Nim

Compiled, Python-like language

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.

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

CHECKS AND FLAGS

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

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

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

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

CONFIGURATION

config.nims
NimScript project config

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 --depsOnly
Install only dependencies

LOCAL WORKFLOW

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"
```

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)
```