Command-line arguments

Here's a list of command-line arguments to `rustc` and what they do.

`-h`/`--help`: get help

This flag will print out help information for `rustc`.

`--cfg`: configure the compilation environment

This flag can turn on or off various `#[cfg]` settings for conditional compilation.

The value can either be a single identifier or two identifiers separated by `=`.

For examples, `--cfg 'verbose'` or `--cfg 'feature="serde"'`. These correspond to `#[cfg(verbose)]` and `#[cfg(feature = "serde")]` respectively.

`-L`: add a directory to the library search path

The `-L` flag adds a path to search for external crates and libraries.

The kind of search path can optionally be specified with the form `-L KIND=PATH` where `KIND` may be one of:

- `dependency` — Only search for transitive dependencies in this directory.
- `crate` — Only search for this crate's direct dependencies in this directory.
- `native` — Only search for native libraries in this directory.
- `framework` — Only search for macOS frameworks in this directory.
- `all` — Search for all library kinds in this directory. This is the default if `KIND` is not specified.

`-l`: link the generated crate to a native library

Syntax: `-l [KIND[:MODIFIERS]=]NAME[:RENAME]`.

This flag allows you to specify linking to a specific native library when building a crate.

The kind of library can optionally be specified with the form `-l KIND=lib` where `KIND` may be one of:

- `dylib` — A native dynamic library.
- `static` — A native static library (such as a `.a` archive).
- `framework` — A macOS framework.

If the kind is specified, then linking modifiers can be attached to it. Modifiers are specified as a comma-delimited string with each modifier prefixed with either a `+` or `-` to indicate that the modifier is enabled or disabled, respectively. The last boolean value specified for a given modifier wins.
Example: `-l static:+whole-archive=mylib`.

The kind of library and the modifiers can also be specified in a `#[link]` attribute. If the kind is not specified in the `link` attribute or on the command-line, it will link a dynamic library if available, otherwise it will use a static library. If the kind is specified on the command-line, it will override the kind specified in a `link` attribute.

The name used in a `link` attribute may be overridden using the form `-l ATTR_NAME:LINK_NAME` where `ATTR_NAME` is the name in the `link` attribute, and `LINK_NAME` is the name of the actual library that will be linked.

Linking modifiers: `whole-archive`

This modifier is only compatible with the `static` linking kind. Using any other kind will result in a compiler error.

`+whole-archive` means that the static library is linked as a whole archive without throwing any object files away.

This modifier translates to `--whole-archive` for `ld`-like linkers, to `/WHOLEARCHIVE` for `link.exe`, and to `-force_load` for `ld64`. The modifier does nothing for linkers that don't support it.

The default for this modifier is `-whole-archive`.
NOTE: The default may currently be different when building dylibs for some targets, but it is not guaranteed.

`--crate-type`: a list of types of crates for the compiler to emit

This instructs `rustc` on which crate type to build. This flag accepts a comma-separated list of values, and may be specified multiple times. The valid crate types are:

- `lib` — Generates a library kind preferred by the compiler, currently defaults to `rlib`.
- `rlib` — A Rust static library.
- `staticlib` — A native static library.
- `dylib` — A Rust dynamic library.
- `cdylib` — A native dynamic library.
- `bin` — A runnable executable program.
- `proc-macro` — Generates a format suitable for a procedural macro library that may be loaded by the compiler.

The crate type may be specified with the `crate_type` attribute. The `--crate-type` command-line value will override the `crate_type` attribute.

More details may be found in the linkage chapter of the reference.

`--crate-name`: specify the name of the crate being built

This informs `rustc` of the name of your crate.

`--edition`: specify the edition to use

This flag takes a value of `2015`, `2018` or `2021`. The default is `2015`. More information about editions may be found in the edition guide.

`--emit`: specifies the types of output files to generate

This flag controls the types of output files generated by the compiler. It accepts a comma-separated list of values, and may be specified multiple times. The valid emit kinds are:

- `asm` — Generates a file with the crate's assembly code. The default output filename is `CRATE_NAME.s`.
- `dep-info` — Generates a file with Makefile syntax that indicates all the source files that were loaded to generate the crate. The default output filename is `CRATE_NAME.d`.
- `link` — Generates the crates specified by `--crate-type`. The default output filenames depend on the crate type and platform. This is the default if `--emit` is not specified.
- `llvm-bc` — Generates a binary file containing the LLVM bitcode. The default output filename is `CRATE_NAME.bc`.
- `llvm-ir` — Generates a file containing LLVM IR. The default output filename is `CRATE_NAME.ll`.
- `metadata` — Generates a file containing metadata about the crate. The default output filename is `libCRATE_NAME.rmeta`.
- `mir` — Generates a file containing rustc's mid-level intermediate representation. The default output filename is `CRATE_NAME.mir`.
- `obj` — Generates a native object file. The default output filename is `CRATE_NAME.o`.

