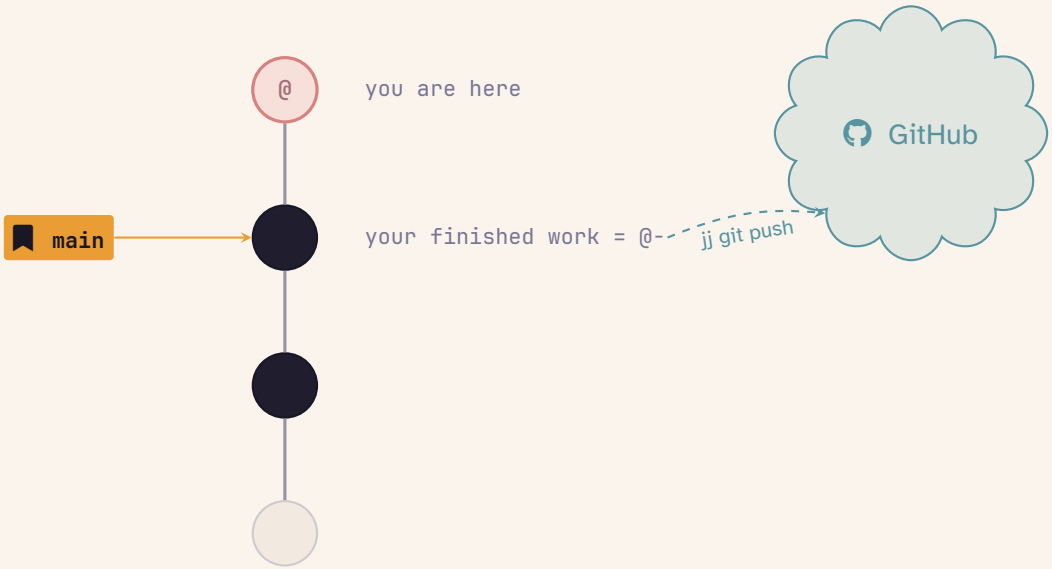
# THE jj FIELD GUIDE

Jujutsu version control, finally explained in plain language.
Save your work · understand bookmarks · push & sync with GitHub · fix mistakes




$: DAΞMON · NetrunnerVault · verified on jj 0.44.0 · rose pine dawn · 2026-08-28

## Read me first

This guide teaches **one** way to do each thing — the way that works, every time, in this vault. No aliases, no shortcuts, no "you could also…". Every dark block is a command you type; every block with the eye is the *actual output* jj 0.44 prints on this machine, so you can compare your screen against it.

If you only ever read one page of this document, make it the card below. This is the whole game.

### THE RITUAL — how a note gets to GitHub

```
the four moves, in order — details in sections 04, 05, 06

jj st                          # 1. look: what did I change?
jj commit -m "update: ..."     # 2. seal it with a message
jj bookmark set main -r @-     # 3. move the sticky note onto it
jj git push                    # 4. send it to GitHub
```

And **once per computer, ever** — not before every push:

```
one-time setup only — section 06 explains

jj bookmark track main@origin  # link your main to GitHub's main. ONCE.
```

> **you can drop a step you've been doing**
>
> You've been running `jj commit` both *before* and *after* the bookmark commands. Only the one **before** does anything. Moving a bookmark just repoints a label — it creates nothing that needs committing, so the second `jj commit` seals an empty change and gives you nothing. Commit once, move the note, push. Three moves, not five.

## Where everything is


01 **The big picture** — the three ideas that replace everything git taught you ..... p. 4

02 **The words** — @, @-, change, bookmark, `main@origin`, tracked ..... p. 7

03 **A worked example** — ShadowVault built from nothing, on real output ..... p. 11

04 **Save your work** — the everyday loop, and commit messages ..... p. 16

05 **Bookmarks, finally explained** — the part that's been driving you crazy ..... p. 20

06 **Push** — put it on GitHub, and what "track" actually means ..... p. 24

07 **Fetch & sync** — get what's on GitHub, safely ..... p. 28

08 **Side branches** — drafts that don't touch main ..... p. 31

09 **When things go wrong** — undo anything ..... p. 34

10 **The reference card** — every command, one place ..... p. 38

01

SECTION 01

# The big picture

Three ideas replace everything you knew from git. Get these three and every command in this guide becomes obvious instead of magic.

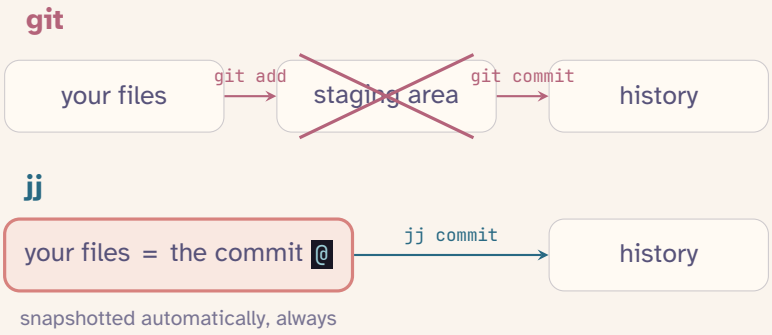
**IN THIS SECTION**


- Idea 1 — your folder *is already* a commit, called `@`
- Idea 2 — `@-` means “the one just below `@`”
- Idea 3 — bookmarks never move by themselves. Ever.
- The git → jj translation table

## Idea 1 — your folder is already a commit

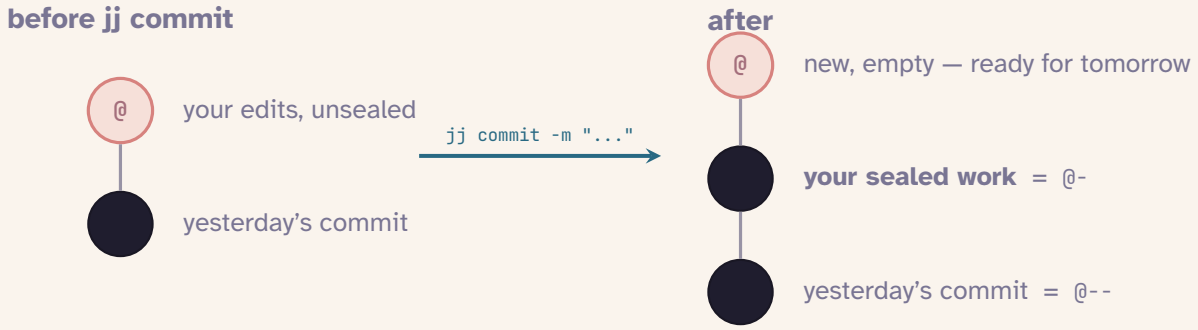
In git, your edits float around “unstaged” until you `git add` them, then sit “staged” until you `git commit` them. Forget a step and your work isn’t saved.

jj deletes that entire idea. **The working copy is itself a commit**, named `@` (say it: *at*). The moment you run *any* jj command, jj silently snapshots every edited file into `@` first. There is no staging area, no `git add`, and nothing to forget. Editing a note in Obsidian and running `jj st` — that’s it, your edits are already inside `@`.




