**Advanced Git Tutorial**

*(Assumes you know the basics: commit, branch, merge, push, pull)*

**1. Advanced Branching Strategies**

**Git Flow (common in big teams)**

* main → stable production code
* develop → integration branch
* feature/\* → new features
* release/\* → pre-release versions
* hotfix/\* → urgent production fixes

Example start:

bash

CopyEdit

git checkout -b feature/payment-gateway develop

**2. Interactive Rebase**

Reorder, squash, or edit commits before pushing — keeps history clean.

bash

CopyEdit

git rebase -i HEAD~5

Then in the editor:

* pick → keep commit
* squash → merge with previous
* edit → change commit message
* drop → remove commit

**3. Bisect to Find Bugs**

Binary search through commits to find the one that broke the code:

bash

CopyEdit

git bisect start

git bisect bad HEAD

git bisect good v1.0.0

Git will check out commits for you — test each, mark as good or bad:

bash

CopyEdit

git bisect good

git bisect bad

When done:

bash

CopyEdit

git bisect reset

**4. Cherry-Picking**

Apply a single commit from another branch without merging the whole branch:

bash

CopyEdit

git cherry-pick abc123

**5. Reset Variants**

* **Soft reset** (keep changes staged):

bash

CopyEdit

git reset --soft HEAD~1

* **Mixed reset** (keep changes unstaged):

bash

CopyEdit

git reset --mixed HEAD~1

* **Hard reset** (discard changes completely):

bash

CopyEdit

git reset --hard HEAD~1

**6. Reflog — Your Time Machine**

If you mess up a reset or rebase:

bash

CopyEdit

git reflog

git checkout abc123

Reflog stores **all** movements of HEAD, even deleted commits.

**7. Submodules (Nested Repos)**

To include another repo inside yours:

bash

CopyEdit

git submodule add https://github.com/user/lib.git libs/lib

git submodule update --init --recursive

**8. Sparse Checkout (Partial Clone)**

Only get certain folders from a huge repo:

bash

CopyEdit

git sparse-checkout init --cone

git sparse-checkout set folder/subfolder

**9. Worktrees (Multiple Checkouts)**

Work on multiple branches without switching:

bash

CopyEdit

git worktree add ../feature-branch feature-branch

**10. Signed Commits**

Prove commits came from you:

bash

CopyEdit

git commit -S -m "Signed commit"

*(requires GPG key setup)*

**11. Advanced Diff Tricks**

Compare current branch to main:

bash

CopyEdit

git diff main..HEAD

Compare staged changes:

bash

CopyEdit

git diff --cached

**12. Hooks — Automate Git Actions**

Hooks are scripts that run on Git events (like commit, push).  
Example: .git/hooks/pre-commit:

bash

CopyEdit

#!/bin/sh

npm run lint

Make it executable:

bash

CopyEdit

chmod +x .git/hooks/pre-commit

**13. Advanced Pull & Push**

* Avoid messy merge commits:

bash

CopyEdit

git pull --rebase

* Push only the current branch:

bash

CopyEdit

git push origin HEAD

**14. Clean Up Old Branches**

Delete local merged branches:

bash

CopyEdit

git branch --merged main | grep -v "main" | xargs git branch -d

**15. Advanced Tagging**

Annotated tag:

bash

CopyEdit

git tag -a v2.0.0 -m "Release version 2.0.0"

git push origin v2.0.0

``]

---

💡 \*\*Pro Git Habits\*\*

- Always work on a feature branch, never directly in `main`.

- Rebase before merging to keep history clean.

- Use `git reflog` as your safety net.

- Automate checks with Git hooks before pushing.

---

If you want, I can make you an \*\*“Advanced Git Disaster Recovery Guide”\*\* — basically a cheat sheet for fixing every nightmare situation: wrong branch commits, force-push mistakes, lost commits, bad rebases, and more.

That’s where most “advanced” Git skills really get tested.