`check-cfg`

The tracking issue for this feature is: #82450.

This feature allows you to enable complete or partial checking of configuration.

`rustc` accepts the `--check-cfg` option, which specifies whether to check conditions and how to check them. The `--check-cfg` option takes a value, called the *check cfg specification*. The check cfg specification is parsed using the Rust metadata syntax, just as the `--cfg` option is.

`--check-cfg` option can take one of two forms:

1. `--check-cfg names(...)` enables checking condition names.
2. `--check-cfg values(...)` enables checking the values within list-valued conditions.

These two options are independent. `names` checks only the namespace of condition names while `values` checks only the namespace of the values of list-valued conditions.

The `names(...)` form

The `names(...)` form enables checking the names. This form uses a named list:

```
rustc --check-cfg 'names(name1, name2, ... nameN)'
```

where each `name` is a bare identifier (has no quotes). The order of the names is not significant.

If `--check-cfg names(...)` is specified at least once, then `rustc` will check all references to condition names. `rustc` will check every `#[cfg]` attribute, `#[cfg_attr]` attribute, `cfg` clause inside `#[link]` attribute and `cfg!(...)` call against the provided list of expected condition names. If a name is not present in this list, then `rustc` will report an `unexpected_cfgs` lint diagnostic. The default diagnostic level for this lint is `Warn`.

If `--check-cfg names(...)` is not specified, then `rustc` will not check references to condition names.

`--check-cfg names(...)` may be specified more than once. The result is that the list of valid condition names is merged across all options. It is legal for a condition name to be specified more than once; redundantly specifying a condition name has no effect.

To enable checking condition names with an empty set of valid condition names, use the following form. The parentheses are required.

```
rustc --check-cfg 'names()'
```

Note that `--check-cfg 'names()'` is *not* equivalent to omitting the option entirely. The first form enables checking condition names, while specifying that there are no valid condition names (outside of the set of well-known names defined by `rustc`). Omitting the `--check-cfg 'names(...)'` option does not enable checking condition names.

Conditions that are enabled are implicitly valid; it is unnecessary (but legal) to specify a condition name as both enabled and valid. For example, the following invocations are equivalent:

```
# condition names will be checked, and 'has_time_travel' is valid
rustc --cfg 'has_time_travel' --check-cfg 'names()'

# condition names will be checked, and 'has_time_travel' is valid
rustc --cfg 'has_time_travel' --check-cfg 'names(has_time_travel)'
```

In contrast, the following two invocations are *not* equivalent:

```
# condition names will not be checked (because there is no --check-cfg names(...))
rustc --cfg 'has_time_travel'

# condition names will be checked, and 'has_time_travel' is both valid and enabled.
rustc --cfg 'has_time_travel' --check-cfg 'names(has_time_travel)'
```

The `values(...)` form

The `values(...)` form enables checking the values within list-valued conditions. It has this form:

```
rustc --check-cfg `values(name, "value1", "value2", ... "valueN")'
```

where `name` is a bare identifier (has no quotes) and each `"value"` term is a quoted literal string. `name` specifies the name of the condition, such as `feature` or `target_os`.

When the `values(...)` option is specified, `rustc` will check every `#[cfg(name = "value")]` attribute, `#[cfg_attr(name = "value")]` attribute, `#[link(name = "a", cfg(name = "value"))]` and `cfg!(name = "value")` call. It will check that the `"value"` specified is present in the list of expected values. If `"value"` is not in it, then `rustc` will report an `unexpected_cfgs` lint diagnostic. The default diagnostic level for this lint is `Warn`.

To enable checking of values, but to provide an empty set of valid values, use this form:

```
rustc --check-cfg `values(name)`
```

The `--check-cfg values(...)` option can be repeated, both for the same condition name and for different names. If it is repeated for the same condition name, then the sets of values for that condition are merged together.

If `values()` is specified, then `rustc` will enable the checking of well-known values defined by itself. Note that it's necessary to specify the `values()` form to enable the checking of well known values, specifying the other forms doesn't implicitly enable it.

Examples

Consider this command line:

```
rustc --check-cfg 'names(feature)' \
      --check-cfg 'values(feature, "lion", "zebra")' \
      --cfg 'feature="lion"' -Z unstable-options \
      example.rs
```

This command line indicates that this crate has two features: `lion` and `zebra`. The `lion` feature is enabled, while the `zebra` feature is disabled. Consider compiling this code:

```
// This is expected, and tame_lion() will be compiled
#[cfg(feature = "lion")]
fn tame_lion(lion: Lion) {}

// This is expected, and ride_zebra() will NOT be compiled.
#[cfg(feature = "zebra")]
fn ride_zebra(zebra: Zebra) {}

// This is UNEXPECTED, and will cause a compiler warning (by default).
#[cfg(feature = "platypus")]
fn poke_platypus() {}

// This is UNEXPECTED, because 'feechure' is not a known condition name,
// and will cause a compiler warning (by default).
#[cfg(feechure = "lion")]
fn tame_lion() {}
```

> *Note: The `--check-cfg names(feature)` option is necessary only to enable checking the condition name, as in the last example. `feature` is a well-known (always-expected) condition name, and so it is not necessary to specify it in a `--check-cfg 'names(...)'` option. That option can be shortened to > `--check-cfg names()` in order to enable checking well-known condition names.*

Example: Checking condition names, but not values

```
# This turns on checking for condition names, but not values, such as 'feature'
values.
rustc --check-cfg 'names(is_embedded, has_feathers)' \
      --cfg has_feathers --cfg 'feature = "zapping"' -Z unstable-options
```

```
#[cfg(is_embedded)]          // This is expected as "is_embedded" was provided in
names()
fn do_embedded() {}

#[cfg(has_feathers)]         // This is expected as "has_feathers" was provided in
names()
fn do_features() {}

#[cfg(has_mumble_frotz)]     // This is UNEXPECTED because names checking is enable
and
                             // "has_mumble_frotz" was not provided in names()
fn do_mumble_frotz() {}

#[cfg(feature = "lasers")]   // This doesn't raise a warning, because values checking
for "feature"
```

```
                            // was never used
fn shoot_lasers() {}
```

Example: Checking feature values, but not condition names

```
# This turns on checking for feature values, but not for condition names.
rustc --check-cfg 'values(feature, "zapping", "lasers")' \
      --cfg 'feature="zapping"' -Z unstable-options
```

```
#[cfg(is_embedded)]         // This is doesn't raise a warning, because names
checking was not
                            // enable (ie not names())
fn do_embedded() {}

#[cfg(has_feathers)]        // Same as above, --check-cfg names(...) was never used
so no name
                            // checking is performed
fn do_features() {}


#[cfg(feature = "lasers")]  // This is expected, "lasers" is in the values(feature)
list
fn shoot_lasers() {}

#[cfg(feature = "monkeys")] // This is UNEXPECTED, because "monkeys" is not in the
                            // --check-cfg values(feature) list
fn write_shakespeare() {}
```

Example: Checking both condition names and feature values

```
# This turns on checking for feature values and for condition names.
rustc --check-cfg 'names(is_embedded, has_feathers)' \
      --check-cfg 'values(feature, "zapping", "lasers")' \
      --cfg has_feathers --cfg 'feature="zapping"' -Z unstable-options
```

```
#[cfg(is_embedded)]         // This is expected because "is_embedded" was provided
in names()
fn do_embedded() {}

#[cfg(has_feathers)]        // This is expected because "has_feathers" was provided
in names()
fn do_features() {}

#[cfg(has_mumble_frotz)]    // This is UNEXPECTED, because has_mumble_frotz is not
in the
                            // --check-cfg names(...) list
fn do_mumble_frotz() {}
```

```rust
#[cfg(feature = "lasers")]  // This is expected, "lasers" is in the values(feature)
list
fn shoot_lasers() {}

#[cfg(feature = "monkeys")] // This is UNEXPECTED, because "monkeys" is not in
                            // the values(feature) list
fn write_shakespear() {}
```