Home Manager Manual

Preface

This manual will eventually describe how to install, use, and extend Home Manager.

If you encounter problems then please reach out on the IRC channel #home-manager hosted by OFTC. If your problem is caused by a bug in Home Manager then it should be reported on the Home Manager issue tracker.

> ***Note*** *Commands prefixed with `#` have to be run as root, either requiring to login as root user or temporarily switching to it using `sudo` for example.*

Chapter 1. Installing Home Manager

Home Manager can be used in three primary ways:

1. Using the standalone `home-manager` tool. For platforms other than NixOS and Darwin, this is the only available choice. It is also recommended for people on NixOS or Darwin that want to manage their home directory independent of the system as a whole. See Section 1.1, “Standalone installation” for instructions on how to perform this installation.
2. As a module within a NixOS system configuration. This allows the user profiles to be built together with the system when running `nixos-rebuild`. See Section 1.2, “NixOS module” for a description of this setup.
3. As a module within a nix-darwin system configuration. This allows the user profiles to be built together with the system when running `darwin-rebuild`. See Section 1.3, “nix-darwin module” for a description of this setup.

1.1. Standalone installation

1. Make sure you have a working Nix installation. Specifically, make sure that your user is able to build and install Nix packages. For example, you should be able to successfully run a command like `nix-instantiate '<nixpkgs>' -A hello` without having to switch to the root user. For a multi-user install of Nix this means that your user must be covered by the `allowed-users` Nix option. On NixOS you can control this option using the `nix.allowedUsers` system option.

2. Add the Home Manager channel that you wish to follow. If you are following Nixpkgs master or an unstable channel then this is done by running

```
$ nix-channel --add https://github.com/nix-community/home-manager/archive/master.tar.gz home
$ nix-channel --update
```

   and if you follow a Nixpkgs version 21.05 channel, you can run

```
$ nix-channel --add https://github.com/nix-community/home-manager/archive/release-21.05.tar.
$ nix-channel --update
```

   On NixOS you may need to log out and back in for the channel to become available. On non-NixOS you may have to add

```
export NIX_PATH=$HOME/.nix-defexpr/channels${NIX_PATH:+:}$NIX_PATH
```

   to your shell (see nix#2033).

3. Run the Home Manager installation command and create the first Home Manager generation:

   ```
   $ nix-shell '<home-manager>' -A install
   ```

   Once finished, Home Manager should be active and available in your user environment.

4. If you do not plan on having Home Manager manage your shell configuration then you must source the

   ```
   $HOME/.nix-profile/etc/profile.d/hm-session-vars.sh
   ```

   file in your shell configuration. Alternatively source

   ```
   /etc/profiles/per-user/$USER/etc/profile.d/hm-session-vars.sh
   ```

   when managing home configuration together with system configuration.

   Unfortunately, we currently only support POSIX.2-like shells such as Bash or Z shell.

   For example, if you use Bash then add

   ```
   . "$HOME/.nix-profile/etc/profile.d/hm-session-vars.sh"
   ```

   to your `~/.profile` file.

If instead of using channels you want to run Home Manager from a Git checkout of the repository then you can use the `programs.home-manager.path` option to specify the absolute path to the repository.

1.2. NixOS module

Home Manager provides a NixOS module that allows you to prepare user environments directly from the system configuration file, which often is more convenient than using the `home-manager` tool. It also opens up additional possibilities, for example, to automatically configure user environments in NixOS declarative containers or on systems deployed through NixOps.

To make the NixOS module available for use you must `import` it into your system configuration. This is most conveniently done by adding a Home Manager channel. For example, if you are following Nixpkgs master or an unstable channel, you can run

```
# nix-channel --add https://github.com/nix-community/home-manager/archive/master.tar.gz home-mana
# nix-channel --update
```

and if you follow a Nixpkgs version 21.05 channel, you can run

```
# nix-channel --add https://github.com/nix-community/home-manager/archive/release-21.05.tar.gz hor
# nix-channel --update
```

It is then possible to add

```
imports = [ <home-manager/nixos> ];
```

to your system `configuration.nix` file, which will introduce a new NixOS option called `home-manager.users` whose type is an attribute set that maps user names to Home Manager configurations.