## Idea 2 — @- is the commit right below you

History in jj is a stack. `@` sits on top; the commit directly underneath it is written `@-` (its parent). One more step down is `@--`. You will use `@-` constantly, because of what `jj commit` does: it **seals your current work and opens a fresh, empty `@` on top of it**. So one second after you commit, *the work you just finished lives at `@-`* — and that is exactly why the bookmark command in the ritual says `-r @-` and not `-r @`.




## Idea 3 — bookmarks never move by themselves

A **bookmark** is jj’s word for what git calls a branch, and it is nothing more than a **sticky note with a name on it, stuck to one single commit**. `main` is a sticky note. Nothing lives “inside” it, and you are never “on” it.

Here is the one difference that has caused every bit of your frustration. In git, committing secretly drags the current branch pointer forward for you. In jj:

> **Nothing moves a bookmark except you.**

You commit — the note stays where it was. You commit ten more times — it still points at the old commit. GitHub reads `main`’s sticky note, so until *you* move it with `jj bookmark set main -r @-`, GitHub

keeps seeing last week's vault, no matter how much you've committed. That's not you doing it wrong; that's the actual design, and section 05 shows exactly how to live with it.

## ⇄ The translation table

| git said | jj says | what changed |
|---|---|---|
| `git add` | — | gone. jj snapshots automatically |
| `git commit` | `jj commit -m "..."` | same idea, and opens a fresh @ on top |
| `branch` | `bookmark` | renamed, and it never auto-moves |
| `git checkout X` | `jj new X` | start a fresh change on top of X |
| `git push` | `jj bookmark set main -r @-`<br>`jj git push` | push sends *bookmark*s, so you move the note first |
| `git pull` | `jj git fetch` then look, then `jj rebase -o main` | split in two on purpose — section 07 |
| `git stash` | — | not needed; @ holds everything, jj new parks it |
| `git log` | `jj log` | the graph, with @ marked |
| `git reflog + reset` | `jj undo` | one word, undoes any operation |

> **why it works like this**
>
> jj splits "record my work" from "publish my work" so that nothing ever leaves your machine as a side effect. The price is one extra command (`jj bookmark set`). The payoff is that *everything* — commits, bookmark moves, fetches, even mistakes — is recorded in an operation log, and `jj undo` can reverse any of it. git cannot do that.

02

SECTION 02

# The words

Eight terms cover all of jj. Each one gets three lines and an example — skim them once now, then come back whenever a word in an error message looks unfamiliar.

**IN THIS SECTION**


- change · change ID · commit ID
- `@` and `@-`
- bookmark · `main` vs `main@origin` · tracked
- revision (`-r`) · operation

### change

One saved snapshot of the whole vault — what git calls a commit. Made with `jj commit`. The stack of changes is your history; see it with `jj log`.

### change ID vs commit ID

Every change has two names: a **change ID** in letters (`ptxpotvt`) that survives edits, and a **commit ID** in hex (`d3020f8c`) that changes every time the content does. Commands accept either. When in doubt, use the letters.

### @ — you are here

The working-copy commit: your files, right now, as jj sees them. Always exists, always current — jj re-snapshots it at the start of every command. `jj st` shows what's in it.

### @- — the one below

The parent of `@`. Right after `jj commit`, your freshly sealed work is `@-` (the new `@` on top is empty). That's why bookmarks get set to `-r @-`. Grandparent: `@--`.

### bookmark

A named sticky note pointing at exactly one change. git calls it a branch. `main` is one. It **never moves unless you move it** with `jj bookmark set`. List them: `jj bookmark list`.

### main vs main@origin

Two *different* sticky notes: `main` is yours, on this machine. `main@origin` is jj's record of **where GitHub's copy of main is**. Push makes GitHub match yours; fetch updates your record of theirs.

### tracked

A one-time marriage between `main` and `main@origin` so jj compares and syncs them. Made once per computer with `jj bookmark track main@origin` — and never again. Untracked pairs make push refuse with the exact error on p. 26.

### revision and -r

“Revision” just means *some specific change*, and `-r` is how a command asks “which one?”. Answers can be `@`, `@-`, a change ID like `ptxpotvt`, or a bookmark name like `main`.

### operation

Every command that touches the repo — commit, bookmark move, fetch, push, undo itself — is logged in `jj op log`. `jj undo` reverses the newest one. This is your safety net; section 09 lives here.

### colocated

This vault has both `.jj/` and `.git/`, so git tools (Obsidian plugins, `git log`) still work read-only. Side effect: `git status` says “not currently on any branch” — that is **normal here**, not damage.

## The picture to keep: two sticky notes, one cloud

Everything in this guide is some version of this one picture. When a command confuses you, come back here and ask: *which sticky note is it about, and which commit should that note point at?*

03

SECTION 03

# A worked example

Everything so far, on a real repo you can watch being built from nothing. We use a throwaway vault called `ShadowVault` — a stunt double for the real one — and every block below is captured live from it, fake name and all.

**IN THIS SECTION**

- Meet ShadowVault · a fake identity that won't leak your real one
- Two on-ramps: a fresh GitHub repo, or a folder of notes you already have
- First commit → first bookmark → first push (and why the first push is special)
- The daily loop and the trap, on real output

## The set-up: a stunt double called ShadowVault

The rest of this guide talks about *your* vault. This section builds a **different, disposable one** so nothing you do here can touch your notes. It's a private GitHub repo named `shadowvault`, worked by a made-up identity — `NOMAD` `<nomad@shadowvault.dev>` — so the author line in every capture is a costume, not your real name and email.

