JJ Reference

This is a *reference* for the Jujutsu version control system. It exists to help you learn and remember the details of Jujutsu, once you have already grokked the basics.

Model

A Jujutsu repository is a DAG (directed acyclic graph) whose nodes are called *changes*. Each change has:

- A state of the filesystem within the repository directory. You can imagine each change storing a full copy of the directory and all the files in it, though of course `jj` is more efficient than this.
- File *conflicts*. Some files in a change may contain conflicts, from a variety of different sources. These conflicts are local to the change. (Unlike `git`, they do not block your use of `jj`.)
- One or more *parent* changes. (Though there is a root change which has no parents and always has an empty directory).
- A textual *description* of the change, a.k.a. a commit message. This is always present, but defaults to the empty string.

There is some additional information attached to the DAG:

- Exactly one of the changes is the *working copy revision*, a.k.a. `@`. (This is analogous to `git`'s `HEAD`.)
- There may be some *bookmarks*, which are unique string labels on changes. (When interfacing with `git`, these bookmarks act as branch names.)
- The repository may also be linked to a *remote repository* (e.g. Github). If so, when `push`ing and `fetch`ing, `jj` records the *last known position* of each remote bookmark, written `BOOKMARK@REMOTE` (e.g. `feat-ui@origin`).

Most `jj` commands modify your local repository DAG in some way. Some general rules will help you predict how it responds to modifications:

- When you make `@` point at a change, your repository directory is updated to match that change's files.
- If you delete the change that `@` is pointing at, `@` moves to a new empty change off of its parent.
- A change represents a diff. Moving a change tries to apply the diff to its new parent, but this may cause merge conflicts.
- Many commands act on `@` by default. Almost all of them can take a `-r/--revision` argument to act on a different change. (Of the commands in the Cheat Sheet that show `@`, all can be applied to a different change using `-r` except for `jj bookmark move` and `jj restore`, which take `--from` and `--to` arguments instead.)

Global Setup Commands

- `jj config set --user name MY_EMAIL`. Set your name for signing commits.
- `jj config set --user email MY_EMAIL`. Set your email address for signing commits.
- `jj config edit --user`. Manually edit the configuration file.

Instead of `--user`, you can pass `--repo` to change the repository specific config, which takes priority.

Repository Commands

- `jj git init`, or `jj git clone URL [DESTINATION]`. Make or clone a git-backed repo.

- `jj git init --colocate`. Make an existing `git` repo also be a `jj` repo.

jj git push

`jj git push` copies changes from the local repo into the remote repo. If a change is present both locally and remotely, the remote change is updated to match the local change. (Though typically the two changes will be identical.) TODO: Test this!

All local bookmarks are similarly copied to the remote repo. However, if a bookmark is present both locally and remotely, `jj` checks if its (locally recorded) *last seen position* matches its current position in the remote repo. If so, its position is updated in the remote repo. If not, this command fails and tells you to `jj git fetch` first (because it means that someone else updated the bookmark since you last pushed it).

jj git fetch

`jj git fetch` copies changes from the remote repo into the local repo. If a change is present both locally and remotely, the local change is updated to match the remote change. (Though typically the two changes will be identical.) TODO: Test this!

Similarly, all remote bookmarks are copied into the local repo. If a bookmark is present both locally and remotely *at different changes*, `jj git fetch` creates a second copy of that bookmark, violating the invariant that bookmarks are unique. (This is analogous to the situation where `git pull` produces a merge conflict.) It is up to you how to resolve this "bookmark conflict":

- If you want to merge the two changes, say `jj new CHANGE-ID-1 CHANGE-ID-2`, resolve any file conflicts, then update the bookmark with `jj bookmark set BOOKMARK-NAME`. (You can get the change ids by running `jj bookmark list BOOKMARK-NAME`.)
- If you want to discard one of the two changes and just use the other one, say `jj bookmark set CHANGE-ID` for the change you want to keep.
- If you want to rebase one of the changes to come *after* the other, say `jj rebase -b SECOND-CHANGE-ID -d FIRST-CHANGE-ID`, then `jj bookmark set BOOKMARK-NAME`. (This will rebase not only the second change itself, but all changes after it forked away from the first change.)

Editing your local repo

The attached JJ Cheat Sheet visually describes the most common/fundamental commands for editing a `jj` repo. Keep it on your desk while learning `jj`!

There are also a couple of "alias" commands that are best thought of as combinations of other `jj` commands:

- `jj commit`. Shorthand for `jj describe; jj new`.
- `jj bookmark set BOOKMARK`. Either `create` or `move` the bookmark, whichever is valid.

