LET’S GO NIX

Scott Windsor
Devops PDX

GOALS

- Beginner-friendly intro to nix
- Introduce concepts and language
- Get you excited about nix!

EXPECTATIONS

- Familiarity with command line & shell
- Understand at least one programming language like `javascript`

BACKGROUND

- Been a developer for > 20 years
- Dealt with countless build/run dependency issues
- Have worked heavily with modernizing legacy systems
- Usually work in small agile teams

THE QUEST FOR REPRODUCIBLE DEVELOPMENT ENVIRONMENTS

- automake (and porting)
- ports / macports
- anisible
- chef
- puppet
- Language managers: (rvm, virtualenv, nvm)
- Docker

BUT THEY ALWAYS FALL SHORT

- System architecture woes (again!)
- Personal machine drift
- Working with multiple projects across teams

HOW NIX IS DIFFERENT

- Saves packages in isolation - `/nix/store`
- Builds packages with a functional language
- Allows you to link to system, user, or shell environments these packages

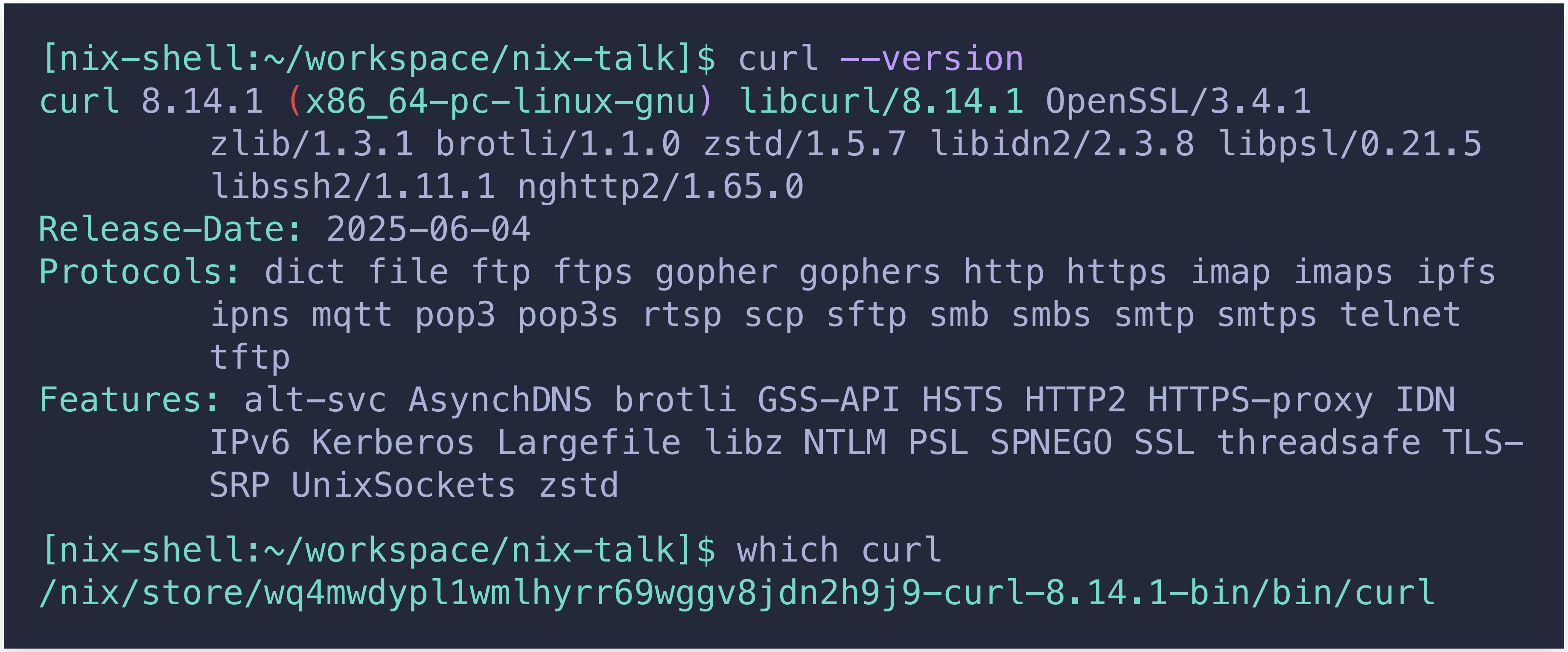
FIRST STEPS - INSTALLING

```
$ curl -fsSL https://install.determinate.systems/nix | sh -s -- install
        --determinate
```

