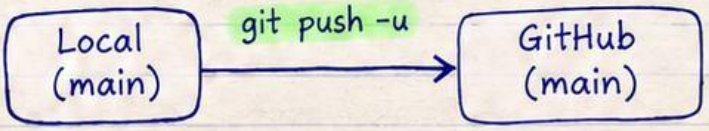
- Make at least one local commit first.

```
git branch --show-current
```

- If your branch is main:

```
git push -u origin main
```

- -u records the upstream for future push/pull commands.
- Later, use git push from that branch.



Result:

- Pushes your committed changes to GitHub.
- Sets the upstream so later you can just use git push from main.

AUTHENTICATION

- HTTPS: use a credential manager or personal access token.
- Your GitHub account password is not used for Git HTTPS.
- SSH: register a public key; keep its private key private.
- Do not put a token inside a shared command or URL.

WRONG REMOTE URL?

```
git remote set-url origin REPO_URL
```

- Use this when origin already exists.
- Adding origin twice produces an error.

REMEMBER

- A commit stays local until you push it.
- Pushing requires permission to write to that repository.

16 / 20 — CLONE, FETCH, PULL & PUSH

CLONE AN EXISTING REPOSITORY

```
git clone REPO_URL zain-project
```

```
cd zain-project
```

- Replace REPO_URL with the actual repository URL.
- A normal clone includes files, history and an origin remote.
- `git remote -v`
- `git branch -a`

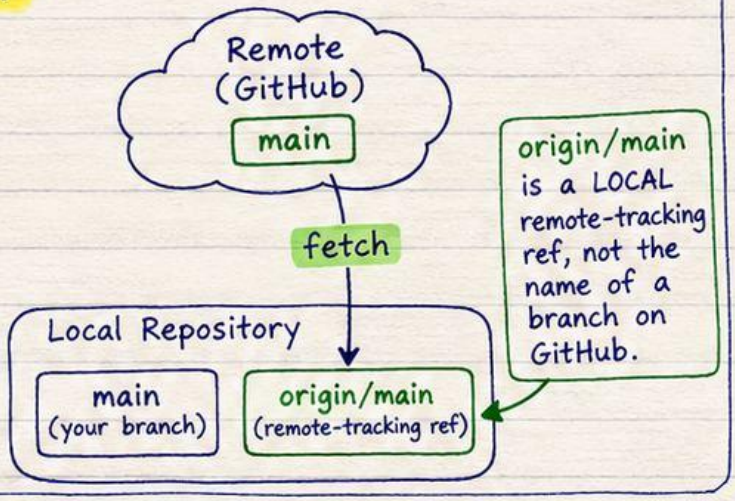
FETCH: DOWNLOAD, THEN REVIEW

```
git fetch origin
```

- Downloads commits and updates remote-tracking refs.
- It does not merge changes into your working branch.

```
git log --oneline HEAD..origin/main
```

- Shows commits on origin/main that HEAD does not contain.



PULL: FETCH + INTEGRATE

- On your local main with clean work:

```
git switch main
```

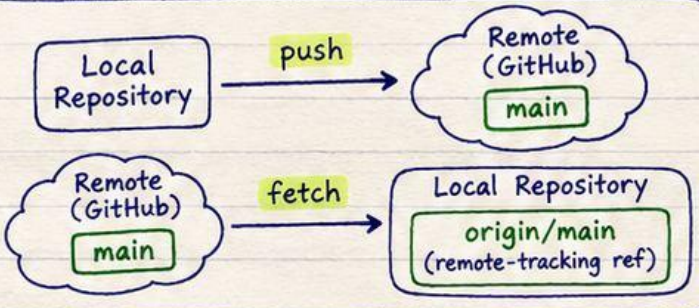
```
git pull --ff-only origin main
```

- Fast-forwards only; stops if histories have diverged.
- Other pull modes can merge or rebase.

PUSH: SHARE LOCAL COMMITS

```
git push
```

- With the intended upstream configured, uploads commits.



IF PUSH IS REJECTED

- Fetch and inspect the incoming changes.
- Merge or rebase as your workflow requires, then test.
- Do not force-push just to silence an error.

