Git Worktrees

Embracing the ADD

Patrick Reagan

Senior Developer @ TestFit

TESTFIT

Scenario

Working on a long-running feature, need quick bugfix

Stash and Switch

```
git add . && git stash save "WIP" && git checkout -b bugfix master
```

Cons

- **Mixed State** — unstaged or untracked files can leak between contexts
- **Context Switching** — easy to forget a stash or apply the wrong one
- **Single Working Directory** — can't work on both things simultaneously

Scenario

Enabling multi-agent orchestration

Multiple Clones

```
cd ../ && \
  git clone git@github.com:user/project.git work-queue && \
  git clone git@github.com:user/project.git experiments
```

Cons

- **Disk Usage** — full copy of repo history and any binary assets
- **Duplicate Maintenance** — hooks, remotes, and config drift between clones
- **Speed** — full clone + re-setup takes time

Alternative: Git Worktrees

What is a Git Worktree?

A worktree is an ***additional working directory*** linked to the same repository.

- First officially introduced in v2.5
- Each worktree has its own checked-out branch and files
- All worktrees **share** the same `.git` history, objects, and remotes
- No extra clone — history is shared, files are a local copy (not a network transfer)

```
my-project/           # main worktree  (branch: main)
my-project-feature/   # linked worktree (branch: feature-x)
my-project-hotfix/    # linked worktree (branch: hotfix-123)
```

> Think of it as multiple workbenches in the same workshop — different projects in progress, same set of tools.

Worktrees are *lighter*, *faster*, and keep everything *in sync*.

Essential Commands

```
# Create a worktree for an existing branch
git worktree add ../my-project-feature feature-branch

# Create a worktree with a new branch
git worktree add -b new-feature ../my-project-new-feature

# List all worktrees
git worktree list

# Remove a worktree (after merging / cleanup)
git worktree remove ../my-project-feature

# Prune stale worktree metadata
git worktree prune
```

Shortcomings of `git worktree add`

- Only checks out **tracked files**.
- Untracked files that your project depends on are left behind:
  - `.env` / `.env.local`
  - Private keys and credentials
  - Vendor config, local overrides

Your new worktree is *available* but does not *run*.

Improving Worktree Management

I use a script that can be installed as an alias:

```
# ~/.gitconfig
[alias]
  worktree-copy = \\!~/.local/bin/git-worktree-copy
```

Configure with a `.worktree-copy` file in your main checkout:

```
# Files to be copied to target worktree
.env
config/master.key

# Commands to run after worktree add
! bin/setup
```

How I've Been Using Worktrees

```
mkdir my-project
git clone git@github.com:user/my-project.git my-project/master
cd my-project/master

git worktree-copy ../work-queue  # <-- Manage Jira work queue
git worktree-copy ../experiments # <-- "Risky" experiments
git worktree-copy ../performance # <-- Fix N+1 w/ Sentry MCP
git worktree-copy ../bug-fixes   # <-- Space for quick fixes
```

Demo?