Crystal Compiled, Ruby-like language

START HERE

`crystal` **run** *<file.cr>*
Compile and run

`crystal` **build --release** *<file.cr>*
Optimized binary

`crystal` **eval '***<code>***'**
Run a snippet

`crystal` **spec**
Run the test suite

`crystal` **init app** *<name>*
Scaffold a project

`crystal` **tool format**
Format source files

shards install
Install dependencies

`crystal` **docs**
Generate API docs

CONCEPTS

Crystal is compiled (via LLVM) with Ruby-like syntax and static types that are mostly inferred. There is no interpreter: run compiles to a cached temp binary each time.

Write src/app.cr
↓ `shards install`
Deps lib/ + shard.lock
↓ `crystal build`
Build bin/app
↓ `crystal spec`
Test spec/*_spec.cr

Plain build is fast but unoptimized (-O0). Benchmark and ship with --release; it is much slower to compile.

--release implies -O3 and a single-module build; use -O1 or -O2 for a middle ground when compile time hurts.

RUN AND BUILD

`crystal` **run** *<file.cr>*
Compile and run a file

`crystal` **run** *<file.cr>* **--** *<args>*
Pass arguments to the program
Everything after -- goes to your program, not the compiler.

`crystal` *<file.cr>*
Shorthand for run

`crystal` **build** *<file.cr>*
Build a binary in the current dir

`crystal` **build --release** *<file.cr>*
Optimized build (-O3, single module)

`crystal` **build -o** *<path>* *<file.cr>*
Choose the output path

`crystal` **build --no-debug** *<file.cr>*
Omit debug symbols

`crystal` **build --static** *<file.cr>*
Link statically
Works best on Alpine/musl; glibc static linking is limited.

`crystal` **build -O***<level>* *<file.cr>*
Set optimization level 0-3, s, z

`crystal` **build -D** *<flag>* *<file.cr>*
Define a compile-time flag

`crystal` **build --cross-compile** *<file.cr>*
Emit .o and print the link command

EVAL AND PLAYGROUND

`crystal` **eval '***<code>***'**
Compile and run a code string

echo '*<code>***' | crystal eval**
Read code from stdin

`crystal` **eval --release '***<code>***'**
Optimized eval

`crystal` **play**
Start the browser playground

`crystal` **play --port** *<port>* **--binding** *<host>*
Set the playground address

`crystal` **version**
Print Crystal, LLVM and target triple

`crystal` **env [***<var>***]**
Print compiler environment variables

`crystal` **clear_cache**
Clear the compiler cache

PROJECTS

`crystal` **init app** *<name>*
Scaffold an application

`crystal` **init lib** *<name>*
Scaffold a library

init creates src/, spec/ and shard.yml. Commit shard.lock for applications, leave it out for libraries.

shards init
Create a shard.yml

SPECS

`crystal` **spec**
Run all specs in spec/

`crystal` **spec** *<file>*
Run one spec file

`crystal` **spec** *<file>*:*<line>*
Run the example at a line

`crystal` **spec -e '***<name>***'**
Run examples matching a name

`crystal` **spec --tag** *<tag>*
Run examples with a tag
Prefix the tag with ~ to exclude it.

`crystal` **spec --fail-fast**
Stop at the first failure

`crystal` **spec --order random**
Randomize example order

`crystal` **spec --profile**
Show the 10 slowest examples

`crystal` **spec -v**
Verbose, one line per example

Options after -- go to the spec runner; compiler flags such as --release go before the file list.

FORMAT AND TOOLS

`crystal` **tool format**
Format all files in place

`crystal` **tool format** *<path>*
Format a file or folder

`crystal` **tool format --check**
Fail if anything is unformatted (CI)

`crystal` **tool unreachable** *<file.cr>*
List methods that are never called

`crystal` **tool unreachable --check**
Exit nonzero if dead code exists

`crystal` **tool dependencies** *<file.cr>*
Show the file require graph

`crystal` **tool expand / context / types**
Macro expansion and type inspection

`crystal` **docs**
Generate API docs into ./docs

SHARDS

shards install
Install deps into lib/ per the lock

shards update
Re-resolve all deps, rewrite shard.lock

shards build
Install deps, build into bin/

shards build *<target>*
Build one named target

shards list
Show installed versions

shards check
Exit 1 if deps are missing

shards prune
Remove unused deps from lib/

shards install --production
Locked only, no dev deps, fail if out of sync

shard.yml

```
name: myapp
version: 0.1.0
targets:
  myapp:
    main: src/myapp.cr
dependencies:
  kemal:
    github: kemalcr/kemal
    version: ~> 1.6
development_dependencies:
  ameba:
    github: crystal-ameba/ameba
```

Add a dependency to shard.yml by hand, then run shards install; there is no shards add command. Then require it by shard name.

LANGUAGE BASICS

Types and methods

```
name = "Ada"
age : Int32 = 36
def greet(who : String) :
String
  "Hi, #{who}"
end
puts greet(name)
```

Classes and structs

```
class User
  property name : String
  getter id : Int32
  def initialize(@name, @id);
end
end
struct Point
  getter x : Int32, y : Int32
  def initialize(@x, @y); end
end
```

Blocks and collections

```
[1, 2, 3].map { |n| n * 2 }
h = {"a" => 1}
h.each do |k, v|
  puts "#{k}=#{v}"
end
def twice
  yield
  yield
end
```

Nil safety

```
# x is String?
x = ENV["HOME"]?
if x
  puts x.size
end
x.try &.size
x.not_nil!
```

require "./file" / require "json"
Load a relative file or stdlib module

puts / p / pp
Print, inspect, pretty inspect

Type? means Type | Nil. The compiler rejects calls on a possible nil; narrow with if, try, or not_nil! (raises at runtime).

Structs are value types and are copied on assignment; classes are heap references.

RECIPES

Start a new app

```
crystal init app hello
cd hello
crystal run src/hello.cr
crystal spec
```

CI gate

```
shards install --production
crystal tool format --check
crystal spec --fail-fast
shards build --release
```

Ship a static binary

```
crystal build --release --
static --no-debug \
  -o bin/app src/app.cr
```

Run one failing spec

```
crystal spec spec/
user_spec.cr:12
crystal spec -e 'rejects blank
name' --fail-fast
```