A Portable Nix-shell Shebang

Updated November 8, 2023

All of my personal scripts with more than a few dependencies start with the Nix-shell shebang: `#!/usr/bin/env nix-shell`. Regardless of what language they are written in.

That only works if nix is installed, so it is difficult to share with others who may not use nix.

If I want the script to be shared with other people, I instead use something like this:

```
#!/usr/bin/env bash

if [ -z "$INSIDE_NIX_RANDOMSTRING" ] && command -v nix-shell &> /dev/null; then
  # Install dependencies with nix
  INSIDE_NIX_RANDOMSTRING=1 nix-shell \
    -I nixpkgs=https://github.com/NixOS/nixpkgs/archive/bceb3bff2ee78424c1073d0b4676858265f926d1.tar.gz \
    --packages jq go \
    --run ""$0" "$@""
  exit $?
fi

go
jq --version
```

Update: The above script breaks with more than 1 argument to the command. The flake version of the command is necessary if you need to take in more than 1 argument.

```
#! /usr/bin/env bash

if [ -z "$INSIDE_NIX_RANDOMSTRING" ] && command -v nix-shell &> /dev/null; then
  # Install dependencies with nix
  INSIDE_NIX_RANDOMSTRING=1 nix shell \
      nixpkgs#jq \
      nixpkgs#go \
      --command "$0" "$@"
  exit $?
fi

go
jq --version
```

That snippet checks for the presence of nix-shell, and then reruns the script under nix-shell after installing the necessary dependencies. The `-I` argument is optional, and can be used to pin nixpkgs to a specific version (using a commit hash).

Portability

For my personal scripts I just use the normal nix-shell shebang line, `#!/usr/bin/env nix-shell`, but I can do that because *I* am the only one that typically uses them, and I use NixOS. If you want to write scripts to share with others, adding Nix as a dependency isn't always going to be a good idea.

Instead, what I do above is use it to install dependencies *if it's available*. That way you can have all the non-Nix folks deal with their dependency nightmares themselves, and it will just work for all the people who have adopted Nix.

A similar approach can be adopted if you want to use Guix instead of Nix. In fact, I'm sure the snippet above could be extended to check for the presence of either. If you know Guix, send me an email with the updated snippet and I'll include it here.

When Not To Use It

One scenario where I don't use this technique is in development repos. Repos tend to end up with a decent amount of helper scripts, and it's tempting to make use of nix-shell to install dependencies for all of those.

I don't think you should bother to do that.

Instead dependencies should be managed at the repo level, by using a `flake.nix` or a `shell.nix` in the root directory. That keeps things a bit simpler and ensures that all the dependencies are installed in one place, and with the same versions.