Auto formatting using `treefmt-nix`

treefmt provides an interface to run multiple code formatters at once, so you don't have to run them manually for each file type in your development project.

Writing the Nix to configure treefmt in your project

Add treefmt and flake-root to your inputs

The `flake-root` flake-parts module is needed to find the root of your project based on the presence of a file, by default it is `flake.nix`.

```
{
  # Inside `inputs`
  treefmt-nix.url = "github:numtide/treefmt-nix";
  flake-root.url = "github:srid/flake-root";
}
```

Import `flakeModule` output of treefmt and flake-root

```
{
  # Inside outputs' `flake-parts.lib.mkFlake`
  imports = [
    inputs.treefmt-nix.flakeModule
    inputs.flake-root.flakeModule
  ];
}
```

Configure your formatter

To actually enable the individual formatters you want to configure treefmt. The example configuration below only consists of formatters required by a haskell project using nix. Refer to treefmt-doc for more formatters.

```
{
  # Inside mkFlake's `perSystem`
  treefmt.config = {
    inherit (config.flake-root) projectRootFile;
    # This is the default, and can be overriden.
    package = pkgs.treefmt;
    # formats .hs files (fourmolu is also available)
    programs.ormolu.enable = true;
    # formats .nix files
    programs.nixpkgs-fmt.enable = true;
    # formats .cabal files
    programs.cabal-fmt.enable = false;
    # Suggests improvements for your code in .hs files
    programs.hlint.enable = false;
  };
}
```

Add treefmt to your devShell

Finally, add the resulting treefmt wrapper (`build.wrapper`) to your devShell. We also add the individual formatters (`build.programs`) to the devShell, so that they can be used directly in text editors and IDEs.

```
{
  # Inside mkFlake's `perSystem`
  haskellProjects.default = {
    devShell.tools = _: {
      treefmt = config.treefmt.build.wrapper;
    } // config.treefmt.build.programs;
  };
}
```

Flake check

The `treefmt-nix` flake module automatically adds a flake check that can be evaluated to make sure that the project is already autoformatted.

Tips

Exclude folders

If there are folders where you wouldn't want to run the formatter on, use the following:

```
# Inside mkFlake's `perSystem.treefmt.config`
settings.formatter.<formatter-name>.excludes = [ "./foo/*" ];
```

Use a different package for formatter

The package shipped with the current nixpkgs might not be the desired one, follow the snippet below to override the package (assuming `nixpkgs-21_11` is present in your flake's inputs).

```
# Inside mkFlake's `perSystem.treefmt.config`
programs.ormolu.package = nixpkgs-21_11.haskellPackages.ormolu;
```

The same can be applied to other formatters.

Pass additional parameters to your formatter

You might want to change a certain behaviour of your formatter by overriding by passing the input to the executable. The following example shows how to pass `ghc-opt` to ormolu:

```
# Inside mkFlake's `perSystem.treefmt.config`
settings.formatter.ormolu = {
  options = [
    "--ghc-opt"
    "-XTypeApplications"
  ];
};
```

Ormolu requires this `ghc-opt` because unlike a lot of language extensions which are enabled by default, there are some which aren't. These can be found using `ormolu --manual-exts`.

Example

- Sample treefmt config for your haskell project

Upcoming

- `treefmt` will provide a pre-commit mode to disable commit if formatting checks fail. This is tracked here: https://github.com/numtide/treefmt/issues/78

Links to this page

Rust FFI in Haskell

> You can find the template at https://github.com/shivaraj-bh/haskell-rust-ffi-template. This template also includes formatting setup with treefmt-nix and VSCode integration.