runix

a type-safe Rust interface to the Nix CLI

flox ❤️

Yannik Sander (@ysndr)
FOSDEM '23

1 2 3 4 5

CLI

1 2 3 4 5

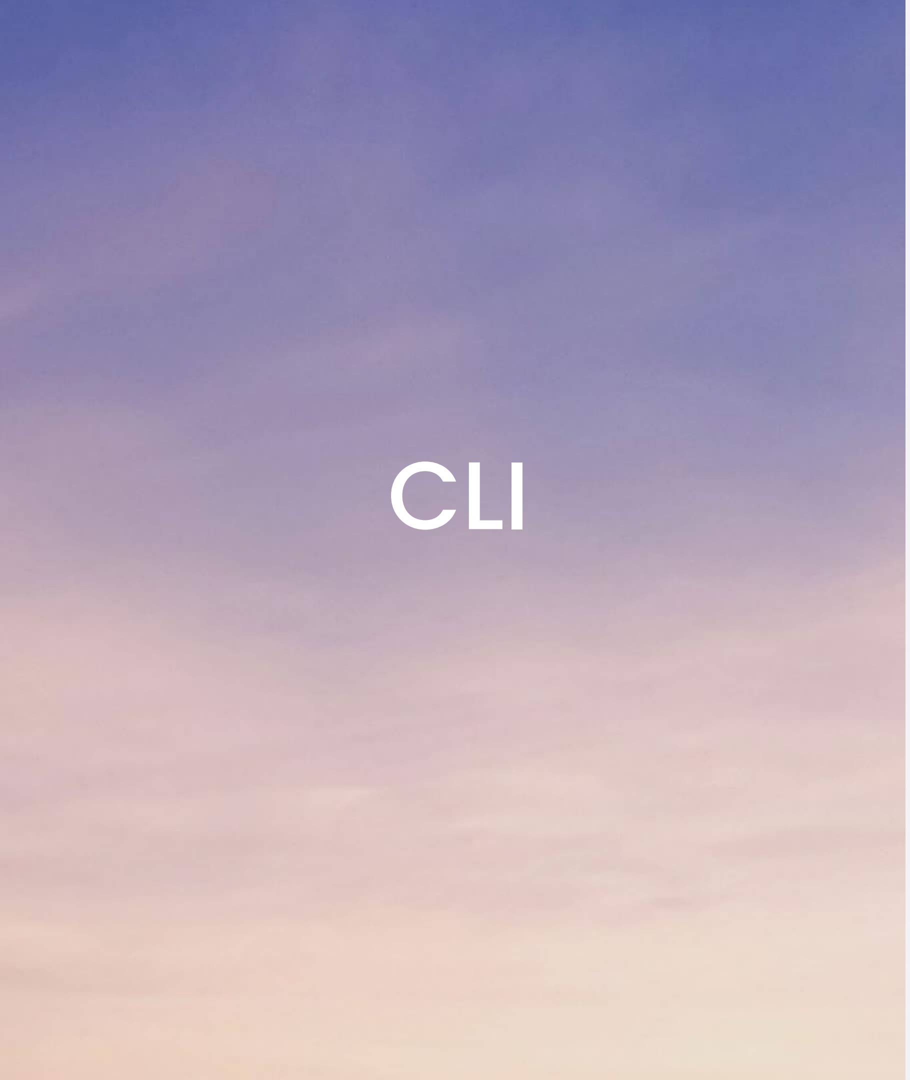
CLI

The most common interface*

```
$ nix build nixpkgs#hello
$ nix develop
$ nix flake lock --update-input nixpkgs
$ nix copy --to s3://cloud
$ nix eval --expr '“Hello FOSDEM”'
```

The list goes on…

* assuming `experimental-features = “nix-command flakes”`



CLI

- UX for user attended use
  - Completions
  - Colors
  - Dialogs
  - …
- Manual
- Single operations
- No shared context

CLI

Scripts

1 ❯ 2 3 4 5

Automation with Nix

(shell) scripts invoking `nix` directly

- Useful for simple automation
  - CI scripts
  - Local workflows
- Shell syntax
- Machine readable interfaces
  - `--json` output modifiers
- Compose `nix` commands in a common context