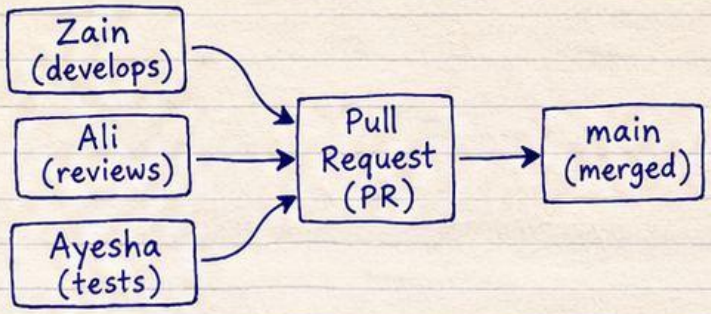
REMEMBER

- origin/main is your local view of the remote branch.
- Fetch refreshes that view; it is not a live connection.

17 / 20 — TEAMWORK, FORKS & PULL REQUESTS

A SMALL TEAM FLOW

- Zain develops; Ali reviews; Ayesha tests.
- Keep each change focused on a feature branch.



START FROM UPDATED main

```
git switch main
git pull --ff-only origin main
git switch -c feature/readme
```

- Edit the README, then:

```
git add README.md
git commit --m "Improve setup steps"
git push -u origin feature/readme
```

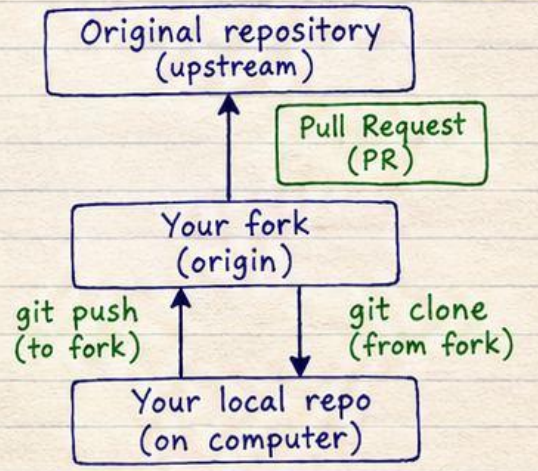
OPEN A PULL REQUEST

- On GitHub, propose feature/readme into main.
- Explain what changed and how you tested it.
- Review the diff; respond to feedback.
- Merge after the required checks and approval.

A pull request is a review proposal, not git pull.

NO WRITE ACCESS? USE A FORK

- Fork: your own repository copy on GitHub.
- Clone: a repository copy on your computer.
- Fork the original, then clone your fork.
- Push your branch to your fork and open a PR to the original repository.



KEEP THE ORIGINAL AS upstream

```
git remote add upstream ORIGINAL_URL
git fetch upstream
```

- Replace ORIGINAL_URL with the original repo's URL.
- origin usually points to your fork in this setup.

AFTER MERGE

- Update local main and delete the finished local branch.
- Follow the repository's contribution rules.
- Do not push directly to protected branches.

GIT IN VISUAL TOOLS

SAME IDEAS, DIFFERENT CONTROLS

- Visual tools use commits, branches and remotes too.
- Review what an action will change before clicking.

VS CODE

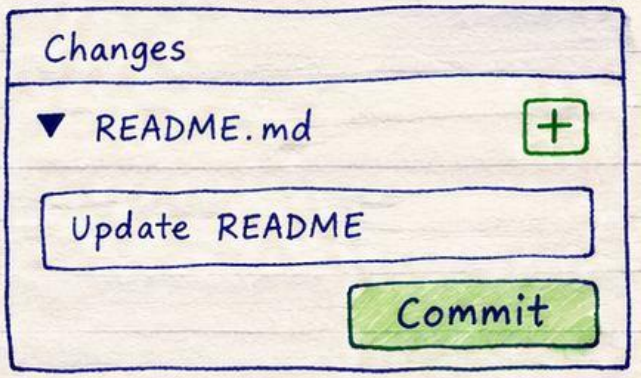
- Open your repository folder, then Source Control.
- Select a changed file to inspect its diff.
- Use + to stage a file and − to unstage it.
- Write a message and commit the staged changes.
- Use the branch control to create or switch branches.
- Sync pulls and then pushes; it is more than saving.