For example, a NixOS configuration may include the lines

```
users.users.eve.isNormalUser = true;
home-manager.users.eve = { pkgs, ... }: {
  home.packages = [ pkgs.atool pkgs.httpie ];
  programs.bash.enable = true;
};
```

and after a `nixos-rebuild switch` the user eve's environment should include a basic Bash configuration and the packages atool and httpie.

Note *By default packages will be installed to `$HOME/.nix-profile` but they can be installed to `/etc/profiles` if*

```
home-manager.useUserPackages = true;
```

is added to the system configuration. This is necessary if, for example, you wish to use `nixos-rebuild build-vm`. This option may become the default value in the future.

Note *By default, Home Manager uses a private `pkgs` instance that is configured via the `home-manager.users.<name>.nixpkgs` options. To instead use the global `pkgs` that is configured via the system level `nixpkgs` options, set*

```
home-manager.useGlobalPkgs = true;
```

This saves an extra Nixpkgs evaluation, adds consistency, and removes the dependency on `NIX_PATH`, which is otherwise used for importing Nixpkgs.

1.3. nix-darwin module

Home Manager provides a module that allows you to prepare user environments directly from the nix-darwin configuration file, which often is more convenient than using the `home-manager` tool.

To make the NixOS module available for use you must `import` it into your system configuration. This is most conveniently done by adding a Home Manager channel. For example, if you are following Nixpkgs master or an unstable channel, you can run

```
# nix-channel --add https://github.com/nix-community/home-manager/archive/master.tar.gz home-mana
# nix-channel --update
```

and if you follow a Nixpkgs version 21.05 channel, you can run

```
# nix-channel --add https://github.com/nix-community/home-manager/archive/release-21.05.tar.gz hor
# nix-channel --update
```

It is then possible to add

```
imports = [ <home-manager/nix-darwin> ];
```

to your nix-darwin `configuration.nix` file, which will introduce a new NixOS option called `home-manager` whose type is an attribute set that maps user names to Home Manager configurations.

For example, a nix-darwin configuration may include the lines

```
users.users.eve = {
  name = "eve";
  home = "/Users/eve";
}
home-manager.users.eve = { pkgs, ... }: {
  home.packages = [ pkgs.atool pkgs.httpie ];
  programs.bash.enable = true;
};
```

and after a `darwin-rebuild switch` the user eve’s environment should include a basic Bash configuration and the packages atool and httpie.

Note *By default user packages will not be ignored in favor of `environment.systemPackages`, but they will be intalled to `/etc/profiles/per-user/$USERNAME` if*

```
home-manager.useUserPackages = true;
```

is added to the nix-darwin configuration. This option may become the default value in the future.

Note *By default, Home Manager uses a private `pkgs` instance that is configured via the `home-manager.users.<name>.nixpkgs` options. To instead use the global `pkgs` that is configured via the system level `nixpkgs` options, set*

```
home-manager.useGlobalPkgs = true;
```

This saves an extra Nixpkgs evaluation, adds consistency, and removes the dependency on `NIX_PATH`, which is otherwise used for importing Nixpkgs.

Chapter 2. Writing Home Manager Modules

The module system in Home Manager is based entirely on the NixOS module system so we will here only highlight aspects that are specific for Home Manager. For information about the module system as such please refer to the Writing NixOS Modules chapter of the NixOS manual.

2.1. Option Types

Overall the basic option types are the same in Home Manager as NixOS. A few Home Manager options, however, make use of custom types that are worth describing in more detail. These are the option types `dagOf` and `gvariant` that are used, for example, by `programs.ssh.matchBlocks` and `dconf.settings`.

`hm.types.dagOf`

Options of this type have attribute sets as values where each member is a node in a directed acyclic graph (DAG). This allows the attribute set entries to express dependency relations among themselves. This can, for example, be used to control the order of match blocks in a OpenSSH client configuration or the order of activation script blocks in `home.activation`.

A number of functions are provided to create DAG nodes. The functions are shown below with examples using an option `foo.bar` of type `hm.types.dagOf types.int`.

