Nim Compiled, Python-like language

START HERE

nim c -r *<file.nim>*
Compile and run

nim r *<file.nim>*
Run via nimcache

nim c -d:release *<file.nim>*
Optimized build

nim c -o:*<out>* *<file.nim>*
Name the binary

nim js *<file.nim>*
Compile to JavaScript

nimble init
Scaffold a package

nimble build
Build the package

nimble test
Run tests/t*.nim

CONCEPTS

nim c builds a debug binary with checks on. Benchmark and ship with -d:release (or -d:danger).

Nim source app.nim
↓ nim c
Generated C in nimcache
↓ cc + ld
Binary app
↓ nim js
JavaScript app.js

Nim output defaults to the source name beside the file; nim r keeps the binary in nimcache.

Nim compiles each backend through generated code: C by default, C++ with cpp, JS with js.

COMPILE AND RUN

nim c *<file.nim>*
Compile with the C backend

nim c -r *<file.nim> <args>*
Compile, then run with args

nim r *<file.nim> <args>*
Compile to nimcache and run

nim c -d:release *<file.nim>*
Optimized release build

nim c -o:*<file>* *<file.nim>*
Set the output file name

nim c --outdir:*<dir>* *<file.nim>*
Set the output directory

nim cpp *<file.nim>*
Compile via C++

nim js *<file.nim>*
Compile to JavaScript

nim c -d:danger *<file.nim>*
Optimize, all runtime checks off

nim c --opt:size *<file.nim>*
Optimize for size

nim c --app:lib *<file.nim>*
Build a shared library

nim c --app:staticlib *<file.nim>*
Build a static library

nim c -f *<file.nim>*
Force a rebuild of all modules

nim e *<script.nims>*
Run a NimScript file

CHECKS AND FLAGS

nim check *<file.nim>*
Check syntax and types only

CHECKS AND FLAGS (cont.)

nim c -d:*<symbol>* *<file.nim>*
Define a symbol

nim c -d:*<symbol>***:***<val>* *<file.nim>*
Define a symbol with a value

nim c --threads:on *<file.nim>*
Enable threads

nim c --mm:*<orc|arc|refc>* *<file.nim>*
Pick memory management

nim c --path:*<dir>* *<file.nim>*
Add a module search path

nim c --hints:off *<file.nim>*
Silence hints

nim c --warning:*<name>***:off** *<file.nim>*
Switch off one warning

nim c --warningAsError:*<w>***:on** *<f.nim>*
Make a warning an error

nim c -x:off *<file.nim>*
Disable runtime checks

nim c -a:off *<file.nim>*
Disable assertions

nim c --stackTrace:on --lineTrace:on *<file.nim>*
Trace lines in crashes

nim c --styleCheck:error *<file.nim>*
Enforce Nim naming style

nim c --parallelBuild:0 *<file.nim>*
Build with all cores

nim dump
Show defines and search paths

C TOOLCHAIN AND CROSS

nim c --cc:*<gcc|clang>* *<file.nim>*
Choose the C compiler

nim c --passC:*<flag>* *<file.nim>*
Pass a flag to the C compiler

nim c --passL:*<flag>* *<file.nim>*
Pass a flag to the linker

nim c --nimcache:*<dir>* *<file.nim>*
Set the generated-files directory

nim c --cpu:*<cpu>* **--os:***<os>* *<file.nim>*
Cross compile

nim c -d:mingw *<file.nim>*
Cross compile to Windows

C TOOLCHAIN AND CROSS (cont.)

nim c --genScript *<file.nim>*
Emit a build script, no link

nim c --debugger:native *<file.nim>*
Build for gdb debugging

nim c --cpu:arm --os:linux calls a cross compiler named like arm.linux.gcc from your config.

CONFIGURATION

nim.cfg
Per-project switches, one per line

config.nims
NimScript project config

config.nims

```
switch("d", "release")
switch("mm", "orc")
switch("path", "$projectDir/src")
task build, "build app":
  exec "nim c -o:bin/app src/app.nim"
```

Command-line switches win over config files; later configs override earlier ones.

NIMBLE PACKAGES

nimble init
Start a new package (wizard)

nimble build
Build the package binaries

nimble run *<bin> <args>*
Build and run a binary

nimble test
Run tests/t*.nim

nimble install
Install the package in this dir

nimble install *<pkg>*
Install a package

nimble install *<pkg>***@***<ver>*
Install a specific version

nimble install --depsOnly
Install only dependencies

nimble uninstall *<pkg>*
Remove a package

nimble search *<term>*
Find packages by name or tag

nimble list -i
List installed packages

nimble refresh
Update the package list

nimble check
Validate the package

nimble tasks
List custom tasks

nimble *<task>*
Run a task from the .nimble file

LOCAL WORKFLOW

nimble setup
Create nimble.paths for the project

nimble lock
Pin versions in nimble.lock

nimble develop *<pkg>*
Clone a package for hacking

Commit nimble.lock; keep nimble.paths and nimble.develop out of git, they are machine-local.

PACKAGE FILE

mypkg.nimble

```
version     = "0.1.0"
author      = "Jane Doe"
description = "Small tool"
license     = "MIT"
srcDir      = "src"
bin         = @["mytool"]

requires "nim ≥ 2.0.0"
requires "jester ≥ 0.5.0"
```

Custom task and hook

```
task hello, "Say hi":
  echo "Hello"

before hello:
  echo "before"
```

LANGUAGE BASICS

Variables

```
let x = 1        # immutable
var y = 2        # mutable
const z = 3      # compile time
```

Procs

```
proc add(a, b: int): int =
  a + b

echo add(1, 2)
```

Loops and collections

```
for i in 0 ..< 3: echo i
var s = @[1, 2, 3]
s.add 4
for x in s: echo x
```

Types

```
type Point = object
  x, y: int
let p = Point(x: 1, y: 2)
```

RECIPES

Start a new binary

```
mkdir hello && cd hello
nimble init
nimble build
nimble run
```

Fast iteration

```
nim r --hints:off app.nim
nim check app.nim
```

Ship a release binary

```
nim c -d:release --opt:size \
  -o:bin/app src/app.nim
```

Web build

```
nim js -d:release -o:app.js app.nim
```