### set your identity once, per computer

jj stamps every commit with a name and email. Set them one time and forget it:

writes to /.config/jj/config.toml — do this once, ever

```
jj config set --user user.name  "NOMAD"
jj config set --user user.email "nomad@shadowvault.dev"
```

Check it any time:

you type this

```
jj config list user
```

what you should see

```
user.name = "NOMAD"
user.email = "nomad@shadowvault.dev"
```

Use `--user` for your real, permanent identity. Set it *before* your first commit and you never meet the "author will stay …" warning that appears when you change identity mid-repo.

## On-ramp A — a repo that already lives on GitHub

If GitHub already has the repo (you made it in the browser, or someone shared it), let jj make the local copy. `--colocate` keeps a `.git` beside `.jj` so Obsidian's git plugins still work:

you type this

```
jj git clone --colocate https://github.com/DAEMON-404/shadowvault.git ShadowVault
```

clone does the tracking for you — note "[new

```
 tracked"]
Fetching into new repo in "/home/nomad/git/ShadowVault"
bookmark: main@origin [new] tracked
Setting the revset alias `trunk()` to `main@origin`.
Working copy  (@) now at: rykuulll 3825de67 (empty) (no description set)
Parent commit (@-): qtvtpwxm f34fe73b main | Initial commit
Added 1 files, modified 0 files, removed 0 files
```

That `main@origin [new] tracked` line is the whole "tracking" setup from section 06, done for you. Clone, and you never type `jj bookmark track` at all. Skip to the daily loop.

## On-ramp B — a folder of notes you already have

This is the real NetrunnerVault situation: the notes exist on disk, GitHub is empty, and you want to marry them. Four moves take a plain folder all the way to GitHub.

### 1 Make the folder a jj repo

Stand in the folder and initialise. `--colocate` again, so git tools keep working:

you type this

```
cd ~/git/ShadowVault
jj git init --colocate
```

what you should see

```
Initialized repo in "."
```

Every file already on disk is now sitting in `@`, unsealed — jj snapshots them the instant the repo exists:

you type this

```
jj st
```

A = a file jj is seeing for the first time

```
Working copy changes:
A 00Meta/README.md
A 02Cybersecurity/Cheatsheets/Nmap.md
A 04Tools/tmux.md
Working copy  (@) : txmlmqmp c60baa4f (no description set)
Parent commit (@-): zzzzzzzz 00000000 (empty) (no description set)
```

### 2 Seal them and put the sticky note on the sealed change

you type this

```
jj commit -m "vault: first import of notes"
jj bookmark create main -r @-
```

commit seals @, then create names @- "main"

```
Working copy  (@) now at: kmpmlqyt ea3272ef (empty) (no description set)
Parent commit (@-)      : txmlmqmp 2adbd45d vault: first import of notes
Created 1 bookmarks pointing to txmlmqmp 2adbd45d main | vault: first import of notes
```

### 3 Tell jj where GitHub is

you type this

```
jj git remote add origin https://github.com/DAEMON-404/shadowvault.git
jj git remote list
```

what you should see

```
origin https://github.com/DAEMON-404/shadowvault.git
```

## 4 First push — you must name the bookmark

Here is the one surprise. A plain `jj git push` **refuses to invent a bookmark on GitHub** — the same safety net that stops a half-baked side branch escaping (section 08):

you type this

```
jj git push
```

the refusal — GitHub has no "main" yet, so jj won't guess

```
Warning: Refusing to create new remote bookmark main@origin
Hint: Run `jj bookmark track main@origin` and try again.
Nothing changed.
```

Name it with `-b main` to say "yes, create this one on GitHub" — and jj tracks the pair automatically at the same time:

you type this

```
jj git push -b main
```

created on GitHub, and now tracked — the "remote:" lines are GitHub talking back

```
Changes to push to origin:
  bookmark: main [add to 2adbd45d331b]
remote:
remote: Create a pull request for 'main' on GitHub by visiting:
remote:      https://github.com/DAEMON-404/shadowvault/pull/new/main
```

> **there is no –allow-new**
>
> Reaching for a `--allow-new` flag like some tools have? jj doesn't have one, and it will say so: `error: unexpected argument '--allow-new'`. The blessed way to create a bookmark on GitHub is to *name* it: `jj git push -b main`. From the second push on, plain `jj git push` carries `main` without the `-b`.

## The daily loop — and the trap — on real output

ShadowVault is now on GitHub. From here it is the everyday loop forever. Watch it bite once, so you recognise it. You edit `Nmap.md`, then:

you type this

```
jj st
jj commit -m "nmap: add UDP top-100 sweep"
jj git push
```

you committed... and push says it did nothing. This is THE trap.

```
Working copy changes:
M 02Cybersecurity/Cheatsheets/Nmap.md
Working copy  (@) : kmpmlqyt c4b4eb51 (no description set)
Parent commit (@-): txmlmqmp 2adbd45d main | vault: first import of notes
Working copy  (@) now at: oqoltnmt d79743ef (empty) (no description set)
Parent commit (@-)      : kmpmlqyt 5a24a49b nmap: add UDP top-100 sweep
Warning: No bookmarks/tags found in the default push revset: remote_bookmarks(remote=origin)..@
Nothing changed.
```

You sealed the work, but `main` still points at the import — so push had no moved note to send. The fix is the third move of the ritual, and then push is happy:

you type this

```
jj bookmark set main -r @-
jj git push
```

the star means "ahead of GitHub"; push then fast-forwards GitHub's main

```
Moved 1 bookmarks to kmpmlqyt 5a24a49b main* | nmap: add UDP top-100 sweep
Changes to push to origin:
  bookmark: main [move forward from 2adbd45d331b to 5a24a49be1b7]
```

### checkpoint — how you know it worked

One command proves it landed — all three lines quote the **same** change ID:

you type this

```
jj bookmark list --all-remotes
```

what you should see

```
main: kmpmlqyt 5a24a49b nmap: add UDP top-100 sweep
  @git: kmpmlqyt 5a24a49b nmap: add UDP top-100 sweep
  @origin: kmpmlqyt 5a24a49b nmap: add UDP top-100 sweep
```

From now on the guide keeps using ShadowVault — the divergence in section 07 and the undo in section 09 are both real moments from this same repo. Sections 04–06 slow down and take each move of the ritual apart.

04

SECTION 04

# Save your work

The everyday loop. GitHub is not involved yet — this is just you, your notes, and a clean save point with your name on it. Two commands, thirty seconds.

