

Jujutsu - jj

Version Control != Git

Hackerschool

2025-04-17

"who is this guy?"

- Y4 CS
- found out about `jj` ~1.5 years ago
 - looks pretty interesting/cool
 - tried it out

"wtf? what? how? i have no idea what's going on"

- someone (great guy 👍) i know wrote a blog
 - <https://blog.chay.dev/basic-jj-workflows/>
 - more and more community guides are popping up
- using `jj` for about ~2 months now
 - clearly not a expert


"why should i care about jj?"

1. version control is (objectively) useful
2. `git` is not the only VCS in the world
 - e.g. `mercurial`, `sapling`
3. a different set of semantics to think about version control
4. it is written in `rust` /s
5. most importantly, "why not?"
 - `hacker spirit` 🦄

convince you that `jj` is a viable alt to `git`

at a glance

- `jj` is the cli, "Jujutsu" is the name
- created by [Martin von Zweigbergk](#) @ Google
 - started out as his 20% project
 - now he works on it full time

 pursue your passion projects! you never know where it will lead to

- `git` compatible
- you can continue collaborating with everyone on `git[hub/lab/bucket/ea]`
 - no one will know you are using `jj`
- `jj` uses a regular `git` repo to store "data" (e.g. commits)
 - supports different backends*
- you can still use `git` in a repo managed by `jj`

getting started

Installation

ref: <https://jj-vcs.github.io/jj/latest/install-and-setup/>

Mac via Homebrew:

```
brew install jj
```

Linux / Windows via cargo:

```
# Linux
cargo install --locked --bin jj jj-cli

# Windows
cargo install --locked --bin jj jj-cli --features vendored-openssl
```

Configuration

setting user-level (global) name and email for your commits

```
jj config set --user user.name "<your name>"
jj config set --user user.email "<your email>"
```

Note

remove `--user` to set repo-level config (run the config at the root of the repo)

get the path to config file:

```
jj config path --user
```

edit the global config file (defaults to `~/.jjconfig.toml`):

```
jj config edit --user
```

baby steps

■ Initializing a new repo

```
# `--colocate`: creates repo that interop with both `jj` and `git`  
jj git init test-repo --colocate
```

you should see both `.jj` and `.git` directories in the target directory

```
ls -lah test-repo
```

let's take a look how `jj`'s revision history compares to `git`'s commit history:

```
jj log
```

```
git log
```

working copy

"the place where files are read from in order to create a commit"

1. `jj` automatically* adds changes from the working copy to the commit
2. "working copy commit" - represented by the `@` symbol

```
@ yuvlmvtu yihong@nushackers.org 2025-04-11 17:13:56 fa815c28
| (empty) (no description set)
◆ zzzzzzzz root() 00000000
```

- `(empty)` - the revision/commit is empty (no changes)
- `(no description set)` - no commit message

3. no staging area

`git`

```
touch hello.txt;
git add hello.txt

# Make some changes to hello.txt
git add hello.txt
```

`jj`

```
touch hello.txt

# Make some changes to hello.txt

# The changes are automatically*
# added to the working copy commit
```


creating and editing a change

There is already an empty change created when we initialize the repo.

Therefore, we can simply just make the intended edits and it will be added automatically to the working copy commit.

■ Creating a new (empty) revision

```
# creates a new revision on top of `@`  
jj new  
  
# creates a new revision on top of >= 1 parents  
jj new <id-1> <id-2> ...
```

■ Editing a previous revision

there are 2 ways to do it, each with its own pros and cons:

■ Create a new change and move/add it into a previous revision

```
jj new  
  
# move changes from `@` to revision specified by <id>  
jj squash --into <id>
```

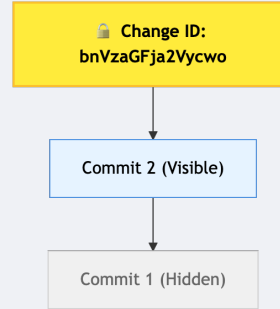
■ Directly editing the previous change

```
jj edit -r <id>  
  
# now `@` is the revision specified by <id>
```

change id vs commit id

`<id-N>` (in 99.9% of cases) refer to change IDs

"A change is a commit that evolves around time"



- Change ID remains the same
- Every new edit (aka a new snapshot) to the working copy (aka a new snapshot) creates a new commit (with a new ID)

deciphering jj log

```
@ yuvmvtu yihong@nushackers.org 2025-04-11 17:13:56 fa815c28  
| (empty) (no description set)  
◆ zzzzzzzz root() 00000000
```

yuvmvtu	yihong@nushackers.org	2025-04-11 17:13:56	fa815c28
Change ID	Author Email	Commit Time	Commit ID

evolving a commit

for each `jj log` command, take note of the commit and change IDs

```
touch hello.txt
```

```
jj log
```

```
touch world.txt
```

```
jj log
```

inspecting the change

"looking at how a change has evolved over time"

```
jj evolog
```

```
# Inspect the patches between the commits in the change  
jj evolog -p
```

Note

You can also "branch off" a (hidden) commit 🧐

inspecting repo status

High-level overview/summary

```
# shorthand: jj st  
jj status
```

Inspect file content changes

```
# inspect file changes from parent revision (`@-`) to working copy commit (`@`)  
jj diff  
  
# to compare between different revisions  
jj diff --from old --to new
```

writing a description/commit message

"giving meaning to a change"

■ 30s personal rant

help your future self and others out by writing good descriptions/commit messages

■ ⚠ Warning

■ bad commit messages: 'fix', 'works now', 'oops', 'yay new button'

