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

Per-Project Shells

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

Building Packages

Build `default.nix` or default pkg from flake	`nix build`
Build specific attributes (flakes)	`nix build .#pkg1 .#pkg2`

Input Management

Flakes

Init flake project	`nix flake init`
Init flake-parts project	`nix flake init -t github:hercules-ci/flake-parts`
Update flake inputs	`nix flake update`
Update and commit lock file	`nix flake update --commit-lock-file`
Update specific input	`nix flake lock --update-input <name>`

Niv (Pre Flakes)

Install Niv Files	`niv init`
Add GitHub Repository	`niv add <github-owner>/<reponame>`
Update inputs	`niv update`
Update specific input	`niv update <name>`
Switch branch/tag of input	`niv update <name> -b <gitref>`

Flake References

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

Nix REPL

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

NixOS System Rebuild

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

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

Garbage Collection

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

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:
 	
`def f(x):`
`  return x + 1`
	`let`
	`  f = x: x + 1;`
	`in`
	`  f 1 # ⇒ returns 2`

Function with 2 parameters as in Python:
 	
`def f(x, y):`
`  return x + y`
	`let`
	`  f = x: y: x + y;`
	`in`
	`  f 1 2 # ⇒ returns 3`

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"`