4 minutes

A way to organize your Bash aliases on multiple hosts

If you use multiple systems, you probably have a method of syncing your dotfiles between them. But different systems also have different needs.

When we talk about Bash, not properly managing those differences could mean having tons of useless aliases polluting your environment, just because they are wanted on other machines. Or a very cluttered `~/.bash_aliases` .

On this post, I'll talk about my approach to handle different Bash aliases on multiple machines.

Some solutions

A first try to handle the differences between hosts is, obviously, throwing some conditionals to your `~/.bash_aliases` . For example, if you have aliases to `cd` to important directories, you could condition them to the existence of the folder. I do this to some extent and helps to keep the environment clean.

However, it only works to a certain point. When you deal with systems that serve totally different purposes, say a Raspberry Pi server vs. your laptop, this method becomes cumbersome. Your `~/.bash_aliases` will be filled with conditionals, some maybe with difficult logic, and badly organized. Not to mention giant.

An alternative to this is to have multiple copies of your aliases file with the needed differences, and point each machine to the correspondent copy. The advantage here is that you have more readable files, but in the other hand, tons of duplication and management hell. Not nice either.

So, let's talk about my approach.

My approach

```
# ~/.bashrc

# Alias definitions.
if [ -f ~/.config/bash_aliases/$HOSTNAME ]; then
  . ~/.config/bash_aliases/$HOSTNAME
else
  . ~/.config/bash_aliases/default
fi
```

```
~/.config/bash_aliases
├── base        # Common aliases
├── default     # Aliases for generic hosts
├── foo         # Aliases for host foo
└── bar         # Aliases for host bar
```

First of all, we create a directory to store the aliases.

On `~/.bashrc`, we search that path for a file named as the current hostname. If it doesn't exist, we source `default`.

We also create a file called `base`, which may be sourced by each of the other files.

Now let's dive into that directory.

The base file

In `base`, you place the aliases that you want to be available for every host.

In my case, this include things as simple as `alias ll="ls -l"` to more complex ones, and it is my biggest file.

I don't limit this file to which software I have available on all machines. My mindset here is to include everything that doesn't target any system specifically.

For example, mine has an alias *(actually a function)* to easily call snapper, although I don't have it installed on all my machines. It is there because, if someday I install snapper on whatever system, I'll probably want that alias.

So if something depends on the presence of a software, and not on which host I am, it probably belongs here on `base`, conditioned to the existence of the binaries on the system. Easy enough.

The host-specific files

The next step is to create a file for each system needing their own aliases. This file is named after their hostname.

Here you source `base`, and then set the additional aliases.

Other thing you might do here is to override some of the `base` aliases. *"Why would I do that?"* You might ask.

A example is if one of your systems uses Busybox instead of Coreutils. The former is a smaller version of the tools we are used to in Linux, and, as such, often has different (and less) command-line options. This breaks some of my aliases, such as `rm -vI`.

I solve it by overriding those aliases in my host-specific file for a "good enough" version of them. I think this makes more sense than removing the affected aliases from `base`.

Back to your host-specific files, they will be something like this:

```
# foo
# Aliases for a host called foo

# Source base
. ~/.config/bash_aliases/base

# Specific and overriding aliases for foo
...
...
```

The default file

Lastly, for the systems in which you don't need specific configuration, you have a fallback file called `default` .

Here you do the same thing as in the host-specific files, i.e., source `base` and then set or override aliases. The difference is that you are targeting the devices that won't match a specific hostname.

An alternative to write this file is to point those systems directly to `base` . For me, though, it makes sense to have a layer on top of `base` , to give me some flexibility.

Conclusion

I came with this approach to my aliases and it's serving me very well. I don't have duplicated code, my files are very readable and easily manageable. I also have a fair amount of flexibility. You can see it in action here.

I can think of many ways this solution could be modified and extended to better suit other particular cases. Feel free to adapt it to your needs.

#sysadmin #Bash #how-to

2018-08-27

SEE ALSO

Mount Borg backups with ease

Smart morning radio with Raspberry Pi, cron and MPD

Web server and Pi Hole - a how-to

Running a web server while using Pi Hole

0 Comments rafaelc.org 1 Login

Recommend Tweet Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON RAFAELC.ORG

Running a web server while using Pi Hole

2 comments • a year ago

João Clobocar — Very useful article.

A way to organize your Bash aliases on multiple hosts

2 comments • a year ago

vencetti — Nice. I would adda subcategory around purpose