

GNU Make Cheat Sheet

RULES

Short rules can be written as below

```
target(s): [prerequisites] [; shell-command(s)]
target(s): [prerequisites]
           [shell-command]
```

```
%.class: %.java; javac $<
```

A target pattern is composed of a “%” between a prefix and a suffix, either or both of which may be empty.

MACROS/VARIABLES

Values assigned to variables with `=` are expanded when used (like a reference) while values assigned with `:=` are expanded at declaration time. `$(myvar)` returns the stored value and `$(call myvar)` executes the value (like a function).

<code>\$(</code>	The name of the target.
<code>\$(</code>	The target member name, when the target is an archive member.
<code>\$<</code>	The name of the first (or only) prerequisite.
<code>\$?</code>	Space separated list of all prerequisites newer than the target.
<code>\$^ \$+</code>	Space separated list of all prerequisites. <code>\$^</code> omits duplicate prerequisites, while <code>\$+</code> retains them and preserves their order.
<code>\$(</code>	The stem with which an implicit rule matches.
<code>\$((D)) \$((F))</code>	The path and the file name of a macro. For instance <code>\$((D))</code> returns the directory-part of <code>\$(</code> .

INCLUDES

For large projects use several makefiles and use the command `include` to call them within your ‘master’ makefile. Use `-include Makefile(s)` if you don’t want `make` to abort when the included Makefile’s missing. The minus sign generally forces `make` to ignore errors.

FUNCTIONS

<code>\$(subst from,to,text)</code>	Replaces each occurrence of <code>from</code> in <code>text</code> by <code>to</code>
<code>\$(patsubst pattern,replacement,text)</code>	Replaces words matching <code>pattern</code> with <code>replacement</code> in <code>text</code> .
<code>\$(strip string)</code>	Removes excess whitespace characters from <code>string</code> .
<code>\$(findstring find,in)</code>	Search <code>in</code> for an occurrence of <code>find</code> .
<code>\$(filter pattern_1 pattern_2...,text)</code>	Selects words in <code>text</code> that match one of the <code>pattern</code> words.
<code>\$(filter-out pattern_1 pattern_2...,text)</