Git

Git is the most commonly used version control system for software development.

Declarative configuration

Git can be declaratively configured in Nix via home-manager. Here is an example:

```
{
  programs.git = {
    enable = true;
    userName = "john";
    userEmail = "john@doe.com";
    ignores = [ "*~" "*.swp" ];
    extraConfig = {
      init.defaultBranch = "master";
    };
  };
}
```

Git related pages

Folgezettel children

> Separate Git “profiles”

Links to this page

Use a local directory as flake input

> A Flake URL can not only be Git repositories. They can also refer to local paths. If you have two projects `~/code/foo` and `~/code/bar`, and `bar` depends on `foo`, you can use the following `flake.nix` in `bar` to have it refer to the local `foo` project:

Separate Git “profiles”

> You want to override Git config (such as commit author email) for only certain repos, such as those under a certain folder. This is useful when dealing with corporate policies, which often block commit pushes that doesn't comfort to certain standards, such as using work email address in the commit email. Those using Bitbucket's Control Freak may be familiar with this error throw in response `git push`:

Install NixOS with Flake configuration on Git

> First we need to install Git:

> Let's store our whole configuration in a Git repository.

> This tutorial will walk you through the steps necessary to install NixOS, enable flakes while tracking the resulting system configuration in a Git repository.

Convert `configuration.nix` to be a flake

> Progress, but we hit another error—Nix understandably cannot write to root-owned directory (it tries to create the `flake.lock` file). One way to resolve this is to move the whole configuration to our home directory, which would also prepare the ground for storing it in Git. We will do this in the next section.