CREATING SHELL WITH CURL

```
$ nix-shell -p curl
these 9 paths will be fetched (0.99 MiB download, 4.14 MiB unpacked):
  /nix/store/9v2s5rbf6pb77vhagihl7dicpqkg3614-c-ares-1.34.5
  /nix/store/wznrhnlrvamvihizpnizjfh5hs55z98n-curl-8.14.1-dev
  /nix/store/48wm9h7wf8ds4wkwgzzcqfrp7l722dm8-krb5-1.21.3-dev
  /nix/store/i1j8dzchkv1p59bqzrr15585s8s4zvx0-libev-4.33
  /nix/store/kss6l466kl66x2bqzy9rv7nz4pjgc55c-libidn2-2.3.8-bin
  /nix/store/9j67k582x3vgcijfiyralx5bj1b33gdg-libidn2-2.3.8-dev
  /nix/store/y37r7yjyvnzzd648lpdgflynfj55hpns-libpsl-0.21.5-dev
  /nix/store/rq4pnjcjrkic79kxc2fq0g7hp78s8ypv-nghttp2-1.65.0
  /nix/store/9pn6y4zlszr9w26rg2h52l3sd0wvzjvd-nghttp2-1.65.0-dev
copying path '/nix/store/48wm9h7wf8ds4wkwgzzcqfrp7l722dm8-krb5-1.21.3-
        dev' from 'https://cache.nixos.org'...
copying path '/nix/store/9v2s5rbf6pb77vhagihl7dicpqkg3614-c-ares-1.34.5'
        from 'https://cache.nixos.org'...
copying path '/nix/store/y37r7yjyvnzzd648lpdgflynfj55hpns-libpsl-0.21.5-
        dev' from 'https://cache.nixos.org'...
```

CREATING SHELL WITH CURL



```
[nix-shell:~/workspace/nix-talk]$ curl --version
curl 8.14.1 (x86_64-pc-linux-gnu) libcurl/8.14.1 OpenSSL/3.4.1
        zlib/1.3.1 brotli/1.1.0 zstd/1.5.7 libidn2/2.3.8 libpsl/0.21.5
        libssh2/1.11.1 nghttp2/1.65.0
Release-Date: 2025-06-04
Protocols: dict file ftp ftps gopher gophers http https imap imaps ipfs
        ipns mqtt pop3 pop3s rtsp scp sftp smb smbs smtp smtps telnet
        tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy IDN
        IPv6 Kerberos Largefile libz NTLM PSL SPNEGO SSL threadsafe TLS-
        SRP UnixSockets zstd

[nix-shell:~/workspace/nix-talk]$ which curl
/nix/store/wq4mwdypl1wmlhyrr69wggv8jdn2h9j9-curl-8.14.1-bin/bin/curl
```

SHOWING RUNTIME DEPENDENCIES (LINUX)

```
[nix-shell:~/workspace/nix-talk]$ ldd $(which curl)
        linux-vdso.so.1 (0x00007f0e95cfa000)
        libcurl.so.4 => /nix/store/frlckg2m2sf0gs8g5pqkryddbpy6qcz1-
        curl-8.14.1/lib/libcurl.so.4 (0x00007f0e95c12000)
        libnghttp2.so.14 => /nix/store/gwwbjkdd3rghq7x74561agq08f4jmh7p-
        nghttp2-1.65.0-lib/lib/libnghttp2.so.14 (0x00007f0e95be3000)
        libidn2.so.0 => /nix/store/ncdwsrgq6n6161l433m4x34057zq0hhf-
        libidn2-2.3.8/lib/libidn2.so.0 (0x00007f0e95bb2000)
        libssh2.so.1 => /nix/store/y6w3rwlym1mlpcysn6l7r5vbdmf9irf1-
        libssh2-1.11.1/lib/libssh2.so.1 (0x00007f0e95b67000)
        libpsl.so.5 => /nix/store/31fknicrbimbw6ivnxly9pdabsqqglk5-
        libpsl-0.21.5/lib/libpsl.so.5 (0x00007f0e95b53000)
        libssl.so.3 => /nix/store/byx7ahs386pskh8d5sdkrkpscfz9yyjp-
        openssl-3.4.1/lib/libssl.so.3 (0x00007f0e95a47000)
        libcrypto.so.3 => /nix/store/byx7ahs386pskh8d5sdkrkpscfz9yyjp-
        openssl-3.4.1/lib/libcrypto.so.3 (0x00007f0e95400000)
        libgssapi_krb5.so.2 =>
```

