Debugging Derivations and Nix Expressions

Show detailed error messages

You can always try to add `--show-trace --print-build-logs --verbose` to the `nixos-rebuild` command to get the detailed error message if you encounter any errors during the deployment. e.g.

```bash
cd /etc/nixos
sudo nixos-rebuild switch --flake .#myhost --show-trace --print-build-logs --ver

# A more concise version
sudo nixos-rebuild switch --flake .#myhost --show-trace -L -v
```

Debugging with `nix repl`

> NOTE: If you have disabled `NIX_PATH`, you won't be able to use syntax like `<nixpkgs>`. Instead, you should use `nix repl -f flake:nixpkgs` to load nixpkgs.

We have frequently used nix repl `<nixpkgs>` throughout this guide to examine the source code. It is a powerful tool that helps us understand how things work in Nix.

Let's take a closer look at the help message of nix repl:

```shell
› nix repl -f '<nixpkgs>'
Welcome to Nix 2.13.3. Type :? for help.

Loading installable ''...
Added 17755 variables.
nix-repl> :?
The following commands are available:

  <expr>        Evaluate and print expression
  <x> = <expr>  Bind expression to variable
```

```
  :a <expr>     Add attributes from resulting set to scope
  :b <expr>     Build a derivation
  :bl <expr>    Build a derivation, creating GC roots in the working directory
  :e <expr>     Open package or function in $EDITOR
  :i <expr>     Build derivation, then install result into current profile
  :l <path>     Load Nix expression and add it to scope
  :lf <ref>     Load Nix flake and add it to scope
  :p <expr>     Evaluate and print expression recursively
  :q            Exit nix-repl
  :r            Reload all files
  :sh <expr>    Build dependencies of derivation, then start nix-shell
  :t <expr>     Describe result of evaluation
  :u <expr>     Build derivation, then start nix-shell
  :doc <expr>   Show documentation of a builtin function
  :log <expr>   Show logs for a derivation
  :te [bool]    Enable, disable or toggle showing traces for errors
```

There are a couple of expressions that I frequently use: `:lf <ref>` and `:e <expr>` .

The `:e <expr>` command is very intuitive, so I won't go into detail about it. Instead, let's focus on `:lf <ref>` :

nix

```
# cd into my nix-config repo(you should replace it with your own nix-config repo
› cd ~/nix-config/

# enter nix repl
› nix repl
Welcome to Nix 2.13.3. Type :? for help.

# load my nix flake and add it to scope
nix-repl> :lf .
Added 16 variables.

# press <TAB> to see what we have in scope
nix-repl><TAB>
# ......omit some outputs
__isInt                         nixosConfigurations
__isList                        null
__isPath                        outPath
__isString                      outputs
__langVersion                   packages

```

```
# ......omit some outputs


# check what's in inputs
nix-repl> inputs.<TAB>
inputs.agenix            inputs.nixpkgs
inputs.darwin            inputs.nixpkgs-darwin
inputs.home-manager      inputs.nixpkgs-unstable
inputs.hyprland          inputs.nixpkgs-wayland
inputs.nil
inputs.nixos-generators

# check what's in inputs.nil
nix-repl> inputs.nil.packages.
inputs.nil.packages.aarch64-darwin
inputs.nil.packages.aarch64-linux
inputs.nil.packages.x86_64-darwin
inputs.nil.packages.x86_64-linux

# check the outputs of my nix flake
nix-repl> outputs.nixosConfigurations.<TAB>
outputs.nixosConfigurations.ai
outputs.nixosConfigurations.aquamarine
outputs.nixosConfigurations.kana
outputs.nixosConfigurations.ruby

nix-repl> outputs.nixosConfigurations.ai.<TAB>
outputs.nixosConfigurations.ai._module
outputs.nixosConfigurations.ai._type
outputs.nixosConfigurations.ai.class
outputs.nixosConfigurations.ai.config
outputs.nixosConfigurations.ai.extendModules
outputs.nixosConfigurations.ai.extraArgs
outputs.nixosConfigurations.ai.options
outputs.nixosConfigurations.ai.pkgs
outputs.nixosConfigurations.ai.type

nix-repl> outputs.nixosConfigurations.ai.config.
outputs.nixosConfigurations.ai.config.age
outputs.nixosConfigurations.ai.config.appstream
outputs.nixosConfigurations.ai.config.assertions
outputs.nixosConfigurations.ai.config.boot
outputs.nixosConfigurations.ai.config.console
outputs.nixosConfigurations.ai.config.containers
```

```
# ......omit other outputs

nix-repl> outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.<TA
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.activation
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.activationPac
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.emptyActivati
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.enableDebugIn
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.enableNixpkgs
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.extraActivati
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.extraBuilderC
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.extraOutputsT
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.extraProfileC
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file
# ......omit other outputs


nix-repl> outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.ses
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.sessionVariab
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.sessionVariab
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.sessionVariab
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.sessionVariab
# ......omit other outputs

# check the value of `TERM`
nix-repl> outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.ses
"xterm-256color"


# check all files defined by `home.file`
nix-repl> outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.fil
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file..bash_pr
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file..bashrc
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file..config/
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file..config/
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file..config/
outputs.nixosConfigurations.ai.config.home-manager.users.ryan.home.file..config/
#......
```

As you can see, after loading your Nix flake into the REPL, you can check every attribute of the flake. This capability is very convenient for debugging purposes.

Debugging functions provided by nixpkgs

TODO

Debugging by using `NIX_DEBUG` in derivation

TODO

References

- How to make nix build display all commands executed by make?
  - use `NIX_DEBUG=7` in derivation
- Collection of functions useful for debugging broken nix expressions.

Loading comments...