Meson Fast build system

START HERE

`meson setup build`
Configure into build/

`meson configure build`
Show options

`meson compile -C build`
Build everything

`meson configure build -D<opt>=<val>`
Change an option

`meson test -C build`
Run the tests

`meson setup build --wipe`
Reset the build dir

`meson install -C build`
Install the project

`meson introspect build --targets`
List targets as JSON

meson.build project definition → `meson setup` → **Build dir** options + Ninja files → `meson compile` → **Artifacts** targets → `meson test` → **Test results** testlog → `meson install` → **Install tree** prefix / DESTDIR

Configure once per build dir, then build, test and install from it

CONCEPTS

Meson generates build files (Ninja by default) and always builds out of source: meson.build stays in the source tree, every build directory is separate and can use different options.

Edits to meson.build trigger an automatic reconfigure on the next compile. Options set with -D are remembered in the build dir.

SETUP

`meson setup <builddir>`
Configure source in cwd into builddir

`meson setup <builddir> <srcdir>`
Configure another source dir

`meson setup <builddir> --buildtype=<type>`
debug, debugoptimized, release, plain

`meson setup <builddir> --prefix=<dir>`
Set the install prefix

`meson setup <builddir> -D<opt>=<val>`
Set a project or built-in option

`meson setup <builddir> --default-library=<kind>`
shared, static or both

`meson setup <builddir> --reconfigure`
Re-run configure with same options

`meson setup <builddir> --wipe`
Delete and recreate the build dir

`meson setup <builddir> --native-file <file>`
Select native toolchain/ properties

`meson setup <builddir> --cross-file <file>`
Cross-compile with a machine file

`meson init -l <lang> -n <name>`
Create a skeleton project

CONFIGURE

`meson configure <builddir>`
List all options and values

CONFIGURE (cont.)

`meson configure <builddir> -D<opt>=<val>`
Change an option later

`-Dwarning_level=<0-3|everything>`
Compiler warning level

`-Dwerror=true`
Treat warnings as errors

`-Db_sanitize=<kind>`
address, undefined, thread

BUILD

`meson compile -C <builddir>`
Build all targets

`meson compile -C <builddir> <target>`
Build one target

`meson compile -C <builddir> -j <N>`
Limit parallel jobs

`meson compile -C <builddir> -v`
Show full command lines

`meson compile -C <builddir> --clean`
Remove build outputs

TEST

`meson test -C <builddir>`
Run all tests

`meson test -C <builddir> --print-errorlogs`
Show output of failures

`meson test -C <builddir> <name>`
Run tests by name

`meson test -C <builddir> --list`
List tests without running

`meson test -C <builddir> --suite <suite>`
Run one suite

`meson test -C <builddir> --repeat <N>`
Repeat each test N times

`meson test -C <builddir> --gdb <name>`
Run one test under gdb

INSTALL AND DIST

`meson install -C <builddir>`
Install to the prefix

`meson install -C <builddir> --destdir <dir>`
Stage into a root dir

INSTALL AND DIST (cont.)

`meson install -C <builddir> --dry-run`
Show what would be installed

`meson dist -C <builddir>`
Create a release tarball

SUBPROJECTS AND WRAPS

`meson wrap search <name>`
Search WrapDB

`meson wrap install <name>`
Add a wrap file to subprojects/

`meson subprojects download`
Fetch all subprojects

`meson subprojects update`
Update subprojects to wrap revisions

Subprojects live in subprojects/ as a directory or a .wrap file. Use dependency('x', fallback: ...) or the wrap's [provide] section so the system library wins when present.

INTROSPECT AND TOOLS

`meson introspect <builddir> --targets`
List build targets (JSON)

`meson introspect <builddir> --buildoptions`
List all options (JSON)

`meson devenv -C <builddir>`
Shell with the build's environment

`meson --version`
Print the Meson version

MESON.BUILD

`project('<name>', 'c', version: '<x.y>')`
Declare project and languages

`executable('<name>', '<main.c>')`
Define an executable

`library('<name>', <sources>)`
Shared or static per default_library

`shared_library('<name>', <sources>)`
Always a shared library

MESON.BUILD (cont.)

`dependency('<name>')`
Find a lib via pkg-config/CMake

`dependency('<n>', required: false)`
Optional dependency

`include_directories('<dir>')`
Add include paths

`subdir('<dir>')`
Descend into a directory

`test('<name>', <exe>)`
Register a test

`install_headers('<file.h>')`
Install public headers

`configure_file(input: .., output: .., configuration: ..)`
Generate a file from a template

`declare_dependency(link_with: <lib>)`
Expose a library to parents

`meson_options.txt`
Project-defined options file

`option('<name>', type: 'boolean', value: true)`
Declare an option

Targets take 'dependencies:', 'link_with:' and 'include_directories:' keyword arguments. Use sources: files() lists rather than globbing; Meson deliberately has no glob.

RECIPES

Minimal project with a test

```
project('app', 'c', version: '0.1')
exe = executable('app', 'main.c', install: true)
test('smoke', exe)
```

Build, test, install

```
meson setup build --buildtype=release --prefix=/usr/local
meson compile -C build
meson test -C build --print-errorlogs
meson install -C build --destdir /tmp/stage
```

Library with a dependency

```
zdep = dependency('zlib')
lib = library('core', 'core.c', dependencies: zdep)
core_dep = declare_dependency(link_with: lib,
  include_directories: include_directories('.'))
```