Cheatsheet

From NixOS Wiki

Notice: until this page is cleaned up, it is much more easily viewed with the Vector wiki theme (

Contents

A NixOS cheat sheet and comp

Task	Ubuntu	NixOS (system-wide and
Basic concepts		
		This column will let you do everything you can with U
Who can install packages and who can run them?	All packages are always system-wide and only root can install packages.	Packages root installs are system-wide. It does so throu root installs packages the same way users do, through also global. Root's default profile is the system-wide de
Package manager	apt which is really running on top of dpkg, sometimes wrapped by UIs like aptitude.	nix, but many system-wide operations are provided by
How do you select your official sources and major releases	These are baked into the distribution (e.g. Ubuntu version X). Upgrades are hard and permanent.	At any time you select from a collection of channels. Th root. You can roll back changes or switch channels wit
Where are packages installed?	apt installs globally into /bin/, /usr/, etc.	System-wide packages are in /run/current-system/sw/ (/etc/nixos/configuration.nix) and /nix/var/nix/profiles/d managed by root). Note that the files are just symlinks nix /nix/store/.
When changes take effect	As soon as the command runs. Commands are not atomic and can leave your machine in a bad state.	Most of the time you modify the configuration file and switch **TODO**: How does one get nixos to do all the work for a actual switching from fetching/building?
Packages	Uniformly referred to as packages	Technically called "derivations" but everyone calls the
Package management		

Install a package for all users	`$ sudo apt-get install emacs`	1. Add to /etc/nixos/configuration.nix:
		`environment.systemPackages = with pkgs; [`
		`  wget # let's assume wget was already present`
		`  emacs`
		`];`
		2. Run :
		`$ sudo nixos-rebuild switch`

List package dependencies	`$ apt-cache depends emacs`	Show the direct dependencies: `$ nix-store --query --requisites /run/current-system` or show a nested ASCII tree of dependencies: `$ nix-store -q --tree /nix/var/nix/profiles/system` (/run/current-system and /nix/var/nix/profiles/system a end up at the same place.)
List which packages depend on this one (reverse dependencies)	`$ apt-cache rdepends emacs`	
Verify all installed packages	`$ debsums`	`$ sudo nix-store --verify --check-contents`
Fix packages with failed checksums	Reinstall broken packages	`$ sudo nix-store --verify --check-contents --repair`
Select major version and stable/unstable	Change sources.list and apt-get dist-upgrade. A an extremely infrequent and destructive operation. The nix variants are safe and easy to use.	`$ nix-channel --add\ https://nixos.org/channels/nixpkgs-unstable <name>` Add the unstable channel. At that address you will find variants. Name can be any string. `$ nix-channel --remove <name>` To eliminate a channel. `$ nix-channel --list` To show all installed channel.
Private package repository	PPA	Define your package tree as in the general column, anc then list your packages in systemPackages to make the
Install a particular		Although Nix on its own doesn't understand the concep install and play with older (or newer!) software via htt /Pinning_Nixpkgs with https://lazamar.co.uk/nix-versio

Show package for file	`$ dpkg -S /usr/bin/emacs`	follow the symlink or
		`nix-locate /bin/emacs`
		(requires
		`nix-index`
		package)

	Services	
Start a service	`$ sudo systemctl start apache`	`$ sudo systemctl start apache`
Stop a service	`$ sudo systemctl stop apache`	`$ sudo systemctl stop apache`

Enable a service	`$ sudo systemctl enable apache`	In /etc/nixos/configuration.nix, add
		`services.tor.enable = true;`
		, then run
		`$ sudo nixos-rebuild switch`

Disable a service	`$ sudo systemctl disable apache`	In /etc/nixos/configuration.nix, add
		`services.tor.enable = false;`
		, then run
		`$ sudo nixos-rebuild switch`

Where your log files live	/var/log/	System-wide packages /var/log/

Adding a user	`$ sudo adduser alice`	Add
		`users.users.alice =`
		`{ isNormalUser = true;`
		`home = "/home/alice";`
		`description = "Alice Foobar";`
		`extraGroups = [ "wheel" "networkmanager" ];`
		`openssh.authorizedKeys.keys =`
		`[ "ssh-dss AAAAB3Nza... alice@foobar" ];`
		`};`
		to /etc/nixos/configuration.nix and then call
		`nixos-rebuild switch`

Install a binary package		
Install a .deb	`$ sudo dpkg -i package.deb`	

Comparison of secret managing sche

Manage secrets (https://nixos.wiki/wiki/Comparison_of_secret_managing_schemes) in your (system) c
can be used and outlines the aims, requirements and implications of each.