JJ Cheat Sheet

jj
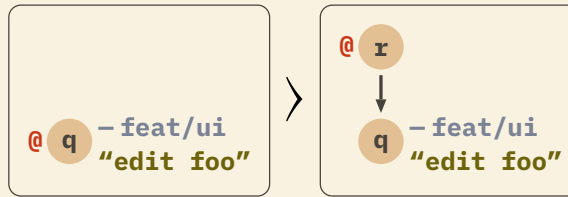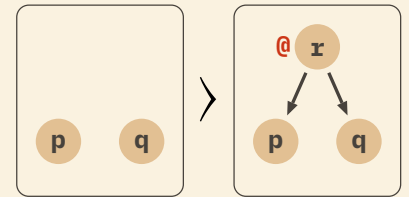Shows important changes in the repo.
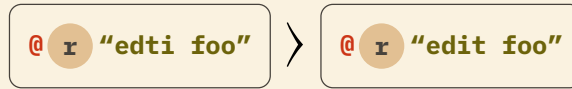
jj undo
Undoes the last command.

jj new

@ q — feat/ui "edit foo" > @ r → q — feat/ui "edit foo"

jj new p q

p   q > @ r → p, q

jj status
Shows current and parent change, and file modifications.

jj describe -m "edit foo"

@ r "edti foo" > @ r "edit foo"

jj show

@ r "edit foo"

Prints this change's description.

jj bookmark list

r — feat/ui   q — feat/api
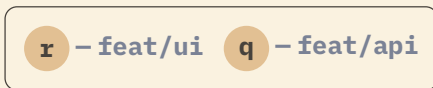
Prints all bookmarks.

jj bookmark create feat/ui

@ r > @ r — feat/ui

jj bookmark delete feat/ui

r — feat/ui > r

jj bookmark move feat/ui

q — feat/ui   @ r > q   @ r — feat/ui
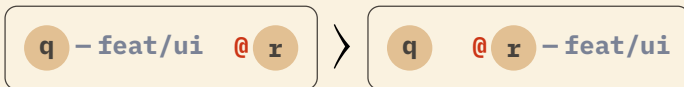
jj bookmark rename feat/ui feat/ux

r — feat/ui > r — feat/ux

jj edit q

@ r   q > r   @ q

jj restore --from q (paths..)

@ r Files2   q Files1 > @ r Files1   q Files1

jj backout

@ r "msg" Files2 → q Files1 > s "Backout 'msg'" Files1 → @ r "msg" Files2 → q Files1

jj abandon q

r Edit3 → q Edit2 → p Edit1 > r Edit3 → p Edit1

jj diff (paths..)

@ r Files2 → q Files1
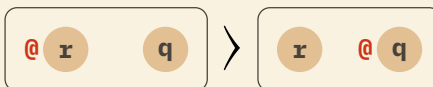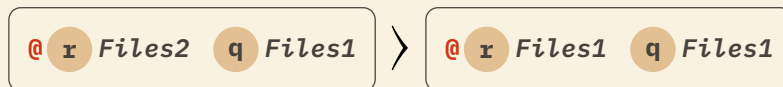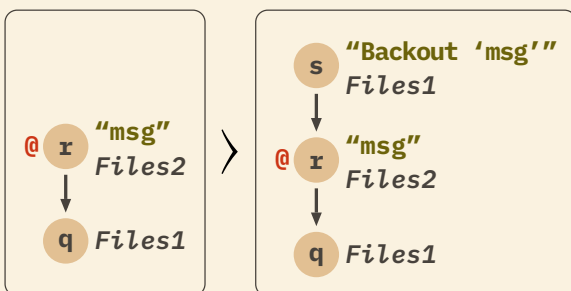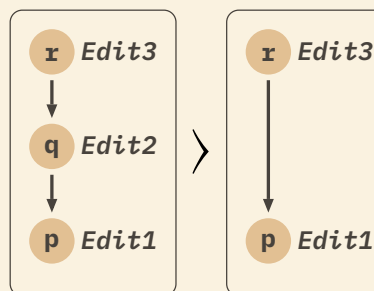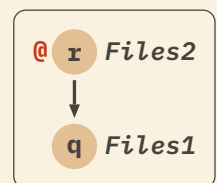
Prints the diff between *Files1* and *Files2*.

jj squash

@ r Files2 → q Files1 > @ r Files2 → q Files2

---
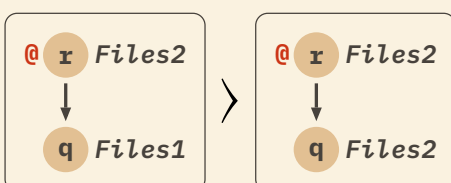
r — a *change*
@ — the working change ("*working copy revision*")
"edit foo" — a change's description
feat/ui — a *bookmark*
Files1 — a state of the filesystem
Edit1 — a change's diff

justinpombrio.net
& lark.gay
Feb 2025