SHOWING RUNTIME DEPENDENCIES (MACOS)

```
[nix-shell:~]$ otool -L $(which curl)
/nix/store/bblr8ccnd4baxm4cf7g1igfz6ya8v93m-curl-8.14.1-bin/bin/curl:
        /nix/store/6l3i3d58xr1r4qv49v1ln8wf309sb15x-curl-
        8.14.1/lib/libcurl.4.dylib (compatibility version 13.0.0,
        current version 13.0.0)
        /nix/store/jkdx2fgyj2lhma8xydrp6xkqgv13a00g-nghttp2-1.65.0-
        lib/lib/libnghttp2.14.dylib (compatibility version 43.0.0,
        current version 43.4.0)
        /nix/store/8jfck34h4ayxg41lylz1aayjjjmy2qhw-libidn2-
        2.3.8/lib/libidn2.0.dylib (compatibility version 5.0.0, current
        version 5.0.0)
        /nix/store/4kk9xgcdga33k9h371p81svlam1aqa07-libssh2-
        1.11.1/lib/libssh2.1.dylib (compatibility version 2.0.0, current
        version 2.1.0)
        /nix/store/lvg9zfb2ig76821dmmpcdlb9xd6md1g5-libpsl-
        0.21.5/lib/libpsl.5.dylib (compatibility version 9.0.0, current
        version 9.5.0)
```

EXITING THE SHELL

```
[nix-shell:~/workspace/nix-talk]$ exit
exit
```

CREATING A FLAKE

```
$ mkdir -p ~/workspace/nix-first-steps
$ cd ~/workspace/nix-first-steps
$ git init
$ nix flake init templates#utils-generic
```

LOADING THE FLAKE

`.envrc`:

```
use flake
```

OUR FIRST FLAKE

`flake.nix`:

```
{
  inputs = {
    utils.url = "github:numtide/flake-utils";
  };
  outputs = { self, nixpkgs, utils }: utils.lib.eachDefaultSystem
        (system:
    let
      pkgs = nixpkgs.legacyPackages.${system};
    in
    {
      devShell = pkgs.mkShell {
        buildInputs = with pkgs; [
        ];
      };
    }
  );
}
```

NIX THE LANGUAGE

```
$ nix repl
Nix 2.29.0
Type :? for help.
nix-repl> 1 + 2
3
```

```
$ nix eval --expr '1+2'
3
```

```
$ echo "1+2" >> math.nix
$ nix eval -f math.nix
3
```

COMMENTS

```
# Text that follows a `#` is a comment!
```

STRINGS

```
# This is a string
"foo"
```

MULTI-LINE STRINGS

```
# This is a multi–line string

''I'm a mult–line
string
''
```

NUMBERS

```
# This is a number
5
```

LISTS

```
# This is a list of numbers and strings
[ 1 2 "foo" ]
```

ATTRIBUTE SETS

```
# This is an empty "attribute set", which is also like a dictionary or
        hash in other languages.

{}
```

ATTRIBUTE SETS

```nix
# attribute sets can assign attributes

{
  foo = "bar";
  baz = "buzz";
}
```

ATTRIBUTE SETS

```
# You can make nested attribute sets

{
  foo = {
    bar = "baz";
  };
}
```

ATTRIBUTE SETS

```
# Or assign them with a "." for shorthand
{ foo.bar = "baz"; }
```

INPUTS EXAMPLE

```nix
# This is our inputs example
{
  inputs = {
    utils.url = "github:numtide/flake-utils";
  };
}
```

ATTRIBUTE SETS

```nix
# This is our inputs example, but shorter
{
  inputs.utils.url = "github:numtide/flake-utils";
}
```

FUNCTIONS

```
# a `:` denotes a function with arguments on left and function body on
        the right

x: x + 1
```

FUNCTIONS

```
# You can call a function by applying an argument, but you may need to
        wrap in parenthesis
(x: x + 1) 2
```

FUNCTIONS

```
# Most of the time you will see attributes as the function arguments
{ a, b }: a + b
```

FUNCTIONS

```
# When calling this you pass an attribute set

({ a, b }: a + b) {
  a = 2;
  b = 3;
}
```

CURRYING

```
# Functions can also be `curried`
a: b: a + b
```

CURRYING

```
# Again, using parenthesis to apply
(a: b: a + b) 2 3
```

CURRYING

```
# Again, using parenthesis to apply
((a: b: a + b) 2) 3
```

OUTPUTS EXAMPLE

```
# Now we can understand the output line a bit better (omitting the
        `system` body for now)...

{
  outputs =
    {
      self,
      nixpkgs,
      utils,
    }:
    utils.lib.eachDefaultSystem (system: { });
}
```