**IN THIS SECTION**


- Step 1 — `jj st`: see what changed
- Step 2 — `jj commit -m`: seal it
- What “saved” means (and doesn’t mean) for GitHub

There is no step zero. You do not stage, you do not add, you do not tell jj which files count — editing files in Obsidian *is* the first half of saving. When you're at a good stopping point:

## 1 Look at what you changed

you type this

```
cd ~/git/NetrunnerVault
jj st
```

real output — A means added file, M means modified

```
Working copy changes:
M note.md
Working copy  (@) : pqtwmqkw 6fba7d6c (no description set)
Parent commit (@-): ykywxwny 5a23fea5 notes: expand field notes
```

Read it as a sentence: *"@ currently holds a modified `note.md` and doesn't have a description yet; the change below it is the last thing you sealed."*

## 2 Seal it with a message

you type this

```
jj commit -m "notes: expand field notes"
```

### checkpoint — how you know it worked

jj answers by describing your new position — `@` is now **empty** on top, and your work sits at `@-` wearing your message:

what you should see

```
Working copy  (@) now at: qwvxwwts 97614bab (empty) (no description set)
Parent commit (@-)      : ykywxwny 2524f49b notes: expand field notes
```

The word `(empty)` on the `@` line is the success signal: your edits were sealed below, and you're standing on a clean slate.

That's the whole loop. Edit → `jj st` → `jj commit -m`. Repeat all day. Each commit is a save point you can always come back to.

### saved here ≠ visible on GitHub

A commit lives only on this machine until two more things happen: the `main` sticky note moves onto it (section 05), and you push (section 06). If you stop after `jj commit`, GitHub still shows the old vault — with zero warnings. This is the number-one jj gotcha, and it is why the ritual has four steps, not two.

> **commit messages that don't fight you**
>
> jj refuses to push a change whose message is empty, so give every commit at least a few honest words: `"notes: add OSCP buffer overflow page"`, `"update: fix nmap flags"`. Typo in the message? `jj describe -r @- -m "better words"` rewrites it — section 09.

## Commit messages — one line, many lines, or from a file

`-m` is the fast path, but a message can be as long as you like. Every form below is a real capture from ShadowVault.

**One line — the everyday case.**

```
you type this

jj commit -m "nmap: add UDP top-100 sweep"
```

**A title plus a body — repeat `-m`.** Each `-m` becomes its own paragraph, so the first is your one-line summary and the second is the explanation:

```
you type this

jj commit -m "hashcat: add NTLM cracking notes" \
          -m "Covers the -m 1000 mode and the rockyou wordlist path."
```

```
jj show @- reads it back — blank line between title and body, exactly like git

    hashcat: add NTLM cracking notes

    Covers the -m 1000 mode and the rockyou wordlist path.
```

**A message that lives in a file.** Write the message in your editor, save it, and feed the file in with a command substitution — the shell drops the file's whole contents, newlines and all, in place of `-m`'s argument:

```
the message file can be as long and as formatted as you want

jj commit -m "$(cat /tmp/commit-msg.txt)"
```

```
the file's first line became the title; the rest became the body, verbatim

    tmux: add pane, session and zoom cheats

    Why: kept re-googling detach/attach mid-engagement.

    - prefix d        detach
    - tmux a -t NAME  attach by name
    - prefix z        zoom one pane fullscreen
```

### the same trick fixes an old message

`jj describe` edits any change's message, and it reads a file from standard input with `--stdin`:

you type this

```
jj describe -r @- --stdin < /tmp/commit-msg.txt
```

what you should see

```
Rebased 1 descendant commits.
Working copy  (@) now at: wwqoytnz d7fd18e0 (empty) (no description set)
Parent commit (@-)      : pyuuvvoq af1de6ad tmux: add pane, session and zoom cheats
```

("Rebased 1 descendant commits" just means jj carried the empty `@` along onto the re-worded change — nothing of yours moved.)

**No `-m` at all — jj opens your editor.** Run `jj commit` or `jj describe -r @-` with no message and jj drops you into `$EDITOR` (nano if you never set one) on a pre-filled template. Type above the `JJ:` lines; they're comments and get stripped:

the template jj hands your editor — the JJ: lines vanish on save

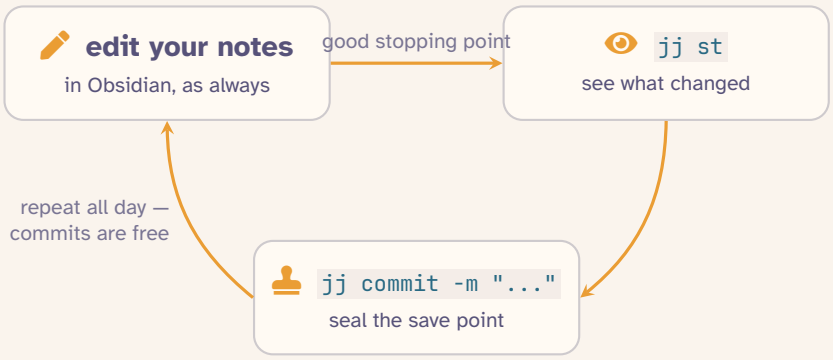
```
tmux: add pane, session and zoom cheats

Why: kept re-googling detach/attach mid-engagement.

JJ: Change ID: pyuuvvoq
JJ: This commit contains the following changes:
JJ:     M 04Tools/tmux.md
JJ:
JJ: Lines starting with "JJ:" (like this one) will be removed.
```

Save and close the editor to keep the message; save it *empty* to abort, and jj leaves the change untouched (`Nothing changed.`).

## The loop, as a loop




When you're ready for GitHub to see any of it, the loop hands off to the ritual: bookmark → push. That's the next two sections.

05

SECTION 05

# Bookmarks, finally explained

The part that's been driving you crazy, taken apart until it can't confuse you anymore. One rule, one command, one way to check it worked.

