Managing Dotfiles with YADM: The Easy Way

Julius Follow
Nov 14, 2019 · 3 min read ★

Bootstrap DNA, photo by mira66

The thing I love most about Unix-systems is being able to keep my configuration files for all my systems available at all times. I can configure VSCode or Vim once be forever happy with my editor. There are many ways of doing this, and the most basic would be to have a git repository at your home directory. I started with this, and it works *fine,* I guess.

The problems with this approach are the following:

- Every directory below your home directory counts as being in this git repository. So programs with git integration will try to parse all files inside
- git will list all files in directories below /home/ as untracked
- Your configuration will be the same on all systems, which can pose issues if you have setups with system-specific configurations

Both of these problems get solved by YADM — Yet Another Dotfiles Manager.
A lovely wrapper for git which brings:

- Alternate files
- Encryption (which I don't use, I keep my confidential data in my password manager)
- Bootstrapping

YADM can do everything that git can — and more.
Also, since the YADM repo gets stored in a .yadm instead of a .git folder, it already solves our first problem. Easy!

Git solves our next problem natively.
We have to use a .gitignore file in our home directory.
I ignore all files in all directories by adding /** and retroactively readd all folders which contain files I need added (or I force add them).

If you want to see how I do this in detail, this is my dotfiles repository.

Alternate files

At work, I have my Arch Linux installation inside of VirtualBox. Thus I want to start the program "VBoxClient-all" at startup, which allows proper clipboard interaction with my Windows host.
My .xinitrc looks like this:

```
exec VBoxClient-all
exec i3
```

I want a different .xinitrc on my other devices, so Alternate Files to the rescue!
Alternate File syntax works like the following:

```
//default file
.xinitrc##
//use this on all Linux devices
.xinitrc##Linux
//use this on all Linux devices with hostname archpc
.xinitrc##Linux.archpc
//use this on all Linux devices with hostname archpc and the username
julius
.xinitc##Linux.archpc.julius
```

YADM will generate symlinks to the appropriate file after doing a `yadm pull`. But you can also trigger this manually with `yadm alt`.

Bootstrapping

For most of my initial installation, I use the automation framework Ansible. It downloads all packages and libraries I need to get started fast.
It is powerful and versatile, thus too much for this article; I will write about it at a later date.
If you still want to have a look, you can see my Ansible playbooks here.

But this represents another problem if I want to my installation to be fire and forget with only one instruction to execute. I need to have ansible and an ansible library already installed.
YADM can help us out again! You can create a script called `~/.yadm/bootstrap` that executes immediately after cloning your repo.

Now the only bottleneck is getting YADM installed:

```
curl -fLo /usr/local/bin/yadm
https://github.com/TheLocehiliosan/yadm/raw/master/yadm && chmod a+x
/usr/local/bin/yadm
```

Afterward, I do a `yadm clone` of my dotfiles repository, and after a few minutes, I get a working machine!

For reference, this is what my bootstrapping-script looks like:

```
#!/bin/sh

# Install ansible aur library
git clone https://github.com/kewlfft/ansible-aur.git
~/.ansible/plugins/modules/aur

# Install ansible
sudo pacman – noconfirm -S ansible

# Execute base install
ansible-playbook ~/deployment/base-install.yml

if [ “$HOSTNAME” = “Cellaris” ]
then
 ansible-playbook ~/deployment/home-install.yml
elif [ “$HOSTNAME” = “Mobilus” ]
then
 ansible-playbook ~/deployment/notebook-install.yml
elif [ “$HOSTNAME” = “workarch” ]
then
 ansible-playbook ~/deployment/work-install.yml
fi
```

You can use any executable file, but don’t forget the shebang!

Dotfiles Linux Bootstrapping Git Github

About Help Legal