LET BLOCKS

```
# `let` blocks allow you to assign values you can use inside an
        `in`attribute set

let
  a = 10;
in
{
  x = a;
}
```

INTERPOLATION

```
# Sometimes you might want to refer to interpolated values for attribute
        keys
# We can use `${}` for this

let
  a = "x";
in
{
  ${a} = 10;
}
```

SYSTEM EXAMPLE

```nix
# This is how `${system}` being used in our flake. Here's smaller
        example that applies both
# functions.

(
  system:
  { nixpkgs }:
  let
    pkgs = nixpkgs.legacyPackages.${system};
  in
  pkgs
)
  "linux"
  { nixpkgs.legacyPackages.linux = "awesome"; }
```

INHERIT

```
# Assigning a value to it's name is so common that there's a shorthand
        with `inherit`

let
  a = 10;
  b = 12;
  c = 5;
in
{
  a = a;
  b = b;
  c = c;
}
```

INHERIT

```nix
# Assigning a value to it's name is so common that there's a shorthand with `inherit`

let
  a = 10;
  b = 12;
  c = 5;
in
{
  inherit a b c;
}
```

ALMOST THERE!

```
# We have one last thing to learn before we understand all of our flake!
# You can do it!
```

WITH

```nix
# Sometimes repeating keys can get a bit cumbersome

let
  x = {
    a = 1;
    b = 3;
    c = 4;
  };
in
[
  x.a
  x.b
  x.c
]
```

WITH

```
# We can use `with` to automatically scope all of the attributes in x

let
  x = {
    a = 1;
    b = 3;
    c = 4;
  };
in
with x;
[
  a
  b
  c
]
```

NIX LANGUAGE COMPLETE

```
# You did it! Great job!
```

REVIEWING OUR FLAKE

`flake.nix`:

```nix
{
  inputs = {
    utils.url = "github:numtide/flake-utils";
  };
  outputs = { self, nixpkgs, utils }: utils.lib.eachDefaultSystem
        (system:
    let
      pkgs = nixpkgs.legacyPackages.${system};
    in
    {
      devShell = pkgs.mkShell {
        buildInputs = with pkgs; [
        ];
      };
    }
  );
}
```

MOVING NIXPKGS TO STABLE

We add an input for nixpkgs to `25.05` (overriding default)

```
inputs = {
  nixpkgs.url = "github:nixos/nixpkgs/nixos-25.05";
  utils.url = "github:numtide/flake-utils";
};
```

ADDING PACKAGES TO OUR DEVSHELL

These are for our rust app, but you can find more at ___________________

```
devShell = pkgs.mkShell {
        buildInputs = with pkgs; [
          cargo
          rustc
          rust-analyzer
          rustfmt
        ];
      };
    }
```

ENTER THE DEVSHELL

We can use `nix develop` to get to the shell. `.#` is a reference to the current flake.

```
$ nix develop .#
(nix:nix-shell-env) bash-5.2$ rustc --version
rustc 1.86.0 (05f9846f8 2025-03-31) (built from a source tarball)
(nix:nix-shell-env) bash-5.2$ cargo --version
cargo 1.86.0 (adf9b6ad1 2025-02-28)
(nix:nix-shell-env) bash-5.2$ exit
exit
```

DIRENV MAKES THIS BETTER

If you don’t already have direnv installed, you can install to your profile via nix.

```
$ nix profile install nixpkgs#direnv
$ echo 'eval "$(direnv hook bash)"' >> ~/.bashrc
$ source ~/.bashrc
```

DIRENV MAKES THIS BETTER

Now the flake is evaluated when we enter the directory

```
$ direnv allow
direnv: loading ~/workspace/nix-first-steps/.envrc
direnv: using flake
warning: Git tree '/Users/scott/workspace/nix-first-steps' has
         uncommitted changes
direnv: export +AR +AS +CC +CONFIG_SHELL +CXX +DEVELOPER_DIR +HOST_PATH
        +IN_NIX_SHELL +LD +LD_DYLD_PATH +MACOSX_DEPLOYMENT_TARGET
        +NIX_APPLE_SDK_VERSION +NIX_BINTOOLS
        +NIX_BINTOOLS_WRAPPER_TARGET_HOST_arm64_apple_darwin
        +NIX_BUILD_CORES +NIX_BUILD_TOP +NIX_CC
        +NIX_CC_WRAPPER_TARGET_HOST_arm64_apple_darwin
        +NIX_CFLAGS_COMPILE +NIX_DONT_SET_RPATH
        +NIX_DONT_SET_RPATH_FOR_BUILD +NIX_ENFORCE_NO_NATIVE
        +NIX_HARDENING_ENABLE +NIX_IGNORE_LD_THROUGH_GCC +NIX_LDFLAGS
        +NIX_NO_SELF_RPATH +NIX_STORE +NM +OBJCOPY +OBJDUMP +PATH_LOCALE
        +RANLIB +SDKROOT +SIZE +SOURCE_DATE_EPOCH +STRINGS +STRIP +TEMP
        +TEMPDIR +TMP +ZERO_AR_DATE +__darwinAllowLocalNetworking
```