The output filename can be set with the `-o` flag. A suffix may be added to the filename with the `-C extra-filename` flag. The files are written to the current directory unless the `--out-dir` flag is used. Each emission type may also specify the output filename with the form `KIND=PATH`, which takes precedence over the `-o` flag.

`--print`: print compiler information

This flag prints out various information about the compiler. This flag may be specified multiple times, and the information is printed in the order the flags are specified. Specifying a `--print` flag will usually disable the `--emit` step and will only print the requested information. The valid types of print values are:

- `crate-name` — The name of the crate.
- `file-names` — The names of the files created by the `link` emit kind.
- `sysroot` — Path to the sysroot.
- `target-libdir` - Path to the target libdir.
- `cfg` — List of cfg values. See conditional compilation for more information about cfg values.
- `target-list` — List of known targets. The target may be selected with the `--target` flag.
- `target-cpus` — List of available CPU values for the current target. The target CPU may be selected with the `-C target-cpu=val` flag.

- `target-features` — List of available target features for the current target. Target features may be enabled with the `-C target-feature=val` flag. This flag is unsafe. See known issues for more details.
- `relocation-models` — List of relocation models. Relocation models may be selected with the `-C relocation-model=val` flag.
- `code-models` — List of code models. Code models may be selected with the `-C code-model=val` flag.
- `tls-models` — List of Thread Local Storage models supported. The model may be selected with the `-Z tls-model=val` flag.
- `native-static-libs` — This may be used when creating a `staticlib` crate type. If this is the only flag, it will perform a full compilation and include a diagnostic note that indicates the linker flags to use when linking the resulting static library. The note starts with the text `native-static-libs:` to make it easier to fetch the output.
- `link-args` — This flag does not disable the `--emit` step. When linking, this flag causes `rustc` to print the full linker invocation in a human-readable form. This can be useful when debugging linker options. The exact format of this debugging output is not a stable guarantee, other than that it will include the linker executable and the text of each command-line argument passed to the linker.

`-g`: include debug information

A synonym for `-C debuginfo=2`.

`-O`: optimize your code

A synonym for `-C opt-level=2`.

`-o`: filename of the output

This flag controls the output filename.

`--out-dir`: directory to write the output in

The outputted crate will be written to this directory. This flag is ignored if the `-o` flag is used.

`--explain`: provide a detailed explanation of an error message

Each error of `rustc`'s comes with an error code; this will print out a longer explanation of a given error.

`--test`: build a test harness

When compiling this crate, `rustc` will ignore your `main` function and instead produce a test harness. See the Tests chapter for more information about tests.

`--target`: select a target triple to build

This controls which target to produce.

`-W`: set lint warnings

This flag will set which lints should be set to the warn level.

Note: The order of these lint level arguments is taken into account, see lint level via compiler flag for more information.

`-A`: set lint allowed

This flag will set which lints should be set to the allow level.

Note: The order of these lint level arguments is taken into account, see lint level via compiler flag for more information.

`-D`: set lint denied

This flag will set which lints should be set to the deny level.

Note: The order of these lint level arguments is taken into account, see lint level via compiler flag for more information.

`-F`: set lint forbidden

This flag will set which lints should be set to the forbid level.

Note: The order of these lint level arguments is taken into account, see lint level via compiler flag for more information.

`-Z`: set unstable options

This flag will allow you to set unstable options of rustc. In order to set multiple options, the -Z flag can be used multiple times. For example: `rustc -Z verbose -Z time`. Specifying options with -Z is only available on nightly. To view all available options run: `rustc -Z help`.

`--cap-lints`: set the most restrictive lint level

This flag lets you 'cap' lints, for more, see here.

`-C`/`--codegen`: code generation options

This flag will allow you to set codegen options.

`-V`/`--version`: print a version

This flag will print out `rustc`'s version.