<https://conventionalcommits.org>

a better description:

■ feat: allow provided config object to extend other configs

by prefixing the type, one can easily filter the history based on the type of change (e.g. `feat`, `fix`, `chore`, etc.)

you are also encouraged to add more context below the title message either in the form of bullet points or in prose

writing a description/commit message

"giving meaning to a change"

`jj` allows empty and description-less changes

Writing a description for the working copy commit

```
# shorthand: `jj desc`  
jj describe
```

Write/Edit a description for a specific change

```
# `@-` refers to the immediate parent of the working copy commit  
jj desc -r @-
```

Note

`jj` supports a language (revset) for selecting changes/revisions (<https://jj-vcs.github.io/jj/latest/revsets/>)

- You can edit/change the description of a change/revision at any time
- With this flexibility, you now have the freedom to develop new workflows!

workflows

Describe-First

for every change i intend to make, let me write the description first and then make the change

```
jj new  
jj desc  
# make the change  
# repeat for every subsequent change
```

Edit-First

for every change i intend to make, let me make the change first and then write the description

This is the default workflow many adopt when using `git`

```
jj new  
# make the change  
jj desc  
# repeat for every subsequent change
```

Do-Everything-First-Split-Into-Commits-Later

i have a feature, let me implement the entire feature and then split it into commits/multiple smaller changes

is this possible with `jj`? remember that `jj` automatically adds changes to the working copy commit 🤔

```
jj new  
# cook up the entire feature  
# okay, now what?
```

splitting revisions

recall that we do not have a staging area

therefore, we need to a way to break the working copy commit into multiple smaller changes

■ Split a revision into 2

```
# split the working copy commit into 2 smaller changes
jj split

# split a specific revision
jj split -r <revset>
```

jj comes with a builtin diff editor that you can use to choose the changes you want to split

🕒 Note

After you split a revision, you stay on parent revision

🇺🇸 A tweeny weeny gotcha 🏠

try inspecting the "time" of the commit using `jj log` and `git log`:

`jj log`

```
o yuvmvtu yihong@nushackers.org 2025-04-12
18:37:15 c02867c9
| (no description set)
◆ zzzzzzz root() 00000000
```

Commit Time

"last modified timestamp of the commit"

`git log`

```
commit c02867c9036e91432deff2a34a6b9cc2c7aceb10
Author: Shen Yi Hong <yihong@nushackers.org>
Date: Fri Apr 11 17:13:56 2025 +0800
```

Author Time

"when the commit was created"

```
# Reset the author time to the current timestamp
jj desc -r <revset> --reset-author --no-edit
```

bookmarks

named pointers to revisions (conceptually similar to `git` branches)

however, this is no concept of an active bookmark (branch) in `jj`

Creating a bookmark

equivalent to creating a local branch in `git` semantics

```
# add a bookmark to the current working copy commit
jj bookmark create <name>

# add a bookmark to a specific revision
jj bookmark create -r <revset> <name>
```

Listing bookmarks

```
jj bookmark list
```

Updating a bookmark

when you make a new change, the bookmark will not automatically update to point to the new change

```
# if `-r` is not specified, it defaults to the working copy commit
jj bookmark set -r <revset> <bookmark-name>
```

conflicts

"different changes are made to the same part of a file"

- `jj` can record conflicts in commits (first-class conflicts)
- you do NOT need to resolve conflicts with `jj` if you don't want to
- conflict resolution can be delayed
- a whole bunch of other advantages with first-class conflicts:
<https://jj-vcs.github.io/jj/latest/conflicts/>

"okay this sounds good, how does it actually look like?"

Resolving conflicts

```
# brings up the builtin diff editor
jj resolve
```

operation log

a log of all operations that modified the repo

this is different from the `jj log` which shows the log of changes/revisions

📄 Viewing the operation log

```
jj op log
```

they seemed to be stored as files under: `.jj/repo/op_store/operations`

📄 Undo an operation

```
└ this is one of my favorite features ❤️
```

```
# undo the last operation
jj undo
```

git remotes

collaborating with others and storing your work on the ☁

■ Adding a remote

```
jj git remote add <name> <url>
```

■ Listing all remotes

```
jj git remote list
```

■ Pushing to remote

```
# if no remote is specified, it defaults to "origin"  
jj git push --remote <name>
```

git remotes

■ Fetching from remote

```
# if no remote is specified, it defaults to "origin"  
# if no branch is specified, it defaults to fetching all branches  
jj git fetch --remote <name> --branch <branch-name>
```

■ Tracking a remote branch

```
jj bookmark track <branch-name>@<remote-name>
```

- once you have tracked a remote branch, it becomes a tracked remote bookmark
- tracked remote bookmarks are associated with local bookmarks with the same name

final thoughts

- (as of 17/04/2025) does not support:
 - `git` submodules
 - `git-lfs`: large file support
- things are brewing: <https://jj-vcs.github.io/jj/latest/roadmap/>
- i am excited about and want to use more of:
 - `jj run`: "run a command across a set of revisions"
 - `jj fix`: "update files with formatting fixes or other changes"
- `jj --help` to explore more commands to try out!

helpful links

- <https://jj-vcs.github.io/jj/latest/>
- Martin's talk at GitMerge 2024: <https://www.youtube.com/watch?v=LV0JzI8IcCY>
- <https://blog.chay.dev/basic-jj-workflows/>
- <https://steveklabnik.github.io/jujutsu-tutorial/>
- Official Jujutsu discord: <https://discord.gg/dkmfj3aGQN>

thank you!