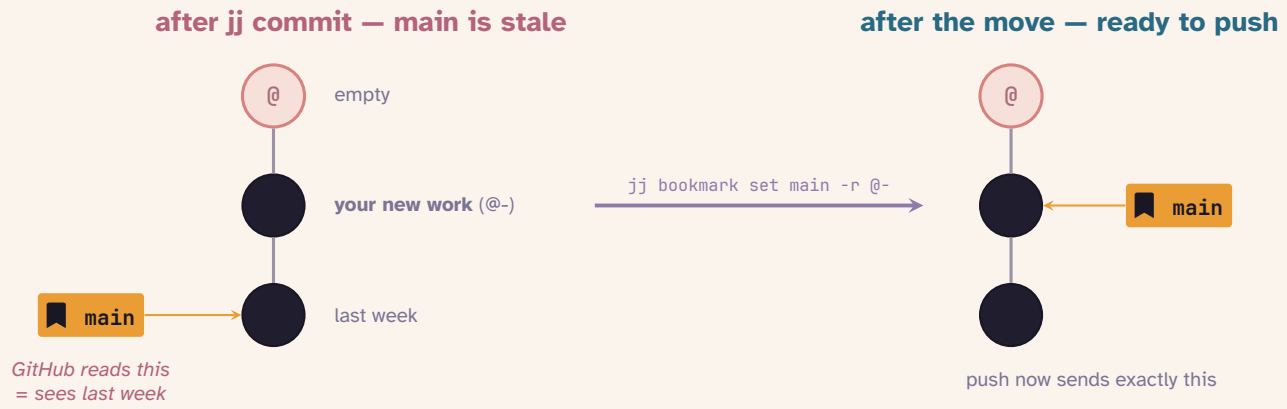
**IN THIS SECTION**


- The sticky-note rule, with before/after pictures
- `jj bookmark set main -r @-` — every word explained
- Why `@-` and not `@`
- How to see where every bookmark is

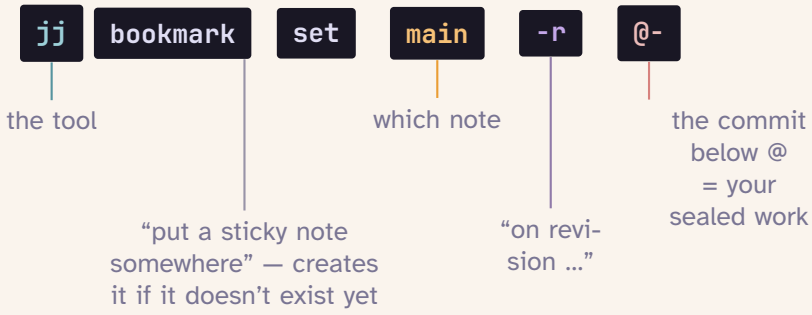
## One rule

A bookmark is a sticky note: a name, stuck to exactly one commit. You already know the rule — **nothing moves it except you**. Watch it bite, then watch the fix.

You just committed. Here's the state — note where `main` is:




## The command, word by word




you type this

```
jj bookmark set main -r @-
```

### checkpoint — how you know it worked

jj states exactly what it did, quoting the commit the note now sits on:

first time it says Created, afterwards Moved

```
Moved 1 bookmarks to ykywxwny 7da4037a main* | notes: expand field notes properly
```

The little `*` after `main` means: *your note has moved ahead of GitHub's copy* — there is something to push. After you push, the star disappears.

> **why it works like this**
>
> Why `@-` and not `@`? Because you run this right after `jj commit`, and commit left `@` **empty** — a blank page waiting for tomorrow. Your finished work is one step down, at `@-`. Pointing `main` at `@` would publish an empty, message-less change; jj would refuse to push it anyway.

## Seeing where every note is

One command answers "where is main, where does GitHub think main is, and did I push?":

you type this

```
jj bookmark list --all-remotes
```

healthy state — all three lines quote the SAME change ID

```
main: ykywxwny 7da4037a notes: expand field notes properly
  @git: ykywxwny 7da4037a notes: expand field notes properly
  @origin: ykywxwny 7da4037a notes: expand field notes properly
```

- `main:` — your sticky note.
- `@git:` — the colocated `.git` directory's copy (bookkeeping; it follows along).
- `@origin:` — **GitHub's copy**, as of your last push or fetch. Indented under `main` means the pair is tracked — exactly what you want.

Reading the three lines as a diagnosis:

- all three quote the **same change ID** — fully published, nothing to do;
- `@origin` quotes an older ID, marked `(behind by 1 commits)` — you moved the note but haven't pushed;
- `main` itself quotes an old ID while `jj log` shows newer work — you committed but forgot the move, so GitHub can't see your notes yet.

## The other bookmark verbs (you'll rarely need them)

| command | what it does |
|---|---|
| `jj bookmark set NAME -r @-` | move (or create) — the only one in the ritual |
| `jj bookmark create NAME -r @-` | create only; errors if NAME exists |
| `jj bookmark list` | your notes only, no remote info |
| `jj bookmark rename OLD NEW` | rename the local note |
| `jj bookmark delete NAME` | delete — **and GitHub's copy dies on next push** |
| `jj bookmark forget NAME` | drop it locally, leave GitHub alone |
| `jj bookmark track NAME@origin` | one-time marriage to the remote note (section 06) |

**delete vs forget**

`delete` means “I want this gone *everywhere*” — the deletion travels to GitHub on your next push. `forget` means “stop tracking it here, leave GitHub alone.” If unsure, you want `forget`.

## Three questions everyone asks

**“Am I ‘on’ main right now?”** No — that question doesn’t exist in jj. You stand on commits (`@` is where you stand); `main` is just a label you move. The closest true statement is “`main` currently points at the commit under me.”

**“Does moving a bookmark change my files?”** Never. It relabels. No file content is touched, nothing is lost, and `jj undo` reverses it — which is why you can run `jj bookmark set` without holding your breath.

**“I set it to the wrong commit — now what?”** `jj undo` puts the note back where it was. Or just set it again with the right `-r`; jj only complains if the move would go *backwards*, and even then it tells you the flag to use if you mean it.

06

SECTION 06

# Push — put it on GitHub

Push does one job: make GitHub's sticky note match yours, carrying whatever commits that requires. Understand that sentence and every push message jj prints becomes readable.

**IN THIS SECTION**

- What push actually sends (bookmarks, not "latest work")
- `track` — the one-time setup, demystified
- The push ritual with real checkpoints
- Every push failure you'll meet, with exact fixes

## What push actually does

`jj git push` does **not** mean "upload my latest work." It means:

> **"For every tracked sticky note that moved: make GitHub's copy match mine."**

If no note moved, there is nothing to say, and push does **nothing — successfully, quietly**. That is the trap you kept falling into: commit without the bookmark move, push, see no error, walk away — and GitHub never changed.