DIRENV MAKES THIS BETTER

Now our packages are just in our path!

```
$ rustc --version
rustc 1.86.0 (05f9846f8 2025-03-31) (built from a source tarball)
```

BUILDING OUR APP

Now that we have our environment, we can build our app.

```
$ cd ~/workspace/nix-first-steps
$ cargo new hello-nix
$ cd hello-nix
```

BUILDING OUR APP

open up `hello-nix/src/main.rs` and change to the following:

```rust
fn main() {
    println!("Hello from nix!");
}
```

BUILDING OUR APP

We can make sure this builds, tests, and runs.

```
$ cargo build
   Compiling hello-nix v0.1.0 (/Users/scott/workspace/nix-first-
        steps/hello-nix)
     Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.77s

$ cargo test
   Compiling hello-nix v0.1.0 (/Users/scott/workspace/nix-first-
        steps/hello-nix)
     Finished `test` profile [unoptimized + debuginfo] target(s) in
        0.11s
       Running unittests src/main.rs (target/debug/deps/hello_nix-
        c7e1c6d541507f78)

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered
        out; finished in 0.00s

$ cargo run
```

BUILDING WITH NIX

Let’s make a new file, `default.nix` and put it in the `hello-nix` directory.

```nix
{ pkgs ? import <nixpkgs> { } }:
pkgs.rustPlatform.buildRustPackage {
  pname = "hello-nix";
  version = "0.0.1";
  cargoLock.lockFile = ./Cargo.lock;
  src = pkgs.lib.cleanSource ./.;
}
```

NEW SYNTAX: ?

```
# The `?` allows us to have optional values in attribute sets. This
        comes in handy for optional
# arguments in functions.

{ foo ? "foo" }: foo
```

NEW SYNTAX FOR NIX

```
# you can either apply without that name set.
({ foo ? "foo" }: foo) {}
```

NEW SYNTAX FOR NIX

```
# or with it
({ foo ? "foo" }: foo) { foo = "bar"; }
```

NEW SYNTAX FOR NIX

```nix
# `import` is a special builtin function for loading code.
# `./filename` is path variable relative by current directory.
# We can use this to import our new `default.nix` file

import ./default.nix
```

NEW SYNTAX FOR NIX

```
# <nixpkgs> is a special value that resolves lookup paths for $NIX_PATH
# This can be used to dynamically load whichever location nix is set to
# That means that the argument to our function takes an attribute set
        with
# an options pkgs that defaults to the imported version of `nixpkgs` if
        passed in.

{ pkgs ? import <nixpkgs> { } }: {}
```

NEW SYNTAX FOR NIX

back to our `default.nix`

```
{ pkgs ? import <nixpkgs> { } }:
pkgs.rustPlatform.buildRustPackage {
  pname = "hello-nix";
  version = "0.0.1";
  cargoLock.lockFile = ./Cargo.lock;
  src = pkgs.lib.cleanSource ./.;
}
```

BUILDING OUR PACKAGE

We can use the `nix build` command to build

```
$ nix build -f default.nix
```

BUILDING OUR PACKAGE

And see the results…

```
$ ls -la result
lrwxr-xr-x 1 scott staff 59 Jun 29 17:26 result ->
        /nix/store/rj2wf0vgsgbsadlad6nxssnb4lhqvjw1-hello-nix-0.0.1
$ ./result/bin/hello-nix
Hello from nix!
$ rm result
```

