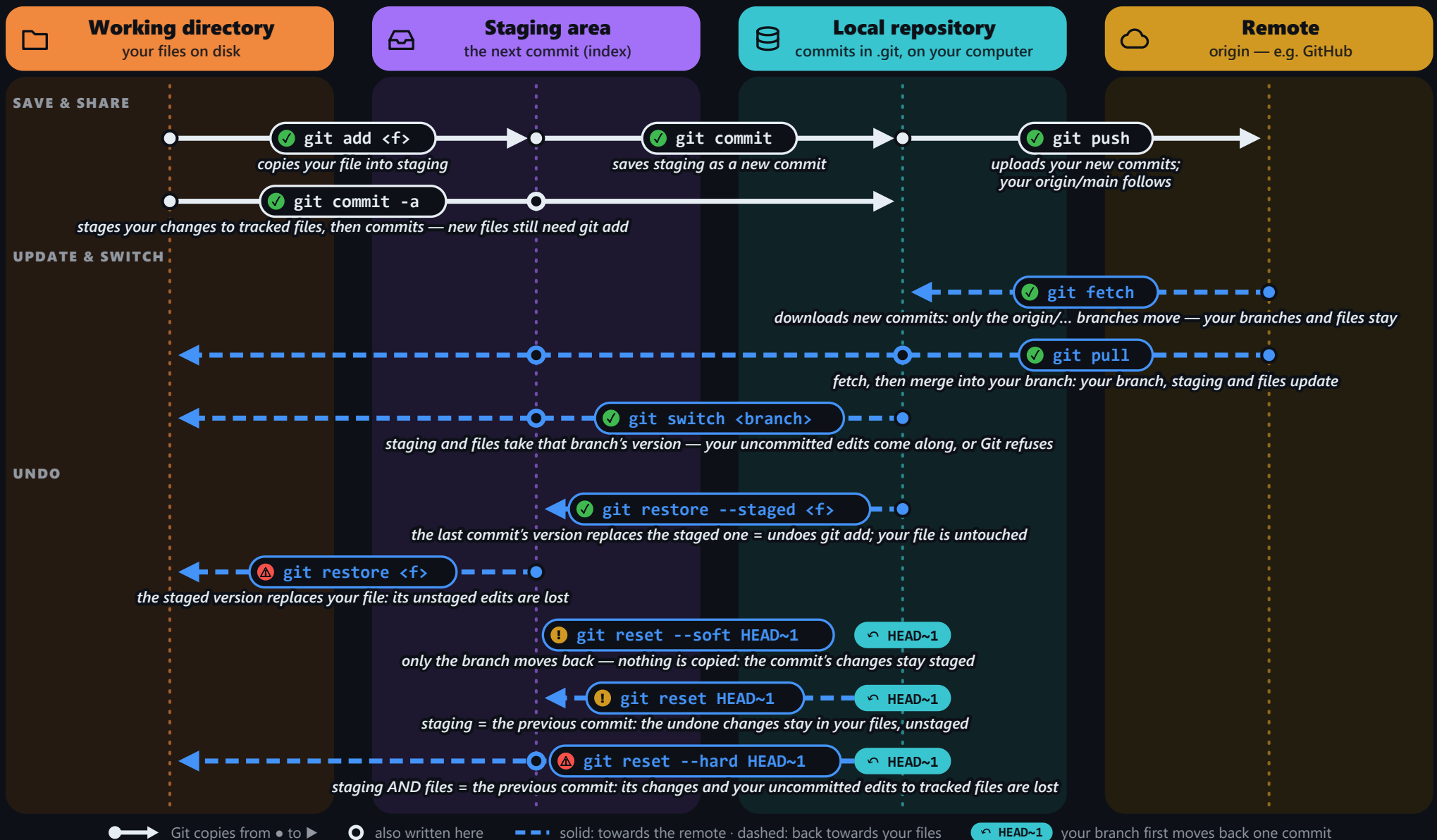


# 01 Git at a glance

The four places Git keeps your work — and exactly what each command copies between them

✓ Safe    ! Rewrites history    ⚠ Can lose work



- o **<f>** a file path: type `app.js`, without the `< >`
- o **HEAD~1** the commit before the one you are on
- o **git pull** if you and the remote both have new commits, Git stops until you choose: `git config pull.rebase false` (merge) or `true` (rebase)

# 02 History at a glance

Before → after: what each command does to your commits and branches

Before → after. Circles are commits, the yellow ring is where you are (HEAD).

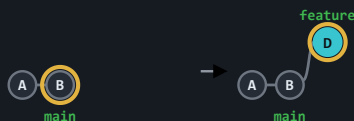
● new commit    ○ no branch points to it any more

## ✓ git commit



A new commit from what is staged; your branch moves onto it.

## ✓ git switch -c feature + commit



A new branch starts where you are; your next commits move it, main stays.

