How I use git worktrees

Mar 05, 2024

My favorite feature of git is one that not enough people know about: *worktrees*.

Worktrees allow you to store branches of your repository in separate directories.

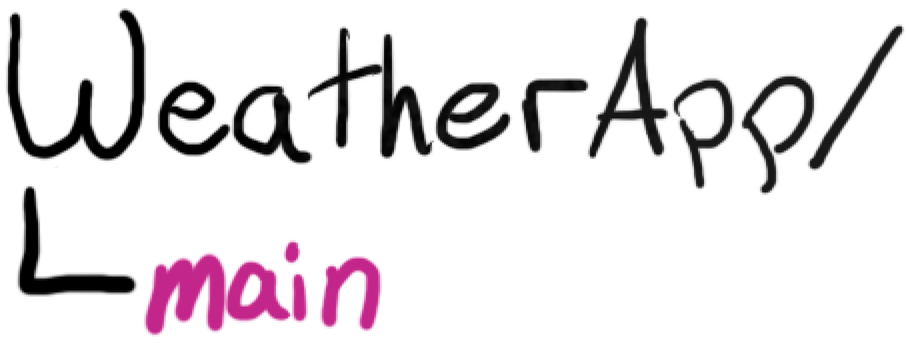
This means you can switch branches by changing directory, instead of switching between branches in the same directory with `git checkout` or `git switch`.

I've never seen anybody describe using worktrees quite the way I do, so I thought I'd write out how I like to work with them.

Project structure

When I create or clone a project, I create a project directory and then clone the `main` branch into a subfolder of that directory.

If I were working on a weather app, the directory structure after this might look like:



Then I would go into the `main` directory and start working on the project as normal.

Since I rarely want to work on the main branch directly, I generally start by creating a worktree for whatever my next task is.

The git command to make a worktree for a branch called `update-node-deps`, assuming I'm in the `main` directory, would be `git worktree add ../update-node-deps update-node-deps`.

Unfortunately, the UI of the `git worktree add` command leaves a lot to be desired, so I've become dependent on my own `worktree` command that wraps it.

Improving the UI

There are some small annoyances my `worktree` script handles for me

- You need to type the branch name twice; once to tell git where to put the directory and again to select the branch
    - `worktree` makes a sensible directory name from the branch name so you only have to type it once
- If you want to create a new branch, you have to add a `-b` argument to `worktree add`
    - `worktree` checks for an existing branch, creates a tracking worktree if it's found, and otherwise assumes you want a new branch
- After you create a worktree, you have to `cd` into it
    - `worktree` will change you into your new directory automatically if you source it
        - That means I usually call it like `. worktree new-feature-branch`, so that I'm brought right to the directory

And one big one that would be a dealbreaker for me.

git worktree doesn't copy untracked files

If your directory has a `node_modules` folder, or you use direnv so it has a `.envrc` file, or you have any other type of local files that don't live in the git repo, you might find the bare `git worktree` command annoying.

On my large work repository, `npm install` takes almost two full minutes; if I had to `npm install` every time I made a worktree, I'd be much less likely to use it.