## The one-time setup you already did (and never do again)

`main` (your note) and `main@origin` (GitHub's) start out as strangers who happen to share a name. **Tracking** introduces them, once, permanently:

ONCE per computer — this vault: already done

```
jj bookmark track main@origin
```

what you should see

```
Started tracking 1 remote bookmarks.
```

From then on jj compares them (`behind by 1 commits`), fetch updates yours, and push is allowed to update theirs. Running it again by accident is harmless — jj just tells you it's already tracked.

### ONCE, EVER (per computer)

`jj bookmark track main@origin`

marries your note to GitHub's. Done for this vault — you never type it again.

### EVERY TIME you publish

`jj bookmark set main -r @-`

moves your note onto today's work. This one is part of the ritual, forever.

These two commands look like cousins, which is why they blurred together. They are not: one is plumbing you did once; the other is a daily move.

## The push ritual

### 1 Seal your work

you type this

```
jj st                          # anything listed? then:
jj commit -m "update: vault notes"
```

what you should see

```
Working copy  (@) now at: truwswwt ce476b39 (empty) (no description set)
Parent commit (@-)      : vwxytzts 17231999 notes: expand field notes
```

## 2 Move the note

you type this

```
jj bookmark set main -r @-
```

what you should see

```
Moved 1 bookmarks to vwxytzts 17231999 main* | notes: expand field notes
```

## 3 Push

you type this

```
jj git push
```

### checkpoint — how you know it worked

A real push always names the bookmark and the movement. `move forward` = clean fast-forward, nothing overwritten:

what you should see

```
Changes to push to origin:
  bookmark: main [move forward from d3020f8cd7a4 to 172319998b29]
```

## 4 Trust, but verify

you type this

```
jj bookmark list --all-remotes
```

three lines, one change ID — published

```
main: vwxytzts 17231999 notes: expand field notes
  @git: vwxytzts 17231999 notes: expand field notes
  @origin: vwxytzts 17231999 notes: expand field notes
```

### rehearse a push without doing it

`jj git push --dry-run` prints exactly what *would* happen and touches nothing. Perfect for "wait, is this going to do what I think?"

## Every push failure you will actually meet

1. **It "worked" but did nothing.**

```
the silent no-op — you forgot to move the note

Warning: No bookmarks/tags found in the default push revset:
remote_bookmarks(remote=origin)..@
Nothing changed.
```

*Meaning:* no tracked note moved, so there was nothing to send. *Fix:* `jj bookmark set main -r @-` then push again.

**2. Non-tracking remote bookmark.**

```
a clone that was never introduced

Error: Non-tracking remote bookmark main@origin exists
Hint: Run `jj bookmark track main@origin` to import the remote bookmark.
```

*Meaning:* on *this* computer, `main` and `main@origin` were never married. *Fix:* do exactly what the hint says, once, then push again.

**3. Empty description.**

```
what you should see

Error: Won't push commit 474feb7526f8 since it has no description
```

*Meaning:* the bookmark points at a commit with no message — usually you set it to `@` instead of `@-`. *Fix:* `jj bookmark set main -r @-` (or give the commit a message with `jj describe`).

**4. Sideways or backwards.**

```
what you should see

Error: Refusing to move bookmark backwards or sideways: main
Hint: Use --allow-backwards to allow it.
```

*Meaning:* you asked `bookmark set` to move the note to an *older or unrelated* commit — usually a typo in `-r`. *Fix:* double-check the target; you almost never actually want `--allow-backwards`.

**5. GitHub moved while you weren't looking** — `main??` in `jj log`. That's divergence, and it has its own section: **06**, next.

07

SECTION 07

# Fetch & sync

There is deliberately no “jj pull”. Downloading GitHub’s history and weaving it into yours are two separate moves with a look in between — that look is what keeps your vault safe.

**IN THIS SECTION**

- The safe sync ritual: seal → fetch → look → rebase
- What fetch does and does not touch
- `main??` — divergence, and the two clean ways out

## The sync ritual

### 1 Seal anything half-done

Never sync over a messy desk. If `jj st` lists changes:

you type this

```
jj commit -m "wip: checkpoint before sync"
```

### 2 Fetch — download only

you type this

```
jj git fetch
```

**checkpoint — how you know it worked**

Fetch updates your record of GitHub (`main@origin`) and **never edits your files**. Two healthy answers:

nothing new on GitHub (usual on a one-computer vault)

```
Nothing changed.
```

GitHub had news — your record updated

```
bookmark: main@origin [updated] tracked
```

### 3 Look before you merge anything

you type this

```
jj log
```

If `main@origin` sits directly *below* your work (a straight line), you were simply ahead — carry on, nothing to integrate. If it sits on a *side line* next to your commits, or you see `main??`, keep reading.

### 4 Bring your work on top of theirs

only needed when GitHub actually had new commits

```
jj rebase -o main@origin       # replay my stack onto GitHub's newest
jj bookmark set main -r @-     # put the note on the rebased tip
jj git push
```

> **never plain git pull in this vault**
>
> `git pull` goes behind jj's back, can jam the colocated `.git` into a merge state, and can silently combine two versions of a file. If you crave `git pull`, the jj spelling is: `jj git fetch`, look, `jj rebase -o main@origin`.

## main?? — when both sides moved

Here is the real thing, from ShadowVault. On the laptop you edited `Nmap.md` to add `--min-rate 5000` and pushed. On the desktop, not knowing that, you edited the *same line* to add `-T4` and moved `main` locally. Then you fetched:

```
jj bookmark list --all-remotes — two "+" candidates = both sides moved

main (conflicted):
  - pyuuvvoq af1de6ad tmux: add pane, session and zoom cheats
  + tkqsrolo 65bc137b nmap: add -T4 to fast sweep
  + lxmrmuxm fd61cf23 nmap: bump fast sweep to --min-rate 5000
  @git (behind by 1 commits): tkqsrolo 65bc137b nmap: add -T4 to fast sweep
  @origin (behind by 1 commits): lxmrmuxm fd61cf23 nmap: bump fast sweep to --min-rate 5000
Hint: Some bookmarks have conflicts. Use `jj bookmark set <name> -r <rev>` to resolve.
```

Read it as: the note used to sit on the `-` line; now there are two `+` candidates, yours (`-T4`) and GitHub's (`--min-rate`), and jj won't pick. In `jj log` the same standoff shows the note as `main??` on both commits:

```
jj log — main?? tags both candidates; nothing is lost

@  lvvmtnup 17710d6a
|  (empty) (no description set)
o  tkqsrolo main?? main@git 65bc137b
|  nmap: add -T4 to fast sweep
| o  lxmrmuxm main?? main@origin fd61cf23
|/   nmap: bump fast sweep to --min-rate 5000
o  pyuuvvoq af1de6ad
|  tmux: add pane, session and zoom cheats
~
```

Nothing is lost — jj is just refusing to guess. Pick one of two exits:

### keep BOTH (usual choice)

Replay your local commits on top of GitHub's, then point the note at the result:

```
jj rebase -o main@origin
jj bookmark set main -r @-
jj git push
```

If the same lines changed on both sides, jj marks a conflict in the file — section 09 shows the calm fix.

