Racket Language and raco tools

START HERE

`racket`
Start the REPL

`raco exe -o <out> <file.rkt>`
Build an executable

`racket <file.rkt>`
Run a module

`raco docs <term>`
Search the docs

`racket -e '<expr>'`
Evaluate an expression

`raco make <file.rkt>`
Compile to bytecode

`raco test <path>`
Run tests

`raco pkg install <pkg>`
Install a package

CONCEPTS

A file starts with #lang racket (or racket/base, typed/racket, ...). Running racket <file> requires that module and runs its main submodule if present.

Write #lang racket in app.rkt
↓ `raco make`
Compile compiled/*.zo bytecode
↓ `raco test`
Test test submodule
↓ `raco exe`
Ship executable + distribute

racket with a file argument does not open a REPL. Add -i to run the file and then stay in the REPL.

raco is the tool launcher: raco help lists every command installed, and packages can add their own (raco fmt).

RUN RACKET

`racket`
Start the REPL

`racket <file.rkt> [<args>...]`
Run a module with arguments

`racket -e '<expr>'`
Evaluate an expression and print it

RUN RACKET (cont.)

`racket -l racket/base -e '<expr>'`
Eval with a fast-loading language
-e needs bindings in scope; -l racket/base is quicker to load than the default racket/init.

`racket -i <file.rkt>`
Run a file, then enter the REPL

`racket -t <file.rkt>`
Require a file and its main submodule

`racket -l <lib>`
Require a library collection path

`racket -v`
Print the version banner

THE REPL

`(exit) / Ctrl+D`
Leave the REPL

`,help`
Show REPL help

`,enter <file.rkt>`
Enter a module's namespace

`,require <module>`
Require a module

`,reload`
Reload the last required module

`,time <expr>`
Time an expression

THE REPL (cont.)

Lines starting with a comma are REPL commands, not Racket code. Use ,enter to test a file's private definitions.

RACO MAKE AND EXE

`raco make <file.rkt>`
Compile a module and its dependencies

`raco exe -o <out> <file.rkt>`
Build a stand-alone executable

`raco distribute <dir> <exe>`
Package an exe with its runtime files

`raco setup <collection>`
Rebuild one collection

Executables are tied to the OS and architecture that built them. Run raco distribute before copying one to another machine.

TESTS

`raco test <file.rkt>`
Run a module's test submodules

`raco test <dir>`
Test all modules in a directory

`raco test -c <collection>`
Test an installed collection

`raco test -p <package>`
Test an installed package

TESTS (cont.)

`raco test -j <n> <path>`
Run tests in parallel

`(module+ test ...)`
Add tests inside the module
The test submodule is skipped when the module runs normally and only runs under raco test.

`(require rackunit)`
Load the unit-test library

`(check-equal? <actual> <expected>)`
Check two values are equal

`(check-true <expr>)`
Check an expression is true

`(check-exn <pred> (lambda () ...))`
Check that code raises an error

`(test-case "<name>" ...)`
Group checks under a name

PACKAGES

`raco pkg install <pkg>`
Install a package from the catalog

`raco pkg install --auto <pkg>`
Install and fetch missing dependencies

`raco pkg update <pkg>`
Update a package

`raco pkg update --all`
Update all packages

`raco pkg remove <pkg>`
Uninstall a package

PACKAGES (cont.)

`raco pkg show`
List installed packages

`raco pkg install --link <dir>`
Link a local directory as a package

`raco pkg install <git-url>`
Install from a Git URL

`raco pkg new <name>`
Create stub files for a package

`raco pkg catalog-show <pkg>`
Look up a package in the catalogs

TOOLS AND DOCS

`raco docs <term>`
Search the documentation

`raco fmt <file.rkt>`
Print the formatted program
Needs the fmt package: raco pkg install fmt.

`raco fmt -i <file.rkt>`
Format in place

`raco expand <file.rkt>`
Show macro-expanded code

`raco help`
List all raco commands

LANGUAGE BASICS

`#lang racket`
Full language with common libraries

LANGUAGE BASICS (cont.)

`#lang racket/base`
Small, fast-loading language

`(require <module>)`
Import a library or file

`(require "<file.rkt>")`
Import a sibling file by path

`(provide <id> ...)`
Export bindings from a module

`(provide (all-defined-out))`
Export everything defined here

`(define (<f> <x>) ...)`
Define a function

`(module+ main ...)`
Code run by racket <file>

`(define args (current-command-line-arguments))`
Read command-line arguments

`(command-line #:args (<a>) ...)`
Parse flags and arguments

RECIPES

Script with tests

```
#lang racket/base
(provide add)
(define (add a b) (+ a b))
(module+ test
  (require rackunit)
  (check-equal? (add 1 2) 3))
(module+ main
  (displayln (add 1 2)))
```

Compile, test, ship

```
raco make main.rkt
raco test .
raco exe -o app main.rkt
raco distribute dist app
```