

Nix / NixOS Cheatsheet

Most Important Documentation Links

Nix Docs (Tool, Language)	https://nixos.org/manual/nix
Nixpkgs (Packaging, prog langs, library)	https://nixos.org/manual/nixpkgs
NixOS	https://nixos.org/manual/nixos
Nixpkgs package and NixOS module search engine	https://search.nixos.org

Imperative Package Management

Update package list	<code>apt update</code>	<i>happens automatically</i>
Search	<code>apt search <pkgname></code>	<code>nix search nixpkgs <pkgname></code>
Install	<code>apt install <pkgname></code>	<code>nix profile install nixpkgs#<pkgname></code>
Upgrade installed	<code>apt upgrade</code>	<code>nix profile upgrade '.*'</code>
List installed	<code>dpkg -l</code>	<code>nix profile list</code>
Remove	<code>apt remove <pkgname></code>	<code>nix profile remove <list-number></code>
Rollback	-	<code>nix profile rollback</code>

Per-Project Shells

Ad-hoc shell with packages	<code>nix shell nixpkgs#pkg1</code> or <code>nix shell nixpkgs#{pkg1,pkg2}</code>
Project-shell with flake	<code>nix develop</code>
Project-shell with shell.nix or default.nix file	<code>nix-shell</code>

Building Packages

Build default.nix or default pkg from flake	<code>nix build</code>
Build specific attributes (flakes)	<code>nix build .#pkg1 .#pkg2</code>

Input Management

Flakes	
Init flake project	<code>nix flake init</code>
Init flake-parts project	<code>nix flake init -t github:hercules-ci/flake-parts</code>
Update flake inputs	<code>nix flake update</code>
Update and commit lock file	<code>nix flake update --commit-lock-file</code>
Update specific input	<code>nix flake lock --update-input <name></code>
Niv (Pre Flakes)	
Install Niv Files	<code>niv init</code>
Add GitHub Repository	<code>niv add <github-owner>/<reponame></code>
Update inputs	<code>niv update</code>
Update specific input	<code>niv update <name></code>
Switch branch/tag of input	<code>niv update <name> -b <gitref></code>

Flake References

Flake in current Directory	<code>.</code>
Local path	<code>[path:]/path/to/repo</code>
HTTPS URL to flake tarball	<code>https://host/flare.tar.gz</code>
Git Repo via HTTPS	<code>git+https://host/repo</code>
Git Repo via SSH	<code>git+ssh://git@host/repo</code>
GitHub Repo	<code>github:owner/repo</code>
Specific Branch/Tag	<code>git+ssh://git@host/repo?ref=abc123</code>

Nix REPL

Start Nix REPL	<code>nix repl</code>
Load local Flake	<code>:lf .</code>
Build derivation	<code>:b attribute.with.derivation</code>
Build & Install derivation	<code>:i attribute.with.derivation</code>
Fully print expression	<code>:p some.expression</code>
Show documentation of builtin function	<code>:doc builtins.listToAttrs</code>
Show REPL help	<code>:?</code>

NixOS System Rebuild

Rebuild system and activate	<code>nixos-rebuild switch</code>
Rebuild system and activate for now without updating bootloader	<code>nixos-rebuild test</code>
Rebuild w/o activating, but update bootloader	<code>nixos-rebuild boot</code>
Rollback	<code>nixos-rebuild switch --rollback</code>
Build on host a, deploy to host b, authorize with sudo	<code>nixos-rebuild switch --build-host a --target-host b --use-remote-sudo</code>

Note: nixos-rebuild expects a flake /etc/nixos/flare.nix to exist, and within that flake, a nixosConfiguration attribute with the hostname of the current system.

Garbage Collection

Collect unreferenced nix store paths	<code>nix-collect-garbage</code>
Also collect old profile/system generations	<code>nix-collect-garbage -d</code>
Only delete up to 50GB	<code>nix-collect-garbage --max-freed 50G</code>
Find and link identical files	<code>nix-store --optimise</code>
Print all GC roots	<code>nix-store --gc --print-roots</code>



Professional Nix & NixOS Trainings

<https://nixcademy.com/>

Nix Language Cheat Sheet

Most Important Documentation Links

Nix builtins.* functions	https://nixos.org/manual/nix/stable/language/builtins
Nixpkgs function library	https://nixos.org/manual/nixpkgs
Nixpkgs function search engine	https://noogle.dev

Types

String	"this is a string"
Multi-line String (double single-quotes)	'' foo bar ''
Boolean	true, false
Null	null
Integer	123, -123
Float	3.14
Path	/an/absolute/path ./a/relative/path ../dir/up/and/down
Simple attribute set	{ a = 1; b = 2; }
Nested attribute set	{ a = 1; b = { c = 3; d = 4; }; }
Recursive attribute set (potential anti-pattern)	rec { x = 1; y = x + 1; }
List	[3 2.0 "one" null]

Syntax

Comment	# a single-line comment /* a multi-line comment */
If-then-else	if x > 3 then 10 else -10
Local variables	let x = 1; y = 2; in x + y # => returns 3
Attribute set update operator //	{ x = 1; } // { y = 2; } # => returns { x = 1; y = 2; } { x = 1; } // { x = 2; } # => returns { x = 2; }
Attribute set has-operator	let set = { x = 1; }; in set ? x # => returns true

Reference attribute keys	let s = { x = { y = 1; }; }; in s.x.y # => returns 1
Reference optional attribute keys	let set = { x = 1; }; in set.y or 2 # => returns 2
Note: Variable assignments exist only in 3 places: attribute sets, let-in blocks, nix REPL	
List concatenation	[1 2] ++ [3 4] # => returns [1 2 3 4]
Inherit	let x = 1; in { inherit x; } same as let x = 1; in { x = x; }
Inherit from scope	let x = { y = 1; }; in { inherit (x) y; } same as let x = { y = 1; }; in { y = x.y; }
With-expressions (potential anti-pattern)	let set = { x = 1; y = 2; }; in with set; x + y # => returns 3

Functions

Simple function as in Python:	let f = x: x + 1; in f 1 # => returns 2
def f(x): return x + 1	
Function with 2 parameters as in Python:	let f = x: y: x + y; in f 1 2 # => returns 3
def f(x, y): return x + y	
Function with named parameters	let f = { x, y }: x + y; in f { x = 1; y = 2; } # => returns 3
Match specific parameters, ignore others	let f = { x, y, ... }: x + y; in f { x=1; y=2; z=10; } # => returns 3
Function with default values	let f = { x, y ? 2 }: x + y; in f { x=1; } # => returns 3

Full attribute set match (potential anti-pattern)	let f = set@{ x, ... }: x + set.y; in f { x = 1; y = 2; } # => returns 3
Recursive function calls itself	let f = x: if x == 0 then 0 else x + f (x - 1); in f 3 # => returns 6

Other Special Syntax

String interpolation	let x = "bar"; in "foo \${x} baz" # => returns "foo bar baz"
Masking \${...}	"this is \\${masked}" # => returns "this is \${masked}"
Masking \${...} in double single-quote strings	''this is ''\${masked}''
Paths (Copied to nix store when referenced)	/an/absolute/path ./a/relative/path ../dir/up/and/down
Key from variable in attribute set	let x = "key"; in { \${x} = "value"; }

Special Builtin Functions

Import file and return expression	import ./some/file.nix
Assertion	assert 1 + 1 == 2; 10 # => returns 10 without error
Abort evaluation with error	abort "this describes the error"
Throw an exception when referenced	throw "this describes the exception"

