Managing Dotfiles Across Machines

DOIN Managing Dotfiles with Chezmoi

Configuring dotfiles manager with chezmoi
Reference:

- Chezmoi
- Managing dotfiles with chezmoi @Blog

Chezmoi Concepts

> Chezmoi computes the target state for the current machine and then updates the destination directory

- The *destination directory* is the directory that `chezmoi` manages, usually the home directory, ~
- The *destination state* is the current state of all the targets in the destination directory
- The *Source state* declares the desired state of your home directory, including templates that use machine-specific data. It contains only regular files and directories
- The *source directory* is where `chezmoi` stores the source state. By default it is `~/.local/share/chezmoi`
- The *config file* contains machine specific data. By default it is `~/.config/chezmoi/chezmoi.toml`
- The *target state* is the desired state of the destination directory. It is computed from the source state, the config file, and the destination state. The target state includes regular files and directories, and may also include symbolic links, script to be run, and targets to be removed
- The *working tree* is the git working tree. Normally it is the same as the source directory, but an be a parent of the source directory

1. Managing Outside of HOME

   Managing files that live outside of the HOME *dir*
   Reference: Manage Files outside of HOME @Doc

2. Manage Machine-to-Machine Differences

   > My primary goal of using `chezmoi` is to be able to manage config files across multiple machines. I want to keep much configuration the same across these machines, but also need machine-specific configurations for emails address and credentials. Chezmoi achieves this by using `text/template` for the source state where needed

   a) Chezmoi Template

> Templates are used to change the contents of a file depending on the environment. For Example, using hostname of the machine to create different configuations of different machines

When reading files from the *source state*, `chezmoi` interprets them as a template if either the following is true:

- The file name has `.tmpl` suffix
- The file is in the `.chezmoitemplate` directory, or sub-directory

For example, Home `~/.gitconfig` on the personal machine might look like:

```
[user]
  email = "me@home.org"
```

Whereas on a work machine, it might be:

```
[user]
  email = "firstname.lastname@company.com"
```

Reference: Chezmoi templating @Doc

i. Create a Template

   To handle this, on each machine create a configuration file called `~/.config/chezmoi.toml` defining variables that might vary from machine to machine e.g. the home machine

   ```
   [data]
     email = "me@home.org"
   ```

   > [!TIP] If you intended to store private data *(e.g. access token)* in `~/.config/chezmoi/chezmoi.toml`, make sure it has permission `0600`. Variable names must start a letter and be followed by zero or more letters or digits

   Then, add `~/.gitconfig` to chezmoi using the `--template` flag to turn it into a template:

   ```
   chezmoi add --template ~/.config/
   ```

ii. Edit a Template

   Open the template *(which will be saved in the* `~/.local/share/chezmoi/dot_gitconfig.tmp` *)*:

   ```
   chezmoi edit ~/.gitconfig
   ```

Edit the file so it looks like:

```
[user]
  email = {{.email | quote}}
```

> [!TIP] For a full list of variable, run: `chezmoi data`

b) Interacting with Password-Managers

> [!NOTE] Chezmoi support functions to interact with password managers

Some password managers allow you to store complete files. The files can be retrieved with chezmoi's template functions. For example, if you have a file stored in `1Password` with the UUID `uuid` then you can retrieve it with the template

```
{{- onepasswordDocument "uuid" -}}
```

The − inside the brackets remove any whitespace before or after the template expression, which is useful if your editor has added any newlines

> [!NOTE] If, after executing the template, the file contents are empty, the target file will be removed. This can be used to ensure that files are only present on certain machines. If you want an empty file to be created anyway, you will need to five an `empty_` prefix

Visit: Password Manager Integration @Org

c) Ignore Files on Different Machines

> For coarser-gained control of files and entire directories managed on different machines, or to exclide certain files completely, you can create `.chezmoiignore` files in the source director.

Specify a list of pattern that chezmoi should ignore, and are interpreted as templates

```
README.md
{{- if ne .chezmoi.hostname "work-laptop" }}
.work # only manage .work on work-laptop
{{- end }}
```

> [!NOTE] The use of `ne` *(not equal)* is deliberate. What we want to achieve is "only install `.work` if hostname is `work-laptop`. But chezmoi installs everything by default, so we have to turn the logic around an instead write "ignore `.work` unless the hostname is `work-laptop`"

Reference: Ignore files on different machines @Doc

3. Password Manager Integration

   Reference: Chezmoi password manager integration @Doc