`-v`/`--verbose`: use verbose output

This flag, when combined with other flags, makes them produce extra output.

`--extern`: specify where an external library is located

This flag allows you to pass the name and location for an external crate of a direct dependency. Indirect dependencies (dependencies of dependencies) are located using the `-L` flag. The given crate name is added to the extern prelude, similar to specifying `extern crate` within the root module. The given crate name does not need to match the name the library was built with.

Specifying `--extern` has one behavior difference from `extern crate`: `--extern` merely makes the crate a *candidate* for being linked; it does not actually link it unless it's actively used. In rare occasions you may wish to ensure a crate is linked even if you don't actively use it from your code: for example, if it changes the global allocator or if it contains `#[no_mangle]` symbols for use by other programming languages. In such cases you'll need to use `extern crate`.

This flag may be specified multiple times. This flag takes an argument with either of the following formats:

- `CRATENAME=PATH` — Indicates the given crate is found at the given path.
- `CRATENAME` — Indicates the given crate may be found in the search path, such as within the sysroot or via the `-L` flag.

The same crate name may be specified multiple times for different crate types. If both an `rlib` and `dylib` are found, an internal algorithm is used to decide which to use for linking. The `-C prefer-dynamic` flag may be used to influence which is used.

If the same crate name is specified with and without a path, the one with the path is used and the pathless flag has no effect.

`--sysroot`: Override the system root

The "sysroot" is where `rustc` looks for the crates that come with the Rust distribution; this flag allows that to be overridden.

`--error-format`: control how errors are produced

This flag lets you control the format of messages. Messages are printed to stderr. The valid options are:

- `human` — Human-readable output. This is the default.
- `json` — Structured JSON output. See the JSON chapter for more detail.
- `short` — Short, one-line messages.

`--color`: configure coloring of output

This flag lets you control color settings of the output. The valid options are:

- `auto` — Use colors if output goes to a tty. This is the default.
- `always` — Always use colors.
- `never` — Never colorize output.

`--remap-path-prefix`: remap source names in output

Remap source path prefixes in all output, including compiler diagnostics, debug information, macro expansions, etc. It takes a value of the form `FROM=TO` where a path prefix equal to `FROM` is rewritten to the value `TO`. The `FROM` may itself contain an `=` symbol, but the `TO` value may not. This flag may be specified multiple times.

This is useful for normalizing build products, for example by removing the current directory out of pathnames emitted into the object files. The replacement is purely textual, with no consideration of the current system's pathname syntax. For example `--remap-path-prefix foo=bar` will match `foo/lib.rs` but not `./foo/lib.rs`.

`--json`: configure json messages printed by the compiler

When the `--error-format=json` option is passed to rustc then all of the compiler's diagnostic output will be emitted in the form of JSON blobs. The `--json` argument can be used in conjunction with `--error-format=json` to configure what the JSON blobs contain as well as which ones are emitted.

With `--error-format=json` the compiler will always emit any compiler errors as a JSON blob, but the following options are also available to the `--json` flag to customize the output:

- `diagnostic-short` - json blobs for diagnostic messages should use the "short" rendering instead of the normal "human" default. This means that the output of `--error-format=short` will be embedded into the JSON diagnostics instead of the default `--error-format=human`.
- `diagnostic-rendered-ansi` - by default JSON blobs in their `rendered` field will contain a plain text rendering of the diagnostic. This option instead indicates that the diagnostic should have embedded ANSI color codes intended to be used to colorize the message in the manner rustc typically already does for terminal outputs. Note that this is usefully combined with crates like `fwdansi` to translate these ANSI codes on Windows to console commands or `strip-ansi-escapes` if you'd like to optionally remove the ansi colors afterwards.
- `artifacts` - this instructs rustc to emit a JSON blob for each artifact that is emitted. An artifact corresponds to a request from the `--emit` CLI argument, and as soon as the artifact is available on the filesystem a notification will be emitted.

Note that it is invalid to combine the `--json` argument with the `--color` argument, and it is required to combine `--json` with `--error-format=json`.

See the JSON chapter for more detail.

`@path`: load command-line flags from a path

If you specify `@path` on the command-line, then it will open `path` and read command line options from it. These options are one per line; a blank line indicates an empty option. The file can use Unix or Windows style line endings, and must be encoded as UTF-8.