`hm.dag.entryAnywhere (value: T)`

Indicates that `value` can be placed anywhere within the DAG. This is also the default for plain attribute set entries, that is

```
foo.bar = {
  a = hm.dag.entryAnywhere 0;
}
```

and

```
foo.bar = {
  a = 0;
}
```

are equivalent.

`hm.dag.entryAfter (afters: list string) (value: T)`

Indicates that `value` must be placed *after* each of the attribute names in the given list. For example

```
foo.bar = {
  a = 0;
  b = hm.dag.entryAfter [ "a" ] 1;
}
```

would place `b` after `a` in the graph.

`hm.dag.entryBefore (befores: list string) (value: T)`

Indicates that `value` must be placed *before* each of the attribute names in the given list. For example

```
foo.bar = {
  b = hm.dag.entryBefore [ "a" ] 1;
  a = 0;
}
```

would place `b` before `a` in the graph.

`hm.dag.entryBetween (befores: list string) (afters: list string) (value: T)`

Indicates that `value` must be placed *before* the attribute names in the first list and *after* the attribute names in the second list. For example

```
foo.bar = {
  a = 0;
  c = hm.dag.entryBetween [ "b" ] [ "a" ] 2;
  b = 1;
}
```

would place `c` before `b` and after `a` in the graph.

`hm.types.gvariant`

This type is useful for options representing GVariant values. The type accepts all primitive GVariant types as well as arrays and tuples. Dictionaries are not currently supported.

To create a GVariant value you can use a number of provided functions. Examples assume an option `foo.bar` of type `hm.types.gvariant`.

`hm.gvariant.mkBoolean (v: bool)`

Takes a Nix value `v` to a GVariant `boolean` value. Note, Nix booleans are automatically coerced using this function. That is,

```
foo.bar = hm.gvariant.mkBoolean true;
```

is equivalent to

```
foo.bar = true;
```

`hm.gvariant.mkString (v: string)`
Takes a Nix value `v` to a GVariant `string` value. Note, Nix strings are automatically coerced using this function. That is,

```
foo.bar = hm.gvariant.mkString "a string";
```

is equivalent to

```
foo.bar = "a string";
```

`hm.gvariant.mkObjectpath (v: string)`
Takes a Nix value `v` to a GVariant `objectpath` value.

`hm.gvariant.mkUchar (v: string)`
Takes a Nix value `v` to a GVariant `uchar` value.

`hm.gvariant.mkInt16 (v: int)`
Takes a Nix value `v` to a GVariant `int16` value.

`hm.gvariant.mkUint16 (v: int)`
Takes a Nix value `v` to a GVariant `uint16` value.

`hm.gvariant.mkInt32 (v: int)`
Takes a Nix value `v` to a GVariant `int32` value. Note, Nix integers are automatically coerced using this function. That is,

```
foo.bar = hm.gvariant.mkInt32 7;
```

is equivalent to

```
foo.bar = 7;
```

`hm.gvariant.mkUint32 (v: int)`
Takes a Nix value `v` to a GVariant `uint32` value.

`hm.gvariant.mkInt64 (v: int)`
Takes a Nix value `v` to a GVariant `int64` value.

`hm.gvariant.mkUint64 (v: int)`
Takes a Nix value `v` to a GVariant `uint64` value.

`hm.gvariant.mkDouble (v: double)`
Takes a Nix value `v` to a GVariant `double` value. Note, Nix floats are automatically coerced using this function. That is,

```
foo.bar = hm.gvariant.mkDouble 3.14;
```

is equivalent to

```
foo.bar = 3.14;
```

`hm.gvariant.mkArray type elements`
Builds a GVariant array containing the given list of elements, where each element is a GVariant value of the given type. The `type` value can be constructed using

- `hm.gvariant.type.string`
- `hm.gvariant.type.boolean`

- `hm.gvariant.type.uchar`
- `hm.gvariant.type.int16`
- `hm.gvariant.type.uint16`
- `hm.gvariant.type.int32`
- `hm.gvariant.type.uint32`
- `hm.gvariant.type.int64`
- `hm.gvariant.type.uint64`
- `hm.gvariant.type.double`
- `hm.gvariant.type.arrayOf type`
- `hm.gvariant.type.maybeOf type`
- `hm.gvariant.type.tupleOf types`

where `type` and `types` are themselves a type and list of types, respectively.

