An Intro to JJ

Lucas Helme

git can have terrible UX

- Interactive rebase is fragile, complex, and prone to breaking.
- Often this leads to messy incremental commits (like `add tests`, `fix CI`), instead of ideally atomic commits.
- Rebasing stacked branches manually when parent branches update is tedious and error-prone.
- Large PRs are often attempted solutions to these problems, but have their downsides.

introducing jujutsu (jj)

What is it?

- **A new front-end for git**: works on top of the existing git protocol and inside existing repos, so teammates don't need to switch.
- **No index/staging area**: files edited in the working copy are tracked automatically.
- **First-class safety**: every operation can be reverted cleanly using `jj undo`.

setup

If you want to follow along, install jj:

```
brew install jj
```

To activate jj in an existing git repo:

```
jj git init
```

revisions

- Revisions are jj's most basic unit of change. A collection of revisions is called a *revset*. This is best seen with the `jj log` command.
- Revsets use a powerful, functional expression language:
  - `@`: current working copy
  - `x-` / `x+`: parent / child of revision `x`
  - `::x` / `x::`: ancestors / descendants of `x`
  - `|` / `&` / `~`: union, intersection, and difference (negation)
  - *Example*: `jj log -r "all() ~ ancestors(origin/main)"`

changes

There are no traditional “commits”. Instead, there are two core commands: `describe` and `new`.

- `jj describe`: set or update the description/message of *any* mutable change at any time (e.g. `jj describe -m "fix bug"`), without needing a rebase.
- `jj new`: starts a new change at any point in the history.
  - `-A, --insert-after`: Insert the new change between the target commit(s) and their children.
  - `-B, --insert-before`: Insert the new change between the target commit(s) and their parents.
- Edited files automatically become part of the current working copy.
- You can go back and edit any change using `jj edit`.

bookmarks

In jj, instead of branches we have **bookmarks**. They are pointers to revisions, primarily used to push to remote servers.

- **Create**: `jj bookmark create <name>` (or `jj b c <name>`)
- **Move**: `jj bookmark set <name>`
- **Track**: `jj bookmark track <name>@origin`
- **Push/Fetch**: `jj git push` / `jj git fetch`

Note: Bookmarks do **not** automatically move to your new changes as you commit. You update them as you work. Helpful for this is `jj bookmark advance`, which moves the last bookmark to your current revision.

split

Since there is no Git-style index, you don't stage files with `git add`. To split a change:

- Run `jj split` (or `jj split --interactive`).
- Whatever you *remove* from the right-side diff is split into a second, new child change.
- **Partial Stashes**:
  - `jj split --parallel <file>` moves a file's changes into a parallel revision.
  - Run `jj squash --from <parallel-change-id>` to merge it back.

squash & move

Moving and combining changes in jj is extremely flexible and doesn't require a complex interactive rebase.

- **Squashing**:
  - `jj squash`: squashes all changes in the current working copy `@` into its parent `@-`.
  - `jj squash -r <rev>`: squashes a specific revision into its parent.
  - `jj squash -i`: interactively select parts of a change to squash.
- **Moving**:
  - `jj move --from <rev1> --to <rev2>`: moves changes between any two arbitrary revisions in the repository, even if they aren't adjacent in history.

conflicts

Conflicts in jj are not roadblock events. A revision can exist in a "conflicted" state in your history.

- You can commit conflicts, rebase them, create new changes on top of them, or switch away from them freely.
- Conflict markers are added to the files themselves.
- **Resolution**:
  - Edit the conflict markers directly, or run `jj resolve` to launch a 3-way merge tool.
  - Once resolved, all descendant revisions automatically rebase.

interoperating with git

You can use `jj` on top of any existing Git repo without affecting your teammates.

- use `jj git fetch` to pull changes from a git remote.
- downside: since we rewrite history, this means that we can have a lot of force pushes to a PR.

in summary - cool things you can do

- `jj new @-`, somewhat analagous to `git stash`
- moving things between stacked github PRs / commits.
- editing commits in response to PR feedback.

further reading

- **Official Documentation**: jj-vcs.github.io/jj