CLI	Scripts	execv		
1	2	3	4	5

Using Nix from *a real* language

Guess what this does...

```
let mut command = Command::with_args("nix", [
    "eval",
    "--json",
    "--no-allow-import-from-derivation",
    "--no-write-lock-file",
].iter());
command.add_arg_pair("-f", EXTRACT_SCRIPT.clone());
command.add_arg_pair("-I",
     "nixpkgs=channel:nixpkgs-unstable");
command.add_args([
     "--override-flake",
     "Input-flake",
     flake_ref.as_ref()].iter());
command.add_args(["--argstr","flake",flake_ref.as_ref()].i
ter());
command.add_arg(kind.as_ref());
command.add_arg_pair("--store",
temp_store_path.canonicalize()?);
command.add_args(extra);
```

Using Nix from *a real* language

At the mercy of your exec interface

- Automate with greater flexibility
- Build *programs* on top of Nix
- Boilerplate
- Structure is up to you
- Strings everywhere

1	2	3	4	5
CLI	Scripts	execv	Almost Native	

A rusty interface?

What do we expect from a rust interface to Nix?

- Typed
- Correct
- Predictable

types, s'il vous plaît

typen, alstublieft

types, please

Typen, bitte

typer , snälla

类型，请

Typed Commands

Eval option groups

```
$ nix eval --help

Options
    · --apply expr
       Apply the function expr to each argument.
    Common evaluation options:
    · --argstr name string
          Pass the string string as the argument name to Nix functions.
      ...
    Common flake-related options:
      ...
    Options that change the interpretation of installables:
```

Typed Commands

Eval option groups

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

- Options grouped into categories
- Modeled after `mixin` classes in the C++ backend