ADDING PACKAGE TO OUR FLAKE

back up to our `flake.nix`, we provide this as the `default` package

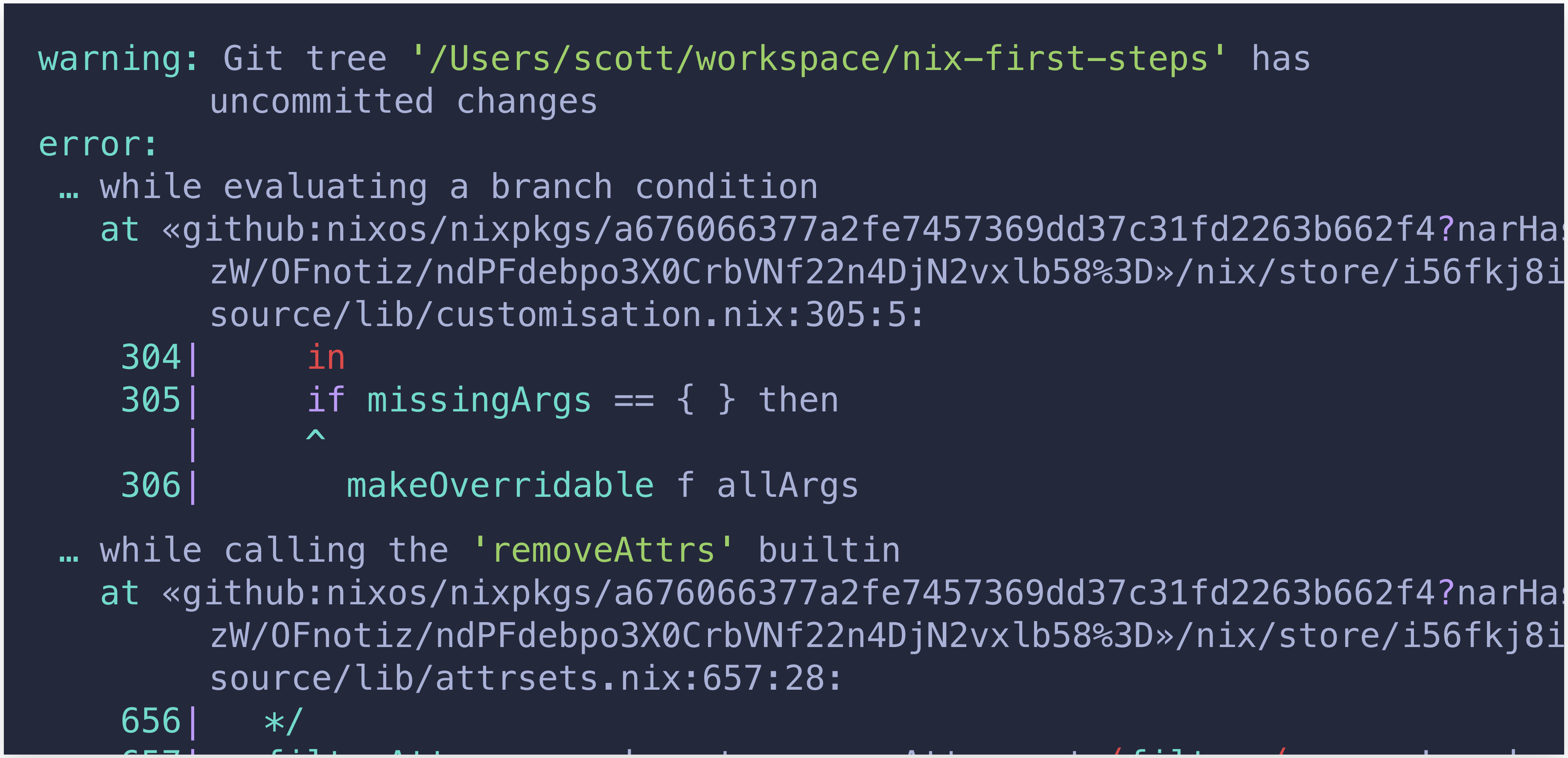
```
{
  devShell = pkgs.mkShell {
    # ...
  };
  packages.default = pkgs.callPackage ./hello-nix { inherit pkgs; }
}
```

ADDING PACKAGE TO OUR FLAKE

and rebuild it! Note the syntax again of `.#`

```
$ nix build .#
```

ERROR WITH BUILD



```
warning: Git tree '/Users/scott/workspace/nix-first-steps' has
         uncommitted changes
error:
 … while evaluating a branch condition
   at «github:nixos/nixpkgs/a676066377a2fe7457369dd37c31fd2263b662f4?narHa
         zW/OFnotiz/ndPFdebpo3X0CrbVNf22n4DjN2vxlb58%3D»/nix/store/i56fkj8i
         source/lib/customisation.nix:305:5:
    304|     in
    305|     if missingArgs == { } then
       |     ^
    306|       makeOverridable f allArgs

 … while calling the 'removeAttrs' builtin
   at «github:nixos/nixpkgs/a676066377a2fe7457369dd37c31fd2263b662f4?narHa
         zW/OFnotiz/ndPFdebpo3X0CrbVNf22n4DjN2vxlb58%3D»/nix/store/i56fkj8i
         source/lib/attrsets.nix:657:28:
    656|   */
```