`hm.gvariant.mkEmptyArray type`
An alias of `hm.gvariant.mkArray type []`.

`hm.gvariant.mkNothing type`
Builds a GVariant maybe value whose (non-existent) element is of the given type. The `type` value is constructed as described for the `mkArray` function above.

`hm.gvariant.mkJust element`
Builds a GVariant maybe value containing the given GVariant element.

`hm.gvariant.mkTuple elements`
Builds a GVariant tuple containing the given list of elements, where each element is a GVariant value.

Chapter 3. Contributing

Contributions to Home Manager are very welcome. To make the process as smooth as possible for both you and the Home Manager maintainers we provide some guidelines that we ask you to follow. See Section 3.1, “Getting started” for information on how to set up a suitable development environment and Section 3.2, “Guidelines” for the actual guidelines.

This text is mainly directed at those who would like to make code contributions to Home Manager. If you just want to report a bug then first look among the already open issues, if you find one matching yours then feel free to comment on it to add any additional information you may have. If no matching issue exists then go to the new issue page and write a description of your problem. Include as much information as you can, ideally also include relevant excerpts from your Home Manager configuration.

3.1. Getting started

If you have not previously forked Home Manager then you need to do that first. Have a look at GitHub’s Fork a repo for instructions on how to do this.

Once you have a fork of Home Manager you should create a branch starting at the most recent `master` branch. Give your branch a reasonably descriptive name. Commit your changes to this branch and when you are happy with the result and it fulfills Section 3.2, “Guidelines” then push the branch to GitHub and create a pull request.

Assuming your clone is at `$HOME/devel/home-manager` then you can make the `home-manager` command use it by either

1. overriding the default path by using the `-I` command line option:

```
$ home-manager -I home-manager=$HOME/devel/home-manager
```

   or

2. changing the default path by ensuring your configuration includes

   ```
   programs.home-manager.enable = true;
   programs.home-manager.path = "$HOME/devel/home-manager";
   ```

   and running `home-manager switch` to activate the change. Afterwards, `home-manager build` and `home-manager switch` will use your cloned repository.

The first option is good if you only temporarily want to use your clone.

3.2. Guidelines

If your contribution satisfy the following rules then there is a good chance it will be merged without too much trouble. The rules are enforced by the Home Manager maintainers and to a lesser extent the Home Manager CI system.

If you are uncertain how these rules affect the change you would like to make then feel free to start a discussion in the #home-manager IRC channel, ideally before you start developing.

3.2.1. Maintain backward compatibility

Your contribution should not cause another user’s existing configuration to break unless there is a very good reason and the change should be announced to the user through an assertion or similar.

Remember that Home Manager is used in many different environments and you should consider how your change may effect others. For example,

- Does your change work for people that do not use NixOS? Consider other GNU/Linux distributions and macOS.
- Does your change work for people whose configuration is built on one system and deployed on another system?

3.2.2. Keep forward compatibility in mind

The master branch of Home Manager tracks the unstable channel of Nixpkgs, which may update package versions at any time. It is therefore important to consider how a package update may affect your code and try to reduce the risk of breakage.

The most effective way to reduce this risk is to follow the advice in Section 3.2.3, “Add only valuable options”.

3.2.3. Add only valuable options

When creating a new module it is tempting to include every option supported by the software. This is *strongly* discouraged. Providing many options increases maintenance burden and risk of breakage considerably. This is why only the most important software options should be modeled explicitly. Less important options should be expressible through an `extraConfig` escape hatch.

A good rule of thumb for the first implementation of a module is to only add explicit options for those settings that absolutely must be set for the software to function correctly. It follows that a module for software that provides sensible default values for all settings would require no explicit options at all.

If the software uses a structured configuration format like a JSON, YAML, INI, TOML, or even a plain list of key/value pairs then consider using a `settings` option as described in Nix RFC 42.