GITHUB DESKTOP

- Clone or add a local repository.
- Review the changed files and selected changes.
- Write a summary and commit to the current branch.
- Fetch origin checks remote work.
- Pull origin integrates incoming changes.
- Push origin publishes your commits.

BUTTON → GIT IDEA

Stage	→	git add
Commit	→	git commit
Fetch	→	git fetch
Push	→	git push
History	→	git log



CONFLICTS IN AN EDITOR

- Read both versions and choose the correct final result.
- Accept Current, Incoming or Both only after review.
- Save, stage and finish the merge or rebase.

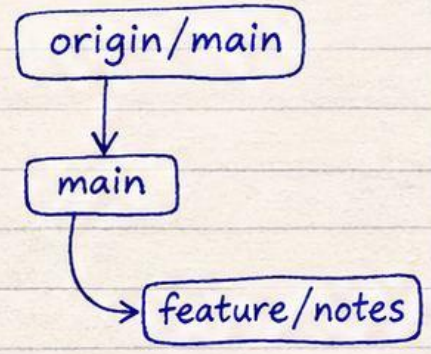
REMEMBER

- Check the current branch and selected files.
- Committing in a visual tool is still a local action.

19 / 20 — A PRACTICAL DAILY WORKFLOW

BEFORE YOU START

- `git status`
- If this repo has a remote and main is your base:
  - `git switch main`
  - `git pull --ff-only origin main`
  - `git switch -c feature/notes`
- Start from committed or safely stashed work.



MAKE ONE USEFUL CHANGE

- Create notes.txt in your editor:

  `Today Zain learned Git.`

- Save the file.

- `git add notes.txt`
- `git diff --staged`
- `git commit -m "Add first learning note"`

MAKE A SECOND CHANGE

- Add another line in your editor:

  `Review before every commit.`

- `git diff`
- `git add notes.txt`
- `git commit -m "Add review reminder"`
- `git log --oneline -2`

SHARE WHEN READY

`git push -u origin feature/notes`

- Open a pull request if working with a team.
- For a local-only practice repo, skip the remote steps.

IF USING AI TO EDIT CODE

- Start from a known working commit.
- Review the diff and run the project checks.
- Commit only changes you understand.
- The review habit is the same for human-written code.

DONE CHECKLIST

- ☑ Correct branch?
- ☑ Changes reviewed?
- ☑ Checks passed?
- ☑ Clear message?
- ☑ Intended commits pushed?

20 / 20 — QUICK REFERENCE & COMMON FIXES

THE DAILY LOOP

```
git status
git diff
git add README.md
git diff --staged
git commit -m "Describe the change"
git push
```

→ Push assumes the intended upstream is configured.

FIND THE RIGHT TOOL

New repo	→ git init
Existing repo	→ git clone REPO_URL
New branch	→ git switch -c feature/name
Inspect history	→ git log --oneline
Fetch remote work	→ git fetch origin
Unstage a file	→ git restore --staged README.md
Undo a shared normal commit	→ git revert HEAD
Pause work	→ git stash push -u

COMMON MESSAGES

"not a git repository"	→ Open the correct project folder first.
"nothing to commit"	→ Save your files; check status and staging.
"remote origin already exists"	→ Inspect git remote -v; update the URL if needed.
"non-fast-forward"	→ Fetch, inspect and integrate; do not blindly force-push.
"Authentication failed"	→ Check the URL, credentials and repository access.

FIVE RULES TO KEEP

1. One clear change per commit.
2. Review staged changes before committing.
3. Keep secrets out of history.
4. Use feature branches for focused work.
5. Know what an undo command will discard.

KEEP THIS FLOW

Edit → Review → Stage → Commit → Share

Reference: git-scm.com/docs

ZainSpire • Learn it. Practise it. Use it.