## ✓ git merge feature (fast-forward)



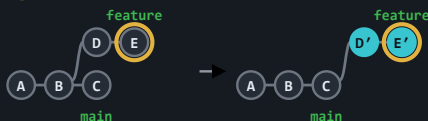
On main, when main has no new commits: main just moves up to E — no merge commit.

## ✓ git merge feature



On main: joins feature with a merge commit that has two parents.

## ! git rebase main



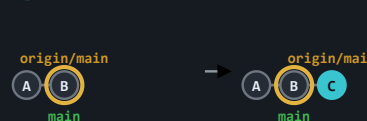
On feature: replays D and E on top of main as copies with new hashes (D' E'); the originals leave the branch.

## ✓ git cherry-pick D



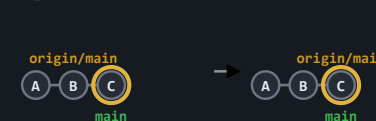
Copies one commit onto your branch (D', new hash); the original stays.

## ✓ git fetch



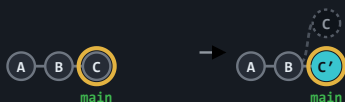
Downloads new commits: only the origin/... labels move. Your branch and files stay.

## ✓ git push



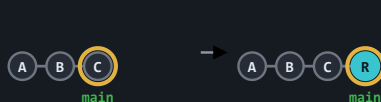
Uploads C: main on the remote — and your origin/main — move to C. Your files do not change.

## ! git commit --amend



Replaces the last commit with a new one (C', new hash); C leaves the branch. Do it before pushing.

## ✓ git revert C



Adds a commit R that undoes C. History is kept: safe on shared branches.

## ! git reset --soft HEAD~1



Moves your branch back to B; C's changes are still staged — commit again to redo it.

## ⚠ git reset --hard HEAD~1



Moves your branch back; staging and files become B: C's changes are gone (git reflog still finds C).

- o A commit is never edited: a command that "changes" one makes a new commit with a new hash.
- o A commit no branch points to any more is still found by **git reflog** (30 days by default).

## START

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

Name recorded in your commits (once per computer)

**git config --global user.email "you@example.com"**

E-mail recorded in your commits

**git init**

Turn the current folder into a repository

**git clone <url>**

Download a repository with its whole history

## LOOK AROUND

**git status**

What changed, what is staged, which branch you are on

**git diff**

Changes not staged yet (working dir vs staging)

**git diff --staged**

What the next commit will contain

**git log --oneline --graph --all**

History of every branch, one line per commit

**git show <commit>**

One commit: message and changes

## SAVE YOUR WORK

**git add <file>**

Stage a file for the next commit

**git add -p**

Stage chosen parts (hunks) only

**git commit -m "message"**

Record what is staged as a commit

**git commit --amend**

 Redo the last commit (message or content)

**git restore <file>**

 Throw away unstaged edits of a file

**git restore --staged <file>**

Unstage (your edit stays on disk)

## SHARE

**git fetch**

Download new commits; your branches do not move

**git pull**

Fetch, then merge into the current branch

**git pull --rebase**

 Fetch, then replay your commits on top (new hashes)

**git push**

Upload your commits to the branch's upstream

**git push -u origin <branch>**

First push of a new branch (sets upstream)

**git remote -v**

Which remotes this repository talks to

- o **HEAD** the commit you are on (usually the tip of your current branch)
- o **origin** the default name of the remote you cloned from
- o **<file>** a file path: type **app.js**, without the **< >**

## CREATE & SWITCH

- **git branch**  
List local branches (\* = current)
- **git branch -a**  
Also list remote-tracking branches
- **git switch <branch>**  
Go to a branch (files follow)
- **git switch -c <branch>**  
Create a branch here and switch to it
- **git switch -**  
Back to the previous branch
- **git branch -m <new-name>**  
Rename the current branch

## COMBINE

- **git merge <branch>**  
Bring a branch into the current one
- **git merge --no-ff <branch>**  
Always record a merge commit
- **git rebase <branch>**  
! Replay your commits on top of <branch> (new hashes)
- **git rebase -i HEAD~3**  
! Edit the last 3 commits: reorder, squash, reword, drop
- **git cherry-pick <commit>**  
Copy one commit onto the current branch

## CONFLICTS

- **git status**  
Lists the files in conflict (e.g. "both modified")
- *Edit the file*  
Keep what you want, delete <<<<<<<  
===== >>>>>>> lines
- **git add <file>**  
Mark the file as resolved
- **git merge --continue**  
Finish (or: git rebase --continue / cherry-pick --continue)
- **git merge --abort**  
Give up and go back to before (rebase --abort, ...)

## CLEAN UP

- **git branch -d <branch>**  
Delete a merged branch (refuses if unmerged)
- **git branch -D <branch>**  
⚠ Delete even if unmerged
- **git push origin --delete <branch>**  
⚠ Delete the branch on the remote (unmerged commits on it go too)
- **git fetch --prune**  
Also delete origin/... copies of branches deleted on the remote

## TAGS

- **git tag**  
List tags
- **git tag -a v1.0 -m "Release 1.0"**  
Annotated tag on the current commit
- **git push origin v1.0**  
Tags are not pushed automatically
- **git push --follow-tags**  
Push commits plus their annotated tags

## READ HISTORY

- **git log --oneline**  
Compact history of the current branch
- **git log -p <file>**  
Every change made to one file
- **git log main..feature**  
Commits on feature that main does not have
- **git blame <file>**  
Who last changed each line

| I WANT TO...                                                                                                                                  | RUN                                                             | GOOD TO KNOW                                                                                                                                                                              |
|-----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  Throw away my edits to a file                                | <code>git restore &lt;file&gt;</code>                           |  Unstaged edits are gone for good                                                                       |
|  Unstage a file (keep my edit)                                | <code>git restore --staged &lt;file&gt;</code>                  |  Only the staging area changes                                                                          |
|  Fix the last commit's message or add a forgotten file        | <code>git add &lt;file&gt; &amp;&amp; git commit --amend</code> |  New hash — do it before pushing                                                                        |
|  Undo the last commit, keep changes staged                    | <code>git reset --soft HEAD~1</code>                            |  Your work stays staged, ready to recommit — not after pushing                                          |
|  Undo the last commit, keep changes in files                  | <code>git reset HEAD~1</code>                                   |  Default --mixed: unstaged but on disk — not after pushing                                              |
|  Undo a commit that is already pushed                         | <code>git revert &lt;commit&gt;</code>                          |  Adds a new commit that cancels it — safe for shared branches                                           |
|  Put my changes aside for now                                 | <code>git stash push -m "wip"</code>                            |  Back with <code>git stash pop</code> — new, untracked files need <code>-u</code>                       |
|  Throw away the last commit and its changes                 | <code>git reset --hard HEAD~1</code>                            |  Its changes and your uncommitted edits to tracked files are lost (git reflog still finds the commit) |
|  Find a “lost” commit (after reset, rebase, deleted branch) | <code>git reflog</code>                                         |  Every place HEAD has been — then <code>git branch rescue &lt;hash&gt;</code>                         |
|  Stop a merge / rebase / cherry-pick in progress            | <code>git merge --abort</code>                                  |  Or <code>rebase --abort</code> , <code>cherry-pick --abort</code>                                    |
|  Replace the remote branch after rewriting history          | <code>git push --force-with-lease</code>                        |  Refuses if the remote branch changed since your last fetch                                           |

## STASH

### `git stash`

Put uncommitted changes aside

### `git stash -u`

Include untracked (new) files

### `git stash list`

All stashes, newest = `stash@{0}`

### `git stash pop`

Re-apply the newest and drop it (kept if it conflicts)

### `git stash apply stash@{1}`

Re-apply one, keep it in the list

## RESET — THREE STRENGTHS

### `git reset --soft <commit>`

 Move the branch; staging and files untouched

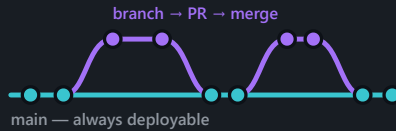
### `git reset <commit>`

 Move the branch and reset staging; files untouched

### `git reset --hard <commit>`

 Move the branch, staging AND files

## GITHUB FLOW



```
git switch main &&
git pull
```

Start from an up-to-date main

```
git switch -c
<feature>
```

One short-lived branch per change

```
git commit -am "... " &&
git push -u origin
<feature>
```

Commit, push, open a pull request

*Merge the pull request on GitHub*

After review; then delete the branch

```
git switch main &&
git pull --prune
```

Back to main; also delete origin/... copies of deleted branches

## GIT FLOW



```
git checkout -b
feature/x develop
```

Features start from develop

```
git checkout develop
&& git merge --no-ff
feature/x
```

Finish a feature into develop

```
git checkout -b
release-1.1 develop
```

Freeze a release

```
git checkout main &&
git merge --no-ff
release-1.1
```

Release to main...

```
git tag -a 1.1 -m
"Release 1.1"
```

...tag it, and merge it back into develop

## TRUNK-BASED



```
git commit -am "small
change"
```

Small, complete commits on main

```
git pull --rebase
```

! Replay yours on top of the others' (new hashes)

```
git add <file> && git
rebase --continue
```

After fixing a conflict: mark it resolved, go on

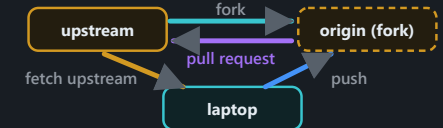
```
git push
```

Integrate at least once a day

*Ship unfinished work switched off*

Behind a feature flag, e.g. NEW\_CHECKOUT = false

## FORKING



```
git remote add
upstream
```

```
<original-url>
```

Once: the original as a second remote

```
git fetch upstream &&
git merge
upstream/main
```

Sync your main with the original

```
git push
```

Update your fork (origin)

```
git switch -c <fix> ...
git push -u origin
<fix>
```

Work on a branch in your fork

*Open a pull request to the original*

Maintainers review and merge