* nix:src/libcmd/installables.cc

** A simplified aspirational version

Typed Commands

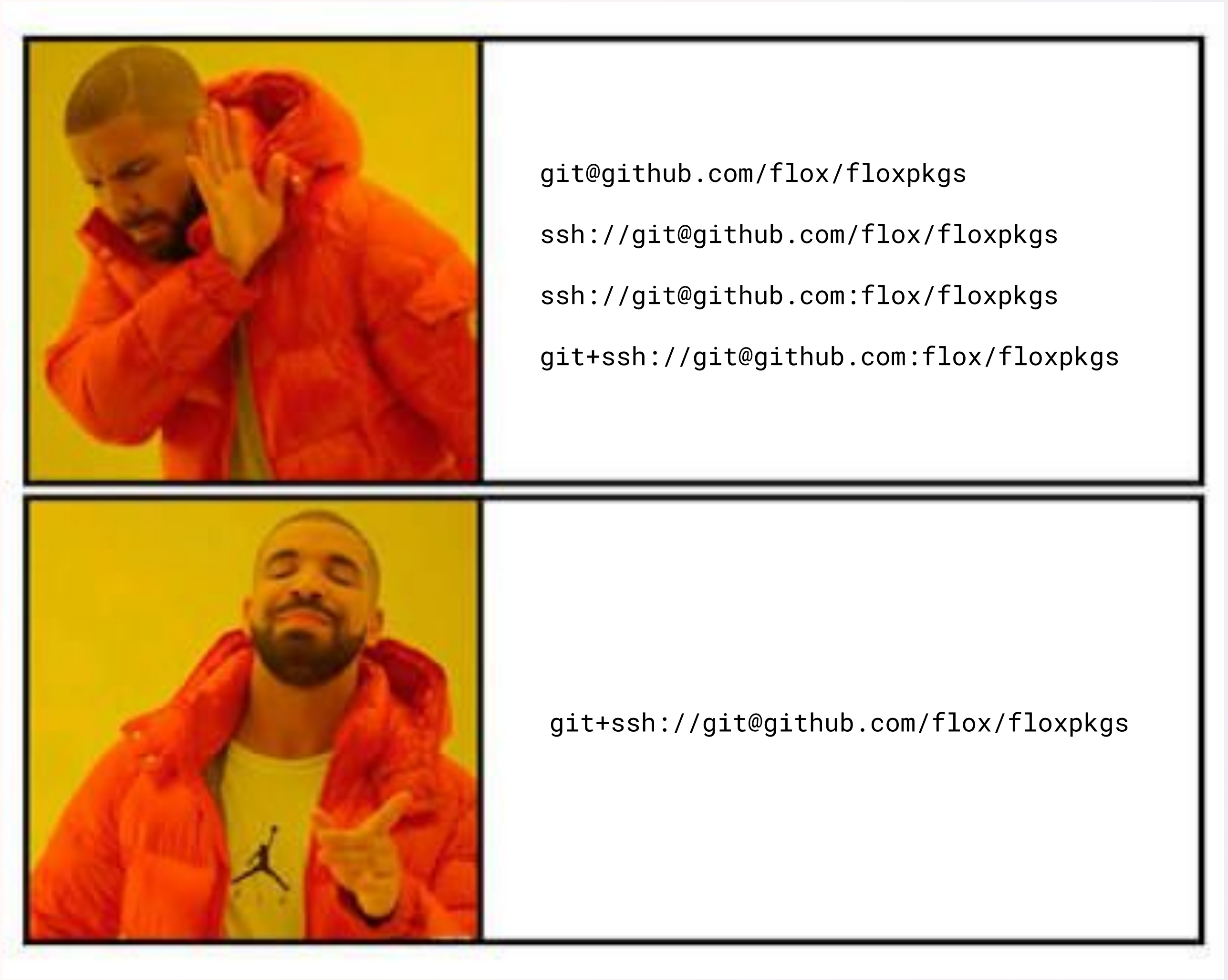
Eval option groups

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
struct FlakeArgs {
    no_write_lock_file: NoWriteLockFile,
    override_input: Vec<InputOverride>
}

struct NoWriteLockFile(bool);
struct InputOverride(String, FlakeRef);
```

* nix:src/libcmd/installables.cc



```
git@github.com/flox/floxpkgs

ssh://git@github.com/flox/floxpkgs

ssh://git@github.com:flox/floxpkgs

git+ssh://git@github.com:flox/floxpkgs
```

```
git+ssh://git@github.com/flox/floxpkgs
```

Typed Commands

Eval option groups

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
struct FlakeArgs {
    no_write_lock_file: NoWriteLockFile,
    override_input: Vec<InputOverride>
}

struct NoWriteLockFile(bool);
struct InputOverride(String, FlakeRef);
```

```
$ nix eval --override-input flox-floxpkgs git@github.com/flox/floxpkgs
error: 'git@github.com/flox/floxpkgs' is not a valid URL

$ nix eval --override-input flox-floxpkgs ssh://git@github.com/flox/floxpkgs
error: input 'ssh://git@github.com/flox/floxpkgs' is unsupported

$ nix eval --override-input flox-floxpkgs ssh://git@github.com:flox/floxpkgs
error: input 'ssh://git@github.com/flox/floxpkgs' is unsupported

$ nix eval --override-input flox-floxpkgs git+ssh://git@github.com:flox/floxpkgs
error: program 'git' failed with exit code 128
```

Typing the Nix CLI

typed commands

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
struct EvaluationArgs {
     include: Vec<Include>,
     argstr: ArgString,
     eval_store: EvalStore,
}

enum Include {
    Anonymous(IncludeValue)
    Named(String, IncludeValue),
}
enum IncludeValue {
    Channel(String)
    Flake(FlakeRef)
}
struct ArgString(String, String)
struct EvalStore(Url)
```

* nix:src/libcmd/common-eval-args.cc

Typing the Nix CLI

typed commands

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
$ nix eval -I hello=chnnel:nixpkgs-unstable \
--expr '<hello>' --impure
warning: Nix search path entry
'chnnel:nixpkgs-unstable' does not exist, ignoring
error: file 'hello' was not found in the Nix search
path (add it using $NIX_PATH
or -I)
```

```
struct EvaluationArgs {
    include: Vec<Include>,
    argstr: ArgString,
    eval_store: EvalStore,
}

enum Include {
    Anonymous(IncludeValue)
    Named(String, IncludeValue),
}
enum IncludeValue {
    Channel(String)
    Flake(FlakeRef)
}
struct ArgString(String, String)
struct EvalStore(Url)
```

Typing the Nix CLI

typed commands

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
struct SourceArgs {
    file: SourceFile,
    expr: SourceExpression,
}

struct SourceArgs(PathBuf)
struct SourceExpression(rnix::AST)
```

* nix:src/libcmd/installables.cc

Typing the Nix CLI

typed commands

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
struct EvalArgs {
    apply: Apply,
    installable: InstallableArg,
}

struct Apply(rnix::ast::Lambda)
struct InstallableArg(Installable)
```

