Nix

Nix is a Linux (and other Unix) systems package manager

- provides around 13,000 packages

Inspired by functional programming research

- atomic (can’t be corrupted by a partially complete operation)
- immutable (can’t be corrupted/altered by updates)
- reversible (can roll back to all previous states)
- composable (can build new components out of existing components)
- garbage collected (bits no longer in use removed automatically)

Nix is a programming language used to create build specifications

- lazy, pure, functional language
- used to create shell script to drive builds
Installing (package manager and language)

SHARCNET/Compute Canada

- graham and cedar: coming soon
- orca, monk, ...: module load nix
- resetting: `rm -fr ~/.nix* ~/.config/nix`

Linux/Mac OS X

- installing: `curl https://nixos.org/nix/install | sh`
- removing: `sudo rm -fr /nix ~/.nix* ~/.config/nix`
Using (package manager)

Querying packages

- installed: `nix-env --query`
- available top level: `nix-env --query --available --attr-path`
- available collection: `nix-env --query --available --attr-path --attr <attribute>`

Installing/removing packages

- installing: `nix-env --install --attr <attribute>`
- removing: `nix-env --uninstall <name>`

Updating

- environment: `nix-env --upgrade`
- packages: `nix-channel --update`

Reverting

- environment: `nix-env --rollback`
- packages: `nix-channel --rollback`
Layers (package manager)

Derivation - output of the Nix language that specifies how to build a package

▶ what dependencies are required (includes source tarballs)
▶ how the environment should be set
▶ what program should be invoked to do the build

Instantiation - result of running the derivation in a sandboxed environment (i.e., with only the closure of the dependencies available)

▶ sanitized copy of the output directory
Key Concept (package manager)

Nix is based around pure immutable builds. Instances of packages are identified/versioned by the hash of their derivation (build instructions).

**Purity** When output is solely determined by input (input is fully specified) a think is said to be pure.

- A build in a fully specified and sandboxed environment should reproduce the same output (to within irrelevant details) each time it is ran.

**Immutability** When a thing is not allowed to change after creation it is said to be immutable.

- Immutability means each build only needs to be evaluated once and all other components that require it can share the results.
Constructs (language)

Comments
  ➤ # ...  

Layout
  ➤ non-space and non-newline sensitive  

Bindings
  ➤ to names: let n1=e1; n2=e2; n3=e3; in e4  
  ➤ from set: with e1; e2  

Null
  ➤ null
Booleans

- true or false

Operators

- conditional: if e1 then e2 else e3
- equality: e1 == e2
- inequality: e1 != e2
- logical negation: !e
- logical and: e1 && e2
- logical or: e1 || e2
- logical implication: e1 -> e2
Integers (language)

Integers

- base-10 numeric values

Operators

- negation: $- e_1$
- addition: $e_1 + e_2$
- subtraction: $e_1 - e_2$ (space required)
- multiplication: $e_1 \times e_2$
- division: $e_1 / e_2$
Strings and Paths (language)

Strings

► inline: "..."
► multiline indent stripped: '...' '
► antiquotes: ${e}

Paths

► absolute: /...
► relative: ./...

Operators

► concatenation e1 + e2
Lists (language)

Lists
- [ e1 e2 e3 ]

Operators
- list concatenation: e1 ++ e2
Sets (language)

Sets

- non-recursive: \{ k1=v1; k2=v2; k3=v3; \}
- recursive: \texttt{rec \{ k1=k2; k2=v2; k3=k1; \}}

Operators

- test attribute path: \texttt{e ? k1.k2}
- extract attribute path: \texttt{e.k1.k2}
- extract attribute path with default: \texttt{e1.k1.k2 or e2}
- override/extend attributes: \texttt{e1 \textbackslash\textbackslash e2}
Functions (language)

Functions

- curried: n1: n2: e
- set values: { n1, n2, n3 }: e
- set and values: args@{ n1, n2, n3 }: e
- set with extra keys: args@{ n1, n2, n3, ... }: e
- set with default keys: { n1, n2 ? e2, n3 ? n3 }: e

Operators

- function call: e1 e2
Packages

- Attribute of a top-level set defined using the Nix language that give the derivation (build instructions) for each package.
- Default search location is under ~/.nix-defexpr (channels, in particular, are ~/.nix-defexpr/channels/nixpkgs).

Top level nixpkgs attribute set (provided by nixpkgs channel)

- $DIR/pkg/top-level/all-packages.nix

Individual packages by category

- $DIR/pkg/pkg/...
Overlays

The top-level nixpkgs attribute set is defined such that it can be modified via ~/.config/nixpkgs/overlays files.

self: super: with self; {
    ...
}

    self  final set of packages (after overlay)
super  initial set of packages (before overlay)
Overrides

Top-level nix packages are included with the `callPackage` function.

- Fills in unspecified arguments for current scope
- Adds `override` attribute the re-evaluates with new arguments

```nix
<package>.override { <old arg> = <new value>; ... }
```
Examples

Nixpkgs Contributor Guide

**SUMO** straightforward GNU autoconfig with lots of dependencies

**CryptoMiniSat** straightforward cmake program with limited dependencies

**NLTK** straightforward python

**Meraculous** not so straightforward cmake program