3.2.4. Add relevant tests

If at all possible, make sure to add new tests and expand existing tests so that your change will keep working in the future. See Section 3.6, "Tests" for more information about the Home Manager test suite.

All contributed code *must* pass the test suite.

3.2.5. Add relevant documentation

Many code changes require changing the documentation as well. Module options should be documented with DocBook. See DocBook rocks! for a quick introduction and DocBook 5: The Definitive Guide for in-depth information of DocBook. Home Manager is itself documented using a combination of DocBook and AsciiDoc. All text is hosted in Home Manager's Git repository.

The HTML version of the manual containing both the module option descriptions and the documentation of Home Manager can be generated and opened by typing the following in a shell within a clone of the Home Manager Git repository:

```
$ nix-build -A docs.html
$ xdg-open ./result/share/doc/home-manager/index.html
```

When you have made changes to a module, it is a good idea to check that the man page version of the module options looks good:

```
$ nix-build -A docs.manPages
$ man ./result/share/man/man5/home-configuration.nix.5
```

3.2.6. Add yourself as a module maintainer

Every new module *must* include a named maintainer using the `meta.maintainers` attribute. If you are a user of a module that currently lacks a maintainer then please consider adopting it.

If you are present in the NixOS maintainer list then you can use that entry. If you are not then you can add yourself to `modules/lib/maintainers.nix` in the Home Manager project.

Also add yourself to `.github/CODEOWNERS` as owner of the associated module files, including the test files. You will then be automatically added as a reviewer on any new pull request that touches your files.

Maintainers are encouraged to join the IRC channel and participate when they have opportunity.

3.2.7. Format your code

Make sure your code is formatted as described in Section 3.4, "Code Style". To maintain consistency throughout the project you are encouraged to browse through existing code and adopt its style also in new code.

3.2.8. Format your commit messages

Similar to Section 3.2.7, “Format your code” we encourage a consistent commit message format as described in Section 3.3, “Commits”.

3.2.9. Format your news entries

If your contribution includes a change that should be communicated to users of Home Manager then you can add a news entry. The entry must be formatted as described in Section 3.5, “News”.

When new modules are added a news entry should be included but you do not need to create this entry manually. The merging maintainer will create the entry for you. This is to reduce the risk of merge conflicts.

3.2.10. Use conditional modules and news

Home Manager includes a number of modules that are only usable on some of the supported platforms. The most common example of platform specific modules are those that define systemd user services, which only works on Linux systems.

If you add a module that is platform specific then make sure to include a condition in the `loadModule` function call. This will make the module accessible only on systems where the condition evaluates to `true`.

Similarly, if you are adding a news entry then it should be shown only to users that may find it relevant, see Section 3.5, “News” for a description of conditional news.

3.2.11. Mind the license

The Home Manager project is covered by the MIT license and we can only accept contributions that fall under this license, or are licensed in a compatible way. When you contribute self written code and documentation it is assumed that you are doing so under the MIT license.

A potential gotcha with respect to licensing are option descriptions. Often it is convenient to copy from the upstream software documentation. When this is done it is important to verify that the license of the upstream documentation allows redistribution under the terms of the MIT license.

3.3. Commits

The commits in your pull request should be reasonably self-contained, that is, each commit should make sense in isolation. In particular, you will be asked to amend any commit that introduces syntax errors or similar problems even if they are fixed in a later commit.

The commit messages should follow the seven rules. We also ask you to include the affected code component or module in the first line. That is, a commit message should follow the template

```
{component}: {description}

{long description}
```

where `{component}` refers to the code component (or module) your change affects, `{description}` is a very brief description of your change, and `{long description}` is an optional clarifying description. As a rare exception, if there is no clear component, or your change affects many components, then the `{component}` part is optional. See Example 3.1, “Compliant commit message” for a commit message that fulfills these requirements.

Example 3.1. Compliant commit message

The commit 69f8e47e9e74c8d3d060ca22e18246b7f7d988ef contains the commit message

```
starship: allow running in Emacs if vterm is used

The vterm buffer is backed by libvterm and can handle Starship prompts
without issues.
```

which ticks all the boxes necessary to be accepted in Home Manager.