### GitHub WINS

Adopt GitHub's version; your local commits stay visible in `jj log` for salvage:

```
jj bookmark set main -r main@origin
jj new main
```

`jj new main` gives you a clean working copy on top of the adopted version.

08

SECTION 08

# Side branches

A draft you want on GitHub without touching main. In jj you don't need a branch to start working — you only need a bookmark at the moment GitHub has to see your work.

**IN THIS SECTION**


- Start a side line · name it · push it
- Switching between main and the draft
- Updating the draft — same move as main
- Deleting it everywhere when you're done

## Start, name, push

### 1 Start a fresh line on top of main

you type this

```
jj new main
```

This parks whatever you were doing (nothing is lost — it's all in `jj log`) and gives you a clean empty `@` sitting on `main`. Now edit files as usual and seal:

you type this

```
jj commit -m "draft: crazy idea"
```

### 2 Stick a new note on it

you type this

```
jj bookmark create idea -r @-
```

what you should see

```
Created 1 bookmarks pointing to rsmormok d1d1d080 idea | draft: crazy idea
```

### 3 Push it by name (first time only needs -b)

you type this

```
jj git push -b idea
```

new on GitHub — and automatically tracked, no track command needed

```
Changes to push to origin:
  bookmark: idea [add to d1d1d080f85f]
```

> **why it works like this**
>
> A plain `jj git push` refuses to invent bookmarks on GitHub — that's a safety net so a half-finished experiment never escapes by accident. Naming it with `-b idea` is you saying "yes, this one, publish it," and jj then tracks the pair automatically. From the second push onward, plain `jj git push` carries `idea` along with `main`.

## Switching around

You're never "on a branch" — you just choose what to stand on next:

you type this

```
jj new main          # stand on main         (work on the vault)
jj new idea          # stand on the draft    (continue the idea)
```

after jj new idea — @ sits on the draft again

```
Working copy  (@) now at: kpvvxylm 5c704f00 (empty) (no description set)
Parent commit (@-)      : rsmormok d1d1d080 idea | draft: crazy idea
```

Files in your folder switch instantly to match. Uncommitted edits? They live in `@`, and `jj new` simply leaves that `@` behind as its own change — retrieve it any time from `jj log`. This is why jj has no stash: nothing ever needs stashing.

## Updating the draft — the exact same move as main

More work on the idea? The pattern you already know, with a different name in the middle:

you type this

```
jj commit -m "draft: more crazy idea"
jj bookmark set idea -r @-
jj git push
```

what you should see

```
Changes to push to origin:
  bookmark: idea [move forward from d1d1d080f85f to a66675ab4b45]
```

One pattern, every bookmark: *commit, set, push*. Learn it once, use it everywhere.

## Done with it — delete everywhere

you type this

```
jj bookmark delete idea
jj git push --deleted
```

what you should see

```
Changes to push to origin:
  bookmark: idea [delete from a66675ab4b45]
```

The commits themselves stay in `jj log` — only the label is gone. To keep the label locally but remove nothing from GitHub, use `jj bookmark forget idea` instead.

09

SECTION 09

# When things go wrong

jj records every operation you run, which means almost nothing is permanent. This is the calmest section in the guide — read it once now so future-you panics less.

**IN THIS SECTION**


- `jj undo` — reverse any last move
- The operation log — your security-camera footage
- Forgot a line · wrong message · conflicts
- The debug ladder

## Undo anything

Every command that changed the repo — commit, bookmark move, fetch, rebase, even a botched undo — is one **operation**. Say you fat-fingered a bookmark in ShadowVault, dragging `main` onto the wrong draft. `jj undo` reverses the newest operation, whatever it was:

you type this

```
jj undo
```

real capture — it names the operation it reversed, and where it landed you

```
Undid operation: 660868458073 (2026-08-28 19:08:21) point bookmark main
to commit 51bbfe901b871d1a1653ee921abaf5e0f7ec14d8
Restored to operation: 0298bbd82c8f (2026-08-28 19:08:08) push bookmark main to git remote origin
```

Undid the wrong thing? `jj redo` puts it back. For anything older than "the last thing", open the footage:

you type this

```
jj op log -n 5
```

newest first — the args line names the exact command you ran

```
@  9dc246f03e53 nomad@shadowvault now, lasted 3 milliseconds
|  undo: restore to operation 0298bbd82c8f...
|  args: jj undo
o  660868458073 nomad@shadowvault now, lasted 4 milliseconds
|  point bookmark main to commit 51bbfe901b87...
|  args: jj bookmark set main -r bloodhound-draft --allow-backwards
o  0298bbd82c8f nomad@shadowvault 13 seconds ago, lasted 1 second
|  push bookmark main to git remote origin
|  args: jj git push
```

time machine — restore the whole repo to that moment by its id

```
jj op restore 0298bbd82c8f
```

> **this is why nothing in this guide is scary**
>
> Moved the wrong bookmark? `jj undo`. Rebase made a mess? `jj undo`. Fetched something weird? `jj undo`. The operation log is repo-wide and includes bookmark moves — git's reflog never covered you like this.

## Small repairs

**Forgot a line in the commit you just made.** Make the edit, then fold it down into `@-`:

you type this

```
jj squash
```

the fix melts into the previous commit; @ is empty again

```
Working copy  (@) now at: qwvxwwts 97614bab (empty) (no description set)
Parent commit (@-)      : ykywxwny 2524f49b notes: expand field notes
```

**Wrong commit message.**

you type this

```
jj describe -r @- -m "notes: expand field notes properly"
```

**Committed, moved the note, pushed — then found a typo.** Just go again: fix the file, `jj commit`, `jj bookmark set main -r @-`, `jj git push`. Another commit costs nothing. (History rewriting exists in jj, but you don't need it to run this vault.)