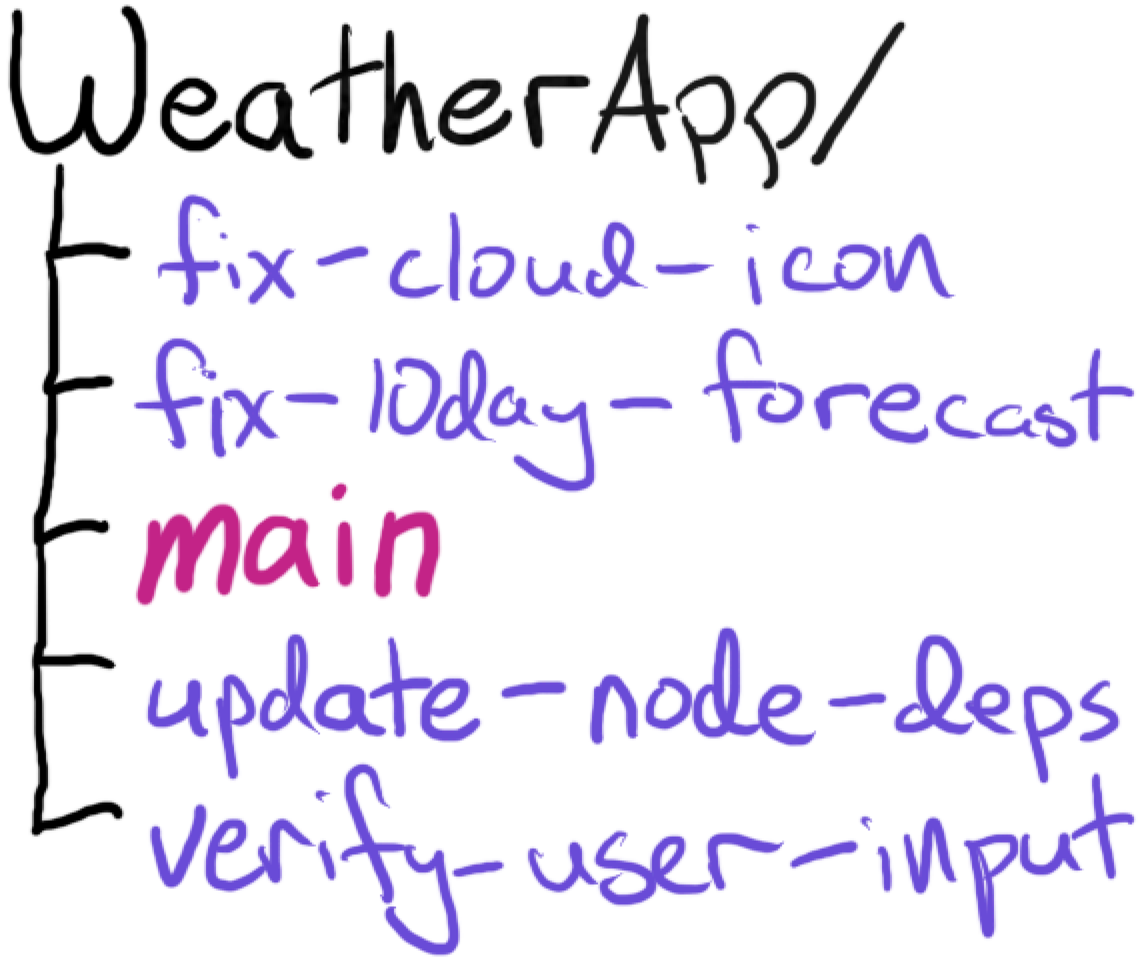
If you're a javascript developer, you may have felt the pain of `node_modules` even when using normal git branches; if you create a new branch, remove a package, then switch back to the main branch, you'll be unable to run any code that depends on that package until you `npm install` again.

The solution that I've come up with is that for every untracked file and directory that I care about, my `worktree` script creates a copy-on-write copy of the file or directory into the new worktree directory.

This means that when I run `. worktree new-feature`, I am dropped into the `new-feature` branch, with all my tools ready to work and able to focus on the new feature rather than on setting up my environment.

Why I like to work this way

After I've been working in my project for a while, its directory structure might look something like this, where each directory is a worktree:



What I love about this is that I can work on a feature branch like `verify-user-input`, drop that work to fix a bug in another directory, file a PR, then go right back to working on my feature branch while I wait for code review.

If somebody later on leaves a comment on my bug fix PR that I want to address, I can jump right back into the directory containing its worktree and make the change without having to pay any cost for switching tasks - I can just change directories instead of modifying one all-singing-all-dancing working directory.

In this way, I have branches that are a bit less stateful than the normal way of using git.

I find it especially useful that I *always* have a checkout of the `main` branch handy. If I'm working on a feature, and I want to see what was in the main branch before I changed it, I can grep against the `main` directory instead of remembering how to pull that info out of git. If somebody asks a question about what the app does, I can go look at the code in `main` without having to throw away my working state - I just change directories.

Removing worktrees

To remove a worktree, you can use `git worktree remove <branch>`, or just delete the directory containing the worktree and use `git worktree prune`. I have an `rmtree` script as well, but it's much simpler than `worktree`.

The only tricky bit I know with removing worktrees is that removing a worktree won't remove a branch; you've also got to use `git branch -D <branch>` if you want to get rid of it.

I don't expect you to use my scripts

I haven't written this post because I want to advertise my `worktree` and `rmtree` scripts; I wrote them for my specific workflow and it's unlikely to be a perfect fit for you.

However, I do think that git worktrees are pretty useful and that you might want to look into using them in your own workflow.

Backlinks:

- git worktrees step-by-step

↑ up