* nix:src/nix/eval.cc

Typing the Nix CLI

typed commands

```
struct Eval {
    flake: FlakeArgs,
    eval: EvaluationArgs,
    source: SourceArgs,
    eval_args: EvalArgs,
}
```

```
struct EvalArgs {
    apply: Apply,
    installable: InstallableArg,
}


struct Apply(rnix::ast::Lambda)
struct InstallableArg(Installable)
```

```
nix eval .#foo --apply '"not-a-function"'
error: syntax error, unexpected invalid token

      at «string»:1:1:

           1| "not-a-function"
            | ^
```

* nix:src/nix/eval.cc

```
let command = Eval {
    flake: FlakeArgs {
        no_write_lock_file: true,
        override_flake: [ ("Input-flake", flake_ref) ],
    },
    source: SourceArgs {
        file: EXTRACT_SCRIPT
    },
    eval: EvaluationArgs {
        include: [
            Include::Named("nixpkgs", IncludeValue::Channel("nixpkgs-unstable"))
        ],
        argstr: [ ("flake", flake_ref) ],
        eval_store: temp_store_path.canonicalize()?,
    },
};
```

A typed interface to `nix eval`

Typing the Nix CLI

typed Nix configuration

```
struct NixArgs {
    config: NixConfigArgs
}
```

```
struct NixConfigArgs {
    allow_import_from_derivation: AllowIFD,
    flake_registry: FlakeRegistry,
    access_tokens: AccessTokens
};

struct AllowIFD(bool);
struct FlakeRegistry(PathBuf);
struct AccessTokens(Vec<(String, String)>);
```

```
let nix_args = NixArgs {
    config: NixConfigArgs {
         allow_import_from_derivation: false,
    },
};
```

A typed interface to command independent `nix` options

… run the command

```
let nix_args = NixArgs { ... };
let command = Eval { ... };

command.run(&NixCommandLine::default(), &nix_args).await?;
```

… run the command

```
let nix_args = NixArgs { ... };
let command = Eval { ... };

command.run(&NixCommandLine::default(), &nix_args).await?;
command.run(&NixCommandLine::default(), &nix_args).await?;
```

```rust
struct NixArgs {
    config: NixConfigArgs
}
```

```rust
struct NixConfigArgs {
    allow_import_from_derivation: AllowIFD,
    flake_registry: FlakeRegistry,
    access_tokens: AccessTokens
};

struct AllowIFD(bool);
struct FlakeRegistry(PathBuf);
struct AccessTokens(Vec<(String, String)>);
struct AllowIFD(bool);

Impl Flag for AllowIFD {
    const FLAG: &'static str = "--allow-import-from-derivation";
    const FLAG_TYPE: FlagType<Self> = FlagType::switch(true);
}

trait ToArgs { fn to_args(&self) -> Vec<String> }
impl<T: Flag> ToArgs for T { ... }
```

```
struct AllowIFD(bool);

Impl Flag for AllowIFD {
    const FLAG: &'static str = "--allow-import-from-derivation";
    const FLAG_TYPE: FlagType<Self> = FlagType::switch(true);
}

trait ToArgs { fn to_args(&self) -> Vec<String> }

impl<T: Flag> ToArgs for T { ... }
```

CLI	Scripts	execv	Almost Native	True Native
1	2	3	4	5

What's next

Truly native

Build on rust FFI bindings
- More efficient commands
- Custom commands
- A new Nix frontend?

Modular; implemented per command

What’s next

Less ambitious ideas

- Custom Nix frontends* for experimentation
- Testing/Mock backend
- Dry-Run

- Community pitch:

```
$ nix-explain eval --json
```

* ask me how I know

Runix
Nix
Rust
imgflip.com

Thanks for your attention!

@ysndr on Github

Crate:
crates.io/crates/runix

Contributions welcome!
`github.com/flox/runix`

We're hiring!
`floxdev.com/careers`

Interact with Nix

- Manually, via CLI
- Scripts, via sh
- Integration into programs via exec
  - Nixos-search
  - Comma
- Runix: integration that feels native
  - motivations
  - Usage
  - Concepts
- Runix next step: integration that IS native
- Contributions welcome