Zig Compiler and build system

CONCEPTS

zig is one binary: compiler, build system, formatter, test runner and a drop-in C/C++ cross compiler.

build.zig build script
↓ declares
Steps install, run, test
↓ build
Artifacts exe, lib, tests
↓ installs to
zig-out/ bin, lib

Build output goes to zig-out/ and the cache to .zig-cache/; both are safe to delete and belong in .gitignore.

Zig is pre-1.0: build.zig APIs and some flags change between releases, so check zig version first.

PROJECT

zig init
Scaffold build.zig, build.zig.zon and src/

zig init --minimal
Only write a stub build.zig and .zon

zig version
Print the compiler version

zig env
Show lib, std and cache directories

zig fetch --save *<url>*
Add a dependency to build.zig.zon

BUILD SYSTEM

zig build
Run the default install step

zig build run
Build and run the executable

zig build run -- *<args>*
Pass arguments to the program

zig build test
Run the project's test step

zig build -l
List the available steps

zig build --help
Show all options, including project -D ones

BUILD SYSTEM (cont.)

zig build *<step>*
Run a custom step from build.zig

zig build --summary all
Show every step, not just failures

zig build --watch
Rebuild when sources change

zig build -j<*N*>
Limit concurrent jobs

zig build -D<*name*>=<*value*>
Set a project option

zig build -fincremental
Use incremental compilation

OPTIMIZE AND TARGET

zig build -Doptimize=Debug
Default: safety checks on, no optimization

zig build -Doptimize=ReleaseSafe
Optimized, keeps safety checks

zig build -Doptimize=ReleaseFast
Fastest code, safety checks off

zig build -Doptimize=ReleaseSmall
Smallest binary

zig build -Dtarget=<*triple*>
Cross compile the whole project

zig build -Dtarget=x86_64-windows
Example: Windows 64-bit

zig targets
List supported targets and CPUs

SINGLE FILES

zig run *<file.zig>*
Compile and run a file

zig run *<file.zig>* **--** *<args>*
Run with program arguments

zig build-exe *<file.zig>*
Compile an executable

zig build-lib *<file.zig>*
Compile a static library

zig build-exe *<file.zig>* **-O ReleaseFast**
Set the optimization mode

zig build-exe *<file.zig>* **-target** *<triple>*
Cross compile one file

SINGLE FILES (cont.)

zig build-exe *<file.zig>* **-lc**
Link libc

zig build-exe *<file.zig>* **-l**<*lib*>
Link a system library

TEST AND FORMAT

zig test *<file.zig>*
Run the tests in a file

zig test *<file.zig>* **--test-filter** *<text>*
Run only matching tests

zig fmt *<path>*
Format files or directories in place

zig fmt --check *<path>*
List unformatted files, change nothing

C AND C++ TOOLCHAIN

zig cc *<file.c>* **-o** *<out>*
Compile C with the bundled clang

zig c++ *<file.cpp>* **-o** *<out>*
Compile C++

zig cc -target *<triple>* *<file.c>*
Cross compile C

zig cc -target x86_64-linux-musl *<file.c>*
Example: static Linux binary

zig translate-c *<file.h>*
Convert C to Zig on stdout

CC="zig cc" make
Use zig as the C compiler in a make build

RECIPES

Static Linux binary from any host

```
zig build -Dtarget=x86_64-linux-musl -Doptimize=ReleaseSafe
file zig-out/bin/<name>
```

Cross compile a C file for Windows

```
zig cc -target x86_64-windows-gnu hello.c -o hello.exe
```

Add a dependency

```
zig fetch --save <url>
# then in build.zig:
b.dependency("<name>", .{})
```