Working with the nix store

Get the store path for a package

```
$ nix repl
nix-repl> :l <nixpkgs>
Added 7486 variables.
nix-repl> "${xorg.libXtst}"
"/nix/store/nlpnx21yjdjx2ii7ln4kcmbm0x1vy7w9-libXtst-1.2.3"

$ nix-build '<nixpkgs>' --no-build-output -A xorg.libXtst
/nix/store/nlpnx21yjdjx2ii7ln4kcmbm0x1vy7w9-libXtst-1.2.3
```

Adding files to the store

It is sometimes necessary to add files to the store manually. This is particularly the case with package
software packages. For most files, it is sufficient to run:

```
$ nix-store --add-fixed sha256 /path/to/file
```

If you only want to evaluate `configuration.nix` without building (e.g. to syntax-check or see if you are

```
$ nix-instantiate '<nixpkgs/nixos>' -A system
```

This creates the `.drv` file that `nixos-rebuild build` would build.

Manually switching a NixOS system t system closure

(*Or:* What `nixos-rebuild` does under the hoods.)

Step 1: Do this for the equivalent of `nixos-rebuild boot` or `nixos-rebuild switch`, i.e. if you want the chang

If you have the store path, run this, replacing `$systemClosure` with store path to your system closure:

```
$ nix-env --profile /nix/var/nix/profiles/system --set $systemClosure
```

Or, if it was a previous generation, you can run this instead, replacing `$generation` with the desired ger

```
$ nix-env --profile /nix/var/nix/profiles/system --switch-generation $generation
```

Step 2: Do this for all changes:

Run this, replacing `$action` with the action (one of `boot`, `switch`, `test`):

```
$ /nix/var/nix/profiles/system/bin/switch-to-configuration $action
```

If you use a different profile name the procedure is similar, but use `/nix/var/nix/profiles/system-profile`

Building a service as a VM (for testin

While `nixos-rebuild build-vm` allows to build a vm out of the current system configuration, there is a mo

Given the following configuration:

```
# vm.nix
{ lib, config, ... }:
{
  services.tor.enable = true;
  users.users.root.initialPassword = "root";
}
```

Add the following `default.nix` to the project:

```
with import <nixpkgs> {};
linuxPackages.bcc.overrideDerivation (old: {
  # overrideDerivation allows it to specify additional dependencies
  buildInputs = [ bashInteractive ninja ] ++ old.buildInputs;
})
```

To initiate the build environment run `nix-shell` in the project root directory

```
# this will download add development dependencies and set up the environment so build tools will find them.
$ nix-shell
```

The following is specific to bcc or cmake in general: (so you need to adapt the workflow depending on

```
$ mkdir build
$ cd build
# cmakeFlags is also defined in the bcc package. autotools based projects might defined $configureFlags
$ eval cmake $cmakeFlags ..
$ make
```

Evaluate packages for a different pla

Sometimes you want to check whether a change to a package (such as adding a new dependency) wou
to check on `x68_64-linux` whether a package evaluates for `x68_64-darwin` or `aarch64-linux`.

Use the `system` argument:

```
$ nix-instantiate --argstr system "x86_64-darwin" -A mypackage
```

Cross-compile packages

The following command will cross compile the tinc package for the aarch64 CPU architecture from a

```
$ nix-build '<nixpkgs>' --arg crossSystem '(import <nixpkgs> {}).lib.systems.examples.aarch64-multiplatform' -A tinc
```

You can add your own specifications, or look at existing ones, in nixpkgs/lib/systems/examples.nix.

Customizing Packages

```
$ nix-build -E 'with (import ./. {}); (curl.override { stdenv = makeStaticLibraries stdenv;}).out'
```

There is also an stdenv adapter that will build static binaries:

```
$ nix-build '<nixpkgs>' -A pkgsStatic.hello
```

Rebuild a package with debug symbo

```
$ nix-build -E 'with import <nixpkgs> {}; enableDebugging st'
$ file result/bin/st
result/bin/st: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /nix/store/f111ij1fc
```

Download a nix store path from the c

If you want to the exact same nix store path on a different system, you can use the `--realise` or short -

```
$ nix-store -r /nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10
$ find  /nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10
/nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10
/nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10/bin
/nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10/bin/hello
/nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10/share
/nix/store/0vg5bw04dn21czjcqcqczyjrhys5cv30-hello-2.10/share/locale
...
```

Install an arbitrary nix store path in

`nix-env` also accepts the full path to a program in the nix store:

```
$ nix-env -i /nix/store/yzz2gvpcyxg5i68zi11sznbsp1ypccz8-firefox-65.0
```

Check the syntax of a nix file

```
$ echo '{}: bar' > expression.nix
```