Context Switching in Git

Olaf Alders

September 2020

Problem:

- You are working in branch `feature-X`
- You now discover that you also have to solve an unrelated problem
- You need to do this work in branch `feature-Y`

Solution #1

- *Halt work on branch feature-X*
- `git stash`
- `git checkout -b feature-Y origin/master`
- *hack hack hack…*
- `git checkout feature-X` or `git switch -`
- `git stash pop`

Pros:

- Easy workflow for simple changes
- Good for small repositories

Cons:

- You can only have one workspace at any given time
- You can't always pop the stash cleanly, depending on the state of your repository
- The more branches you have, the more complicated your stash becomes

Solution #1 (redux)

- *Halt work on branch feature-X*
- `git add -u` (adds only modified and deleted files)
- `git commit -m "WIP"`
- `git checkout -b feature-Y origin/master`
- *hack hack hack…*
- `git checkout feature-X` or `git switch -`
- `git reset HEAD~1`

Pros:

- Easy workflow for simple changes
- Good for small repositories

Cons:

- You can only have one workspace at any given time
- WIP commits can sneak into your final branch if you're not vigilant

Solution #2

- Clone a new repository for each new feature branch

Pros:

- Work in multiple workspaces simultaneously
- No need for `git stash` or WIP commits

Cons:

- Can use a lot of disk space.
  - Shallow clones can help here
- Your clones will be agnostic about each other.
  - You'll need to track where your clones live.
  - There are some tools to help with this, like `App::GitGot`.
- `git` hooks will need to be set up on each clone.

Solution #3

- `git add worktree`
- *Manage multiple working trees attached to the same repository*

What is a worktree?

- The files in the repository which belong to your project
- Think of it as a workspace

You get your first worktree for free

```
$ mkdir /tmp/foo && cd /tmp/foo
$ git init
$ git worktree list
/tmp  0000000 [master]
```

Adding a worktree

```
$ git clone https://github.com/oalders/http-browserdetect.git
$ cd http-browserdetect/
$ git worktree list
/Users/olaf/http-browserdetect  90772ae [master]

$ git worktree add ~/trees/oalders/feature-X -b oalders/feature-X origin/master
$ git worktree add ~/trees/oalders/feature-Y -b oalders/feature-Y e9df3c555e96b3f1

$ git worktree list
/Users/olaf/http-browserdetect       90772ae [master]
/Users/olaf/trees/oalders/feature-X  90772ae [oalders/feature-X]
/Users/olaf/trees/oalders/feature-Y  e9df3c5 [oalders/feature-Y]
```

How much disk space is used?

```
$ du -sh /Users/olaf/http-browserdetect
2.9M

$ du -sh /Users/olaf/trees/oalders/feature-X
1.0M
```

Pros:

- Work in multiple workspaces simultaneously
- Avoid needing the stash
- `git` tracks all of your worktrees
- No need to set up `git` hooks
- Faster than `git clone`
- More efficient use of disk space
- A bit like a shallow clone, but you still have your history

Cons:

- Yet another thing to remember

Cleaning up

```
git worktree remove /Users/olaf/trees/oalders/feature-X
```

Or

```
rm -rf /Users/olaf/trees/oalders/feature-X
```

- You can then clean up remaining files via `git worktree prune`
- Or, do nothing now and this will happen at some point in the future via `git gc`

Notable notes

- Removing a worktree does not delete the branch
- You can switch branches within a worktree
- You cannot check out the same branch in multiple worktrees
- Like many other `git` commands, `git worktree` commands need to be run from inside a repository
- You can have many worktrees at once
- Create your worktrees from the same local checkout, or they will be agnostic about each other

git rev-parse

- To find the root of the parent repository
  - `git rev-parse --git-common-dir`
- To find the root of the repository we're in
  - `git rev-parse --show-toplevel`