Cheatsheet

From NixOS Wiki

Notice: until this page is cleaned up, it is much more easily viewed with the Vector wiki theme (https://nixos.wiki/index.php?title=Cheatsheet&useskin=vector).

A NixOS cheat sheet and comparison to Ubuntu

This is meant to give you basic ideas and get you unstuck. NixOS is very different from most distributions, a deeper understanding will be necessary sooner or later. Please follow the links to the manual pages or browse the Wiki for more in-depth NixOS tutorials.

The system-wide column is the equivalent of using `apt` under Ubuntu.

TODO: Provide well-commented sample configuration.nix and ~/.nixpkgs/config.nix files with examples of common tasks.

Task	Ubuntu	
		This column will let you do everything you can wi
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 s do, through ~/.nixpkgs/config.nix, they are also g
Package manager	apt which is really running on top of dpkg, sometimes wrapped by UIs like aptitude.	nix, but many system-wide operations are provid
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 chann channels with ease.
Where are packages installed?	apt installs globally into /bin/, /usr/, etc.	System-wide packages are in /run/current-systen /nix/profiles/default/bin/ (this is the profile manag /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 **TODO**: How does one get nixos to do all the worl
Packages	Uniformly referred to as packages	Technically called "derivations" but everyone call

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

Install a package for a specific user only	Not possible	1. Add to /etc/nixos/configuration.nix:
		`users.users.alice.packages = with pkg`
		2. Run:
		`$ sudo nixos-rebuild switch`

Install a service	`$ sudo apt install openssh-server`	1. Add to /etc/nixos/configuration.nix: `services.openssh.enable = true;` 2. Run: `$ sudo nixos-rebuild switch`
Uninstall a package	`sudo apt-get remove emacs`	remove from /etc/nixos/configuration.nix `$ sudo nixos-rebuild switch`
Uninstall a package removing its configuration	`$ sudo apt-get purge emacs`	All configuration is in configuration.nix
Update the list of packages	`$ sudo apt-get update`	`$ sudo nix-channel --update`
Upgrade packages	`$ sudo apt-get upgrade`	`$ sudo nixos-rebuild switch`
Check for broken dependencies	`$ sudo apt-get check`	`$ nix-store --verify --check-contents`
List package dependencies	`$ apt-cache depends emacs`	Show the direct dependencies: `$ nix-store --query --requisites /run` or show a nested ASCII tree of dependencies: `$ nix-store -q --tree /nix/var/nix/pr` (/run/current-system and /nix/var/nix/profiles/syst
List which packages depend on this one (reverse dependencies)	`$ apt-cache rdepends emacs`	
Verify all installed packages	`$ debsums`	`$ sudo nix-store --verify --check-con`
Fix packages with failed checksums	Reinstall broken packages	`$ sudo nix-store --verify --check-con`

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` Add the unstable channel. At that address you w `$ 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 colum make them available system wide
Install a particular version of a package	`$ apt-get install package=version`	Although Nix on its own doesn't understand the c via https://nixos.wiki/wiki/FAQ/Pinning_Nixpkgs (h (https://lazamar.co.uk/nix-versions). For instance, to launch an older version of Vim y `$ nix-shell \ -p vim \ -I nixpkgs=\https://github.com/Ni`
Configure a package	`$ sudo dpkg-reconfigure <package>`	Edit /etc/nixos/configuration.nix
Global package configuration	Modify configuration file in /etc/	Edit /etc/nixos/configuration.nix
Find packages	`$ apt-cache search emacs`	`$ nix-env -qaP '.*emacs.*'` or `$ nix search emacs`
Show package description	`$ apt-cache show emacs`	`$ nix-env -qa --description '.*emacs.`

Show files installed by package	`$ dpkg -L emacs`	`$ readlink -f $(which emacs)` `/nix/store/ji06y4haijly0i0knmr986l2d` then `$du -a /nix/store/ji06y4haijly0i0knmr`
Show package for file	`$ dpkg -S /usr/bin/emacs`	follow the symlink or `nix-locate /bin/emacs` (requires `nix-index` package)
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" "networkma`
		`   openssh.authorizedKeys.keys =`
		`      [ "ssh-dss AAAAB3Nza... alice@f`
		` };`
		to /etc/nixos/configuration.nix and then call
		`nixos-rebuild switch`

List binaries	`$ ls /usr/bin/`	`$ ls /run/current-system/sw/bin &&\`
		`ls /nix/var/nix/profiles/default/bin/`

Get the current version number	`$ cat /etc/debian_version`	`$ nixos-version`
Get sources for a package	`$ sudo apt-get source emacs`	

Compile & install a package from source		
Install a binary package		
Install a .deb	`$ sudo dpkg -i package.deb`	

Comparison of secret managing schemes

Manage secrets (https://nixos.wiki/wiki/Comparison_of_secret_managing_schemes) in your (system) configuration. That page tries to give an overview of different schemes that 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 packages that cannot be downloaded automatically, for example, proprietary software packages. For most files, it is sufficient to run:

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

Unfortunately, `nix-store` will try to load the entire file into memory, which will fail if the file size exceeds available memory. If we have root access, we can copy the file to the store ourselves:

```
$ sudo unshare -m bash  # open a shell as root in a private mount namespace
$ largefile=/path/to/file
$ hash=$(nix-hash --type sha256 --flat --base32 $largefile)  # sha256 hash of the file
$ storepath=$(nix-store --print-fixed-path sha256 $hash $(basename $largefile))  # dest
$ mount -o remount,rw /nix/store  # remount the store in read/write mode (only for this
$ cp $largefile $storepath  # copy the file
$ printf "$storepath\n\n0\n" | nix-store --register-validity --reregister  # register t
$ exit  # exit to the original shell where /nix/store is still mounted read-only
```

To add a file with fixed name (when the input filename is not stable), or to add entire directories with filter, you can use **builtins.path**:

```
$ nix-instantiate --eval --read-write-mode -E 'builtins.path { path = ./myfile; name =
```

Build nixos from nixpkgs repo

The following snippet will build the system from a git checkout:

```
$ nixos-rebuild -I nixpkgs=/path/to/nixpkgs switch
```

This method can be used when testing nixos services for a pull request to nixpkgs.