## Conflicts, without the panic

When a rebase makes two versions of the same lines collide, jj does *not* stop the world with "fix this NOW" like git. It writes conflict markers into the file, marks the commit `(conflict)` in `jj log`, and lets you resolve whenever you like. First, ask what's conflicted:

you type this

```
jj st
```

the ShadowVault --min-rate vs -T4 clash from section 07

```
Working copy  (@) : lvvmtnup 29783d55 (conflict) (empty) (no description set)
Parent commit (@-): tkqsrolo fec93595 main* | (conflict) nmap: add -T4 to fast sweep
Warning: There are unresolved conflicts at these paths:
02Cybersecurity/Cheatsheets/Nmap.md 2-sided conflict
```

Open that file. jj's markers are **not** git's `<<<<<<< / ======= / >>>>>>>`. One side is written out in full after `+++++++`; the other is shown as a *diff* after `%%%%%%%`:

the real markers jj 0.44 writes into Nmap.md

```
## Fast sweep
<<<<<<< conflict 1 of 1
+++++++ lxmrmuxm "nmap: bump fast sweep to --min-rate 5000" (rebase destination)
nmap -sC -sV --min-rate 5000 -oA scans/initial 10.10.11.0/24
%%%%%%% diff from: pyuuvvoq "tmux: add ..." (parents of rebased revision)
\\\\\\\        to: lvvmtnup (rebased revision)
-nmap -sC -sV -oA scans/initial 10.10.11.0/24
+nmap -sC -sV -T4 -oA scans/initial 10.10.11.0/24
>>>>>>> conflict 1 of 1 ends
```

To resolve: delete everything from `<<<<<<<` to `>>>>>>>` and type the one line you actually want — here, both flags together: `nmap -sC -sV -T4 --min-rate 5000 -oA scans/initial 10.10.11.0/24`. Save the file, then fold the fix down:

you type this

```
jj squash                # after saving Nmap.md, fold the fix into the conflicted commit
```

the conflict is gone; @ is empty again, ready to bookmark + push

```
Working copy  (@) now at: tttswwnw 95baa06e (empty) (no description set)
Parent commit (@-)      : tkqsrolo 2084c3d6 main* | nmap: add -T4 to fast sweep
Existing conflicts were resolved or abandoned from 1 commits.
```

No `--continue`, no `--abort`, no detached-HEAD purgatory. And jj refuses to push conflicted commits, so a half-resolved mess can never reach GitHub.

## The debug ladder

When something feels off, climb until it makes sense:

you type this

```
jj st                          # 1. what does jj think just changed?
jj bookmark list --all-remotes # 2. where is every sticky note?
jj log                         # 3. what shape is history in?
jj op log -n 10                # 4. what did I actually run?
jj undo                        # 5. reverse the last thing I ran
```

010

SECTION 010

# The reference card

Everything from the guide on two pages. This is the part to print, screenshot, or keep open in a split pane while the habits form.

**IN THIS SECTION**


- The ritual (again, on purpose)
- Task → command, grouped
- Symptom → fix

### THE RITUAL

every publish, in order

```
jj st                          # look
jj commit -m "update: ..."     # seal
jj bookmark set main -r @-     # move the note
jj git push                    # send
```

## Task → command

| start a repo | |
|---|---|
| clone one that's on GitHub | `jj git clone --colocate <url> Dir` |
| turn this folder into a repo | `jj git init --colocate` |
| point it at GitHub | `jj git remote add origin <url>` |
| first push of a brand-new main | `jj git push -b main` |
| **see** | |
| what did I change? | `jj st` |
| history graph | `jj log` |
| where is every bookmark? | `jj bookmark list --all-remotes` |
| diff of my current edits | `jj diff` |
| diff of one commit | `jj show @-` |
| what would push do? | `jj git push --dry-run` |
| **save** | |
| seal my work with a message | `jj commit -m "..."` |
| title + body (repeat -m) | `jj commit -m "title" -m "body"` |
| message from a file | `jj commit -m "$(cat msg.txt)"` |
| add a forgotten fix to last commit | `jj squash` |
| fix last commit's message | `jj describe -r @- -m "..."` |
| rewrite a message from a file | `jj describe -r @- --stdin < msg.txt` |
| **publish (main)** | |
| move the note onto my work | `jj bookmark set main -r @-` |
| send to GitHub | `jj git push` |
| one-time link to GitHub's main | `jj bookmark track main@origin` |
| **get from GitHub** | |
| download news (touches no files) | `jj git fetch` |
| put my work on top of theirs | `jj rebase -o main@origin` |
| adopt GitHub's version wholesale | `jj bookmark set main -r main@origin` |
| **side branches** | |
| start a draft line on main | `jj new main` |
| name it | `jj bookmark create idea -r @-` |
| first push of a new branch | `jj git push -b idea` |
| update it later | `jj bookmark set idea -r @-  then push` |
| switch: main / draft | `jj new main · jj new idea` |
| delete everywhere | `jj bookmark delete idea then jj git push --deleted` |
| **rescue** | |
| reverse the last operation | `jj undo (redo: jj redo)` |
| see every operation | `jj op log -n 10` |
| whole-repo time machine | `jj op restore <id>` |

## Symptom → fix

| you see | it means / do this |
|---|---|
| push prints `Nothing changed.` | no note moved — `jj bookmark set main -r @-`, push again |
| `Non-tracking remote bookmark main@origin exists` | run `jj bookmark track main@origin` once, push again |
| GitHub shows old files, `jj st` clean | committed but never moved the note — same fix as above |
| `Won't push commit ... has no description` | note points at empty `@` — set it to `@-` |
| `main??` in the log | divergence — section 07: rebase onto `main@origin` or adopt it |
| `Refusing to move bookmark backwards` | typo in `-r` target — recheck; you rarely want `--allow-backwards` |
| `(conflict)` in the log | edit the markers, `jj squash` — section 09 |
| `git status`: “not currently on any branch” | normal in a colocated repo — read `jj bookmark list` instead |
| did something dumb | `jj undo`. Truly. |

the jj field guide · built with tectonic + rose pine dawn · verified on jj 0.44.0 · 2026-08-28