Finally, when adding a new module, say `programs/foo.nix`, we use the fixed commit format `foo: add module`. You can, of course, still include a long description if you wish.

3.4. Code Style

The code in Home Manager is formatted by the nixfmt tool and the formatting is checked in the pull request tests. Run the `format` tool inside the project repository before submitting your pull request.

Keep lines at a reasonable width, ideally 80 characters or less. This also applies to string literals.

We prefer `lowerCamelCase` for variable and attribute names with the accepted exception of variables directly referencing packages in Nixpkgs which use a hyphenated style. For example, the Home Manager option `services.gpg-agent.enableSshSupport` references the `gpg-agent` package in Nixpkgs.

3.5. News

Home Manager includes a system for presenting news to the user. When making a change you, therefore, have the option to also include an associated news entry. In general, a news entry should only be added for truly noteworthy news. For example, a bug fix or new option does generally not need a news entry.

If you do have a change worthy of a news entry then please add one in `news.nix` but you should follow some basic guidelines:

- The entry timestamp should be in ISO-8601 format having "+00:00" as time zone. For example, "2017-09-13T17:10:14+00:00". A suitable timestamp can be produced by the command

  ```
  $ date --iso-8601=second --universal
  ```

- The entry condition should be as specific as possible. For example, if you are changing or deprecating a specific option then you could restrict the news to those users who actually use this option.
- Wrap the news message so that it will fit in the typical terminal, that is, at most 80

characters wide. Ideally a bit less.

- Unlike commit messages, news will be read without any connection to the Home Manager source code. It is therefore important to make the message understandable in isolation and to those who do not have knowledge of the Home Manager internals. To this end it should be written in more descriptive, prose like way.

- If you refer to an option then write its full attribute path. That is, instead of writing

  ```
  The option 'foo' has been deprecated, please use 'bar' instead.
  ```

  it should read

  ```
  The option 'services.myservice.foo' has been deprecated, please
  use 'services.myservice.bar' instead.
  ```

- A new module, say `foo.nix`, should always include a news entry that has a message along the lines of

  ```
  A new module is available: 'services.foo'.
  ```

  If the module is platform specific, e.g., a service module using systemd, then a condition like

  ```
  condition = hostPlatform.isLinux;
  ```

  should be added. If you contribute a module then you don’t need to add this entry, the merger will create an entry for you.

3.6. Tests

Home Manager includes a basic test suite and it is highly recommended to include at least one test when adding a module. Tests are typically in the form of "golden tests" where, for example, a generated configuration file is compared to a known correct file.

It is relatively easy to create tests by modeling the existing tests, found in the `tests` project directory.

The full Home Manager test suite can be run by executing

```
$ nix-shell --pure tests -A run.all
```

in the project root. List all test cases through

```
$ nix-shell --pure tests -A list
```

and run an individual test, for example `alacritty-empty-settings`, through

```
$ nix-shell --pure tests -A run.alacritty-empty-settings
```

Chapter 4. Frequently Asked Questions (FAQ)

4.1. Why is there a collision error when switching generation?

Home Manager currently installs packages into the user environment, precisely as if the packages were installed through `nix-env --install`. This means that you will get a collision error if your Home Manager configuration attempts to install a package that you already have installed manually, that is, packages that shows up when you run `nix-env --query`.

For example, imagine you have the `hello` package installed in your environment

```
$ nix-env --query
hello-2.10
```

and your Home Manager configuration contains

```
home.packages = [ pkgs.hello ];
```

Then attempting to switch to this configuration will result in an error similar to

```
$ home-manager switch
these derivations will be built:
  /nix/store/xg69wsnd1rp8xgs9qfsjal017nf0ldhm-home-manager-path.drv
[…]
Activating installPackages
replacing old ‘home-manager-path’
installing ‘home-manager-path’
building path(s) ‘/nix/store/b5c0asjz9f06l5219812w6k39ifr49jj-user-environment’
Wide character in die at /nix/store/64jc9gd2rkbgdb4yjx3nrgc91bpjj5ky-buildenv.pl line 79.
collision between ‘/nix/store/fmwa4axzghz11cnln5absh31nbhs9lq1-home-manager-path/bin/hello’ and ‘
builder for ‘/nix/store/b37x3s7pzxbasfqhaca5dqbf3pjjw0ip-user-environment.drv’ failed with exit c
error: build of ‘/nix/store/b37x3s7pzxbasfqhaca5dqbf3pjjw0ip-user-environment.drv’ failed
```

The solution is typically to uninstall the package from the environment using `nix-env --uninstall` and reattempt the Home Manager generation switch.