Building nixos from a git is an alternative to using nix channels and set up permanent following this blog article (https://web.archive.org/web/20160327190212/http://anderspapitto.com/posts/2015-11-01-nixos-with-local-nixpkgs-checkout.html). It has a couple of advantages over nixpkgs as it allows back-porting of packages/changes to stable versions as well as applying customization.

Use the following command to build directly from a particular branch of a repository in GitHub:

```
$ nixos-rebuild -I nixpkgs=https://github.com/nixcloud/nixpkgs/archive/release-17.03.ta
```

Evaluate a NixOS configuration without building

If you only want to evaluate `configuration.nix` without building (e.g. to syntax-check or see if you are using module options correctly), you can use:

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

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

Manually switching a NixOS system to a certain version of 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 changes to persist after reboot:

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 generation number:

```
$ 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-profiles/$profileName` instead of `/nix/var/nix/profiles/system` .

Building a service as a VM (for testing)

While `nixos-rebuild build-vm` allows to build a vm out of the current system configuration, there is a more light-weight alternative when only a single service needs to be tested.

Given the following configuration:

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

a vm can be build using the following command:

```
$ nixos-rebuild -I nixpkgs=/path/to/nixpkgs -I nixos-config=./vm.nix build-vm
```

where `-I nixpkgs=/path/to/nixpkgs` is optionally depending whether the vm should be build from git checkout or a channel.

On non-nixos (linux) systems the following command can be used instead:

```
$ nix-build '<nixpkgs/nixos>' -A vm -k -I nixos-config=./vm.nix
```

By default the resulting vm will require X11 to create a virtual display. By specifying additional arguments via the environment variables `QEMU_OPTS` and `QEMU_KERNEL_PARAMS` it is possible to reuse the current running terminal as serial console for the vm:

```
$ export QEMU_OPTS="-nographic -serial mon:stdio" QEMU_KERNEL_PARAMS=console=ttyS0
$ /nix/store/lshw31yfbb6izs2s594jd89ma4wf8zw6-nixos-vm/bin/run-nixos-vm
```

To forward a port you can set export `QEMU_NET_OPTS` . In the following example port 2222 on the host is forwarded to port 22 in the vm:

```
$ export QEMU_NET_OPTS="hostfwd=tcp::2222-:22"
```

Don't forget that by default nixos comes with a firewall enabled:

```
{...}: {
  networking.firewall.enable = false;
}
```

Reuse a package as a build environment

As packages already contains all build dependencies, they can be reused to a build environment quickly. In the following a setup for the cmake-based project [bcc](https://github.com/iovisor/bcc (https://github.com/iovisor/bcc)) is shown. After obtaining the source:

```
$ git clone https://github.com/iovisor/bcc.git
$ cd bcc
```

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 t
$ nix-shell
```

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

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

Evaluate packages for a different platform

Sometimes you want to check whether a change to a package (such as adding a new dependency) would evaluate even on a different type of system. For example, you may want 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 different architecture (e.g. x86_64).

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

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

Customizing Packages

Upgrading individual packages to a different channel

One can track multiple channels on NixOS simultaneously, and then declaratively change packages from the default channel to another one.

For example one can have both the unstable and stable channels on system root:

```
$ sudo nix-channel --list
nixos https://nixos.org/channels/nixos-17.03
nixos-unstable https://nixos.org/channels/nixos-unstable
```

and the following in `configuration.nix`:

```
nixpkgs.config = {
  # Allow proprietary packages
  allowUnfree = true;

# Create an alias for the unstable channel

packageOverrides = pkgs: {
unstable = import <nixos-unstable> { # pass the nixpkgs config to the unstable alias #
config = config.nixpkgs.config;
};
};
};
```

which allows you to switch particular packages to the unstable channel:

```
environment.systemPackages = with pkgs; [
    ddate
    devilspie2
    evince
    unstable.google-chrome
    # ...
    zsh
];
```

Building statically linked packages

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

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

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

Rebuild a package with debug symbols

```
$ 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,
```

Download a nix store path from the cache

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

```
$ 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 into a user profile

`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
$ nix-instantiate --parse-only expression.nix
error: undefined variable 'bar' at /tmp/expression.nix:1:5
```

Using override with nix-build

using channels

```
nix-build -E 'with (import <nixpkgs>{}); polybar.override { i3Support = true; }'
```

using a local repo

```
nix-build -E 'with (import ./default.nix{}); polybar.override { i3Support = true; }'
```

See also

- Garbage Collection (/wiki/Garbage_Collection) - Nix store on NFS (/wiki/NFS#Nix_store_on_NFS)

Retrieved from "https://nixos.wiki/index.php?title=Cheatsheet&oldid=6170 (https://nixos.wiki/index.php?title=Cheatsheet&oldid=6170)"