CLEANING UP GIT

```
$ echo "target" >> .gitignore
$ echo ".direnv" >> .gitignore
$ git add "hello-nix"
```

BUILD SUCCESS

```
$ nix build .#
warning: Git tree '/Users/scott/workspace/nix-first-steps' has
         uncommitted changes
$ ls -l result
lrwxr-xr-x 1 scott staff 59 Jun 29 17:45 result ->
         /nix/store/yqw9zry7dsgyr692y18pb330xhwlrwr5-hello-nix-0.0.1
$ ./result/bin/hello-nix
Hello from nix!
$ rm result
```

PORTABILITY OF PACKAGE

If we push this to github we could run this automatically!

```
$ nix run github:sentientmonkey/nix-first-steps
Hello from nix!
$ nix run .#
Hello from nix!
```

LET'S BUILD FOR DOCKER

Create a new file `hello-nix/build-docker.nix`

```
{
  pkgs ? import <nixpkgs> { }
}:

pkgs.dockerTools.buildImage {
  name = "hello-nix";
  tag = "0.0.1";
  config = {
    Cmd = [ "${pkgs.hello}/bin/hello" ];
  };
}
```

BUILDING AND LOADING

```
$ docker load < $(nix build -f hello-nix/build-docker.nix --no-link --
        print-out-paths)
Loaded image: hello-nix:0.0.1
$ docker run hello-nix:0.0.1
Hello, World!
```

ADDING DOCKERIMAGE TO OUR FLAKE

add to our top level `flake.nix`

```
packages.default = pkgs.callPackage ./hello-nix { inherit pkgs; }
packages.dockerImage = pkgs.callPackage ./hello-nix/build-docker.nix {
  inherit pkgs;
}
```

RUN DOCKER BUILD WITH FLAKE

```
$ git add hello-nix/build-docker.nix
$ docker load < $(nix build .#dockerImage --no-link --print-out-paths)
Loaded image: hello-nix:0.0.1
$ docker run hello-nix:0.0.1
Hello, World!
```

SMALL REFACTOR IN OUR FLAKE

```nix
let
  pkgs = nixpkgs.legacyPackages.${system};
  helloNix = pkgs.callPackage ./hello-nix { inherit pkgs; };
in
{
  # ...
  packages.default = helloNix;
  packages.dockerImage = pkgs.callPackage ./hello-nix/build-docker.nix {
    inherit pkgs;
  };
}
```

SMALL REFACTOR TO OUR FLAKE

```nix
packages.dockerImage = pkgs.callPackage ./hello-nix/build-docker.nix {
  inherit pkgs helloNix;
};
```

BACK TO OUR BUILD, WE CAN USE OUR PACKAGE

```
{
  helloNix,
  pkgs ? import <nixpkgs> { },
}:

pkgs.dockerTools.buildImage {
  name = "hello-nix";
  tag = helloNix.version
  config = {
    Cmd = [ "${helloNix}/bin/hello-nix" ];
  };
}
```

BUILDING AGAIN WITH OUR PACKAGE NOW

```
$ docker load < $(nix build .#dockerImage --no-link --print-out-paths)
Loaded image: hello-nix:0.0.1
$ docker run hello-nix
Hello from nix!
```

EXTENDING OUR DOCKER IMAGE WITH BASH

```
pkgs.dockerTools.buildImage {
  name = "hello-nix";
  tag = helloNix.version;

  copyToRoot = pkgs.buildEnv {
    name = "image-root";
    paths = with pkgs; [
      helloNix
      bashInteractive
      coreutils
    ];
    pathsToLink = [ "/bin" ];
  };
  config = {
    Cmd = [ "/bin/hello-nix" ];
  };
}
```

EXTENDING OUR DOCKER IMAGE WITH BASH

```
$ docker load < $(nix build .#dockerImage --no-link --print-out-paths)
Loaded image: hello-nix:0.0.1
$ docker run -it /bin/bash
bash-5.2#
```

ADDING RUNTIME DEPENDENCIES

Back to our `flake.nix`

```nix
devShell = pkgs.mkShell {
  buildInputs = with pkgs; [
    cargo
    rustc
    rust-analyzer
    rustfmt
    figlet
    lolcat
  ];
};
```