You could also opt to unistall *all* of the packages from your profile with `nix-env --uninstall '*'`.

4.2. Why are the session variables not set?

Home Manager is only able to set session variables automatically if it manages your Bash or Z shell configuration. If you don’t want to let Home Manager manage your shell then you will have to manually source the `~/.nix-profile/etc/profile.d/hm-session-vars.sh` file in an appropriate way. In Bash and Z shell this can be done by adding

```
. "$HOME/.nix-profile/etc/profile.d/hm-session-vars.sh"
```

to your `.profile` and `.zshrc` files, respectively. The `hm-session-vars.sh` file should work in most Bourne-like shells.

4.3. How to set up a configuration for multiple users/machines?

A typical way to prepare a repository of configurations for multiple logins and machines is to prepare one "top-level" file for each unique combination.

For example, if you have two machines, called "kronos" and "rhea" on which you want to configure your user "jane" then you could create the files

- `kronos-jane.nix`,
- `rhea-jane.nix`, and
- `common.nix`

in your repository. On the kronos and rhea machines you can then make `~jane/.config/nixpkgs/home.nix` be a symbolic link to the corresponding file in your configuration repository.

The `kronos-jane.nix` and `rhea-jane.nix` files follow the format

```
{ ... }:

{
  imports = [ ./common.nix ];

  # Various options that are specific for this machine/user.
}
```

while the `common.nix` file contains configuration shared across the two logins. Of course, instead of just a single `common.nix` file you can have multiple ones, even one per program or service.

You can get some inspiration from the Post your home-manager home.nix file! Reddit thread.

4.4. Why do I get an error message about `ca.desrt.dconf`?

You are most likely trying to configure the GTK or Gnome Terminal but the DBus session is not aware of the dconf service. The full error you might get is

```
error: GDBus.Error:org.freedesktop.DBus.Error.ServiceUnknown: The name ca.desrt.dconf was not pro
```

The solution on NixOS is to add

```
services.dbus.packages = with pkgs; [ gnome.dconf ];
```

to your system configuration.

4.5. How do I install packages from Nixpkgs unstable?

If you are using a stable version of Nixpkgs but would like to install some particular packages from Nixpkgs unstable – or some other channel – then you can import the unstable Nixpkgs and refer to its packages within your configuration. Something like

```
{ pkgs, config, ... }:

let

  pkgsUnstable = import <nixpkgs-unstable> {};

in

{
  home.packages = [
    pkgsUnstable.foo
  ];

  # …
}
```

should work provided you have a Nix channel called `nixpkgs-unstable`.

You can add the `nixpkgs-unstable` channel by running

```
# nix-channel --add https://nixos.org/channels/nixpkgs-unstable nixpkgs-unstable
# nix-channel --update
```

Note, the package will not be affected by any package overrides, overlays, etc.

4.6. How do I override the package used by a module?

By default Home Manager will install the package provided by your chosen `nixpkgs` channel but occasionally you might end up needing to change this package. This can typically be done in two ways.

1. If the module provides a `package` option, such as `programs.beets.package`, then this is the recommended way to perform the override. For example,

   ```
   programs.beets.package = pkgs.beets.override { enableCheck = true; };
   ```

2. If no `package` option is available then you can typically override the relevant package using an overlay.

   For example, if you want to use the `programs.skim` module but use the `skim` package from Nixpkgs unstable, then a configuration like

   ```
   { pkgs, config, ... }:

   let

     pkgsUnstable = import <nixpkgs-unstable> {};

   in

   {
     programs.skim.enable = true;

     nixpkgs.overlays = [
       (self: super: {
         skim = pkgsUnstable.skim;
       })
     ];

     # …
   }
   ```

   should work OK.