A simpler way to manage your dotfiles

May 5, 2018

2018 · Tips

Like most folks, I use git to manage my dotfiles. This lets me have a versioned backup for my configurations, and if something breaks (and it does often for me) I can revert to a working configuration fairly easily. For a long time, I've followed the normal path of having a `dotfiles` folder and a script that symlinks into the files in it from my `$HOME`. Recently, I came across this thread in HackerNews and it literally blew my mind. In this post, I would like to share this very elegant solution that avoids the need for any symlinking.

The key idea is really simple: make `$HOME` the git `work-tree`. The normal way of doing this would be to do a `git init` in your `$HOME`, but that would totally mess up git commands if you have other repositories in your `$HOME` (also, you probably don't want your entire `$HOME` in a git repo). So, instead, we will create a dummy folder and initialize a **bare** repository (essentially a git repo with **no** working directory) in there. All git commands will be run with our dummy as the git directory, but `$HOME` as the work directory.

First Time Setup

Setting this method up the first time is really easy. First, let's create our bare repository. I chose to name my placeholder `.dotfiles` (duh!)

```
mkdir $HOME/.dotfiles
git init --bare $HOME/.dotfiles
```

Now for the fun part. We will make an alias for running git commands in our `.dotfiles` repository. I'm calling my alias `dotfiles`:

```
alias dotfiles='/usr/local/bin/git --git-dir=$HOME/.dotfiles/ --work-tree=$HOME'
```

Add this alias to your `.bashrc` or `.zshrc`. From now on, any git operation you would like to do in the `.dotfiles` repository can be done by the `dotfiles` alias. The cool thing is that you can run `dotfiles` from anywhere.

Let's add a remote, and also set status not to show untracked files:

```
dotfiles config --local status.showUntrackedFiles no
dotfiles remote add origin git@github.com:anandpiyer/.dotfiles.git
```

You'll need to change the remote URL to your git repo. Now, you can easily add the config files you want to be in version control from where they are supposed to be, commit and push. For example, to add `tmux` config files, I'll do:

```
cd $HOME
dotfiles add .tmux.conf
dotfiles commit -m "Add .tmux.conf"
dotfiles push
```

Setting Up a New Machine

To set up a new machine to use your version controlled config files, all you need to do is to clone the repository on your new machine telling git that it is a bare repository:

```
git clone --separate-git-dir=$HOME/.dotfiles https://github.com/anandpiyer/.dotfiles.git ~
```

However, some programs create default config files, so this might fail if git finds an existing config file in your `$HOME`. In that case, a simple solution is to clone to a temporary directory, and then delete it once you are done:

```
git clone --separate-git-dir=$HOME/.dotfiles https://github.com/anandpiyer/.dotfiles.git t
rsync --recursive --verbose --exclude '.git' tmpdotfiles/ $HOME/
rm -r tmpdotfiles
```

There you go. No symlink mess.

My dotfiles are here for reference.