TESTING PACKAGE DEPENDENCIES FOR DEVELOPMENT

```
$ cd hello-nix
$ cargo run | figlet | lolcat
    Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.02s
     Running `target/debug/hello-nix`
 _   _      _ _          __                              _      _
| | | | ___| | | ___    / _|_ __ ___  _ __ ___    _ __ (_)_  _| |
| |_| |/ _ \ | |/ _ \  | |_| '__/ _ \| '_ ` _ \  | '_ \| \ \/ / |
|  _  |  __/ | | (_) | |  _| | | (_) | | | | | | | | | | |>  <|_|
|_| |_|\___|_|_|\___/  |_| |_|  \___/|_| |_| |_| |_| |_|_/_/\_(_)
```

USING MAKEWRAPPER

```nix
pkgs.rustPlatform.buildRustPackage {
  # ...
  nativeBuildInputs = [ pkgs.makeWrapper ];

  postInstall = ''
    wrapProgram $out/bin/hello-nix \
      --prefix PATH : ${pkgs.lolcat}/bin \
      --prefix PATH : ${pkgs.figlet}/bin \
      --add-flags "| figlet | lolcat"
  '';
}
```

USING MAKEWRAPPER

```
$ nix build .#
$ cat result/bin/hello-nix
#! /nix/store/xy4jjgw87sbgwylm5kn047d9gkbhsr9x-bash-5.2p37/bin/bash -e
PATH=${PATH:+':'$PATH':'}
PATH=${PATH/':''/nix/store/jjf7ym331wzp1jsyn05b7cscflk291bd-lolcat-
        100.0.1/bin'':'/':'}
PATH='/nix/store/jjf7ym331wzp1jsyn05b7cscflk291bd-lolcat-
        100.0.1/bin'$PATH
PATH=${PATH#':'}
PATH=${PATH%':'}
export PATH
PATH=${PATH:+':'$PATH':'}
PATH=${PATH/':''/nix/store/q00xb5g6hv24yc7r6k3r6jws226vw8rm-figlet-
        2.2.5/bin'':'/':'}
PATH='/nix/store/q00xb5g6hv24yc7r6k3r6jws226vw8rm-figlet-2.2.5/bin'$PATH
PATH=${PATH#':'}
PATH=${PATH%':'}
```

RUNNING OUR BUILD

```
$ nix run .#

 _   _      _ _          __                              _      _
| | | | ___| | | ___    / _|_ __ ___  _ __ ___    _ __ (_)_  _| |
| |_| |/ _ \ | |/ _ \  | |_| '__/ _ \| '_ ` _ \  | '_ \| \ \/ / |
|  _  |  __/ | | (_) | |  _| | | (_) | | | | | | | | | | |>  <|_|
|_| |_|\___|_|_|\___/  |_| |_|  \___/|_| |_| |_| |_| |_|_/_/\_(_)
```

RUNNING FROM DOCKER

```
$ docker load < $(nix build .#dockerImage --no-link --print-out-paths)
Loaded image: hello-nix:0.0.1
$ docker run -it hello-nix:0.0.1

 _   _      _ _          __                             _      _
| | | | ___| | | ___    / _|_ __ ___  _ __ ___    _ __ (_)_  _| |
| |_| |/ _ \ | |/ _ \  | |_| '__/ _ \| '_ ` _ \  | '_ \| \ \/ / |
|  _  |  __/ | | (_) | |  _| | | (_) | | | | | | | | | | |>  <|_|
|_| |_|\___|_|_|\___/  |_| |_|  \___/|_| |_| |_| |_| |_|_/_/\_(_)
```

RUNNING THE UNWRAPPED VERSION

```
$ docker run -it hello-nix:0.0.1 /bin/.hello-nix-wrapped
Hello from nix!
```

TAKE-AWAYS AND JUMPING OFF POINTS

Now that you’ve gotten a quick tour of how nix can be helpful in building out your dev environments, I encourage you to explore and learn more.
Some jumping off points:

- ________ for pinning languages and adding services (i.e. postgres, redis)
- _____________ for building containers with nix
- ______ for more details about building flakes
- __________________________________ to help build your own packages
- __________________ to explore packages
- ___________ to learn more about nix

I hope this inspires you to learn more and experiment!

THANK YOU!

REPOS WITH SLIDES AND CODE

-
-
-
-

CONTACT INFO

- `_swindsor` on PDX DevOps Discord
- `swindsor` at gmail for email
- on github

Speaker notes