# Overlays

In the previous section, we learned about overriding derivations by `pkgs.xxx.override { ... }` or `pkgs.xxx.overrideAttrs (finalAttrs: previousAttrs: { ... });`. However, this approach will generate a new derivation and doesn't modify the original derivation in `pkgs` instance. If the derivation you want to override is also used by other Nix packages, they will still use the unmodified derivation.

To globally modify derivations in the default nixpkgs instance, Nix provides a feature called "overlays".

In traditional Nix environments, overlays can be configured globally using the `~/.config/nixpkgs/overlays.nix` or `~/.config/nixpkgs/overlays/*.nix` files. However, with Flakes feature, to ensure system reproducibility, overlays cannot rely on configurations outside of the Git repository.

When using `flake.nix` to configure NixOS, both Home Manager and NixOS provide the `nixpkgs.overlays` option to define overlays. You can refer to the following documentation for more details:

- Home Manager docs - `nixpkgs.overlays`
- Nixpkgs source code - `nixpkgs.overlays`

Let's take a look at an example module that loads overlays. This module can be used as a Home Manager module or a NixOS module, as the definitions are the same:

nix

```nix
# ./overlays/default.nix
{ config, pkgs, lib, ... }:

{
  nixpkgs.overlays = [
    # Overlay 1: Use `self` and `super` to express
    # the inheritance relationship
    (self: super: {
      google-chrome = super.google-chrome.override {
        commandLineArgs =
          "--proxy-server='https=127.0.0.1:3128;http=127.0.0.1:3128'";
      };
    })
```

```
    # Overlay 2: Use `final` and `prev` to express
    # the relationship between the new and the old
    (final: prev: {
      steam = prev.steam.override {
        extraPkgs = pkgs: with pkgs; [
          keyutils
          libkrb5
          libpng
          libpulseaudio
          libvorbis
          stdenv.cc.cc.lib
          xorg.libXcursor
          xorg.libXi
          xorg.libXinerama
          xorg.libXScrnSaver
        ];
        extraProfile = "export GDK_SCALE=2";
      };
    })

    # Overlay 3: Define overlays in other files
    # The content of ./overlays/overlay3/default.nix is the same as above:
    # `(final: prev: { xxx = prev.xxx.override { ... }; })`
    (import ./overlay3)
  ];
}
```

In the above example, we define three overlays.

1. Overlay 1 modifies the `google-chrome` derivation by adding a command-line argument for a proxy server.
2. Overlay 2 modifies the `steam` derivation by adding extra packages and environment variables.
3. Overlay 3 is defined in a separate file `./overlays/overlay3/default.nix`.

One example of importing the above configuration as a NixOS module is as follows:

nix

```
# ./flake.nix
{
  inputs = {
    # ...
```

```
  };

  outputs = inputs@{ nixpkgs, ... }: {
    nixosConfigurations = {
      my-nixos = nixpkgs.lib.nixosSystem {
        system = "x86_64-linux";
        modules = [
          ./configuration.nix

          # import the module that contains overlays
          (import ./overlays)
        ];
      };
    };
  };
}
```

This is just an example. Please write your own overlays according to your needs.

---

## Multiple nixpkgs Instances with different Overlays

The `nixpkgs.overlays = [...];` mentioned above directly modifies the global nixpkgs instance `pkgs`. If your overlays make changes to some low-level packages, it might impact other modules. One downside is an increase in local compilation (due to cache invalidation), and there might also be functionality issues with the affected packages.

If you wish to utilize overlays only in a specific location without affecting the default nixpkgs instance, you can instantiate a new nixpkgs instance and apply your overlays to it. We will discuss how to do this in the next section The Ingenious Uses of Multiple nixpkgs Instances.

---

## References

- Chapter 3. Overlays - nixpkgs Manual

Loading comments...