



A SIMPLE GUIDE

Git, *explained.*

Four branches and one loop you run all day. That's the whole thing to start – the rest can wait until you need it.

EVERYONE

main

What users see

The live one. Prod. Everything here is in front of real people, so nothing lands on it except by merge, reviewed, after it already worked somewhere else.

Production

Protected

Merge only

staging

The dress rehearsal

A copy of prod with realistic data, where the release gets tried before anyone outside sees it. If it breaks here, nobody gets paged.

Pre-release

Safe to break

develop

Where work collects

Finished features collect here and get tested together. Plenty of small projects skip it and merge straight into main – that's fine.

Integration

Optional

feature/

Yours, for one job

One branch per thing you're working on, named after it: **feature/copy-link**. Cut from the latest main, keep it short, merge back by pull request.

One per task

Short-lived

Merged by PR

JUST YOU

RULE OF
THUMB

Branch, commit, push, PR, review, merge. Never commit straight to *main*.

What a day *actually* looks like.

Nine steps. You'll do this loop so many times it stops being a list and becomes a reflex.

<code>git switch main</code>	Get back to the branch everything is cut from.
<code>git pull</code>	Take everyone else's work before you start, not after.
<code>git switch -c feature/x</code>	New branch, named after the job, switched to in one go.
<i>do the work</i>	Edit, run it, check it. Commit whenever a piece works, not once at the end.
<code>git add .</code>	Stage what you changed. <code>git add -p</code> to pick line by line.
<code>git commit -m "..."</code>	Save it with a message that says <i>why</i> .
<code>git push</code>	Send the branch up. First push on a new branch prints the PR link.
<i>open a PR</i>	Describe what changed and why. Someone reads it. You fix what they find.
<i>merge, delete</i>	Merge it, delete the branch. Finished branches are clutter.

Write the message for the person who git-blames it

That person is you, in eight months, trying to work out why this line exists. The diff already says what changed – the message is the only place the reason can live.

~~"fix"~~

→ "Fix total ignoring discounts on refunds"

~~"updates"~~

→ "Cache the feed for 60s; the API rate-limits at 100/min"

~~"wip"~~

→ "Add failing test for the empty-cart case"

Two things that never go in a commit

Secrets and generated files. API keys, `.env`, tokens, `node_modules`, build output. Put them in `.gitignore` on day one – a secret that reaches a public repo must be rotated, not deleted, because the history keeps it.

Undoing things, *by how far it got.*

Almost nothing in git is lost. What you run depends only on how far the mistake travelled.

Edited, not staged	<code>git restore <file></code> – throws your changes away. This one really does lose work.
Staged, not committed	<code>git restore --staged <file></code> – unstages it and keeps the edits.
Committed, not pushed	<code>git reset --soft HEAD~1</code> – undoes the commit, keeps the changes staged. Redo the message and commit again.
Committed to the wrong branch	Branch from where you are, then reset the branch you were on. Nothing is lost before a push.
Pushed already	<code>git revert <sha></code> – a new commit that undoes the old one. Don't rewrite history others have pulled.

A conflict is a question, not an error

You and somebody else changed the same lines, so git stops and asks which version wins. Open the file, delete the markers, keep the right result, then `git add` and commit. The longer your branch lives, the more of these you get – which is the real argument for short branches.

Pull requests people actually review

Keep them small	A 50-line PR gets read. A 2,000-line PR gets "looks good to me", which is nobody reviewing it at all.
Say why in the description	What changed, why, and what you want eyes on. Link the issue. Two sentences beats a title alone.
Review the way you want reviewing	Comment on the code, not the person. Say which notes are blocking and which are preferences.
Protect main	Require a PR and a passing build. The rule needs to survive the day someone is in a hurry.

SEND THIS
ON

Know someone editing files on the live site? Send them bennybytes.net/explainers/git-explained before they learn this the expensive way.