Makefile - Directives

There are numerous directives available in various forms. The **make** program on your system may not support all the directives. So please check if your **make** supports the directives we are explaining here. **GNU make** supports these directives.

Conditional Directives

The conditional directives are −

- The **ifeq** directive begins the conditional, and specifies the condition. It contains two arguments, separated by a comma and surrounded by parentheses. Variable substitution is performed on both arguments and then they are compared. The lines of the makefile following the ifeq are obeyed if the two arguments match; otherwise they are ignored.
- The **ifneq** directive begins the conditional, and specifies the condition. It contains two arguments, separated by a comma and surrounded by parentheses. Variable substitution is performed on both arguments and then they are compared. The lines of the makefile following the ifneq are obeyed if the two arguments do not match; otherwise they are ignored.
- The **ifdef** directive begins the conditional, and specifies the condition. It contains single argument. If the given argument is true then condition becomes true.
- The **ifndef** directive begins the conditional, and specifies the condition. It contains single argument. If the given argument is false then condition becomes true.
- The **else** directive causes the following lines to be obeyed if the previous conditional failed. In the example above this means the second alternative linking command is used whenever the first alternative is not used. It is optional to have an else in a conditional.
- The **endif** directive ends the conditional. Every conditional must end with an endif.

Syntax of Conditionals Directives

The syntax of a simple conditional with no else is as follows −

```
conditional-directive
   text-if-true
endif
```

The text-if-true may be any lines of text, to be considered as part of the makefile if the condition is true. If the condition is false, no text is used instead.

The syntax of a complex conditional is as follows −

```
conditional-directive
   text-if-true
else
   text-if-false
endif
```

If the condition is true, text-if-true is used; otherwise, text-if-false is used. The text-if-false can be any number of lines of text.

The syntax of the conditional-directive is the same whether the conditional is simple or complex. There are four different directives that test various conditions. They are as given −

```
ifeq (arg1, arg2)
ifeq 'arg1' 'arg2'
ifeq "arg1" "arg2"
ifeq "arg1" 'arg2'
ifeq 'arg1' "arg2"
```

Opposite directives of the above conditions are are follows −

```
ifneq (arg1, arg2)
ifneq 'arg1' 'arg2'
ifneq "arg1" "arg2"
ifneq "arg1" 'arg2'
ifneq 'arg1' "arg2"
```

Example of Conditionals Directives

```
libs_for_gcc = -lgnu
normal_libs =

foo: $(objects)
ifeq ($(CC),gcc)
   $(CC) -o foo $(objects) $(libs_for_gcc)
else
   $(CC) -o foo $(objects) $(normal_libs)
endif
```

The include Directive

The **include directive** allows **make** to suspend reading the current makefile and read one or more other makefiles before continuing. The directive is a line in the makefile that looks follows −

```
include filenames...
```

The filenames can contain shell file name patterns. Extra spaces are allowed and ignored at the beginning of the line, but a tab is not allowed. For example, if you have three `.mk' files, namely, `a.mk', `b.mk', and `c.mk', and $(bar) then it expands to bish bash, and then the following expression.

```
include foo *.mk $(bar)

is equivalent to:

include foo a.mk b.mk c.mk bish bash
```

When the **make** processes an include directive, it suspends reading of the makefile and reads from each listed file in turn. When that is finished, **make** resumes reading the makefile in which the directive appears.

The override Directive

If a variable has been set with a command argument, then ordinary assignments in the makefile are ignored. If you want to set the variable in the makefile even though it was set with a command argument, you can use an override directive, which is a line that looks follows−

```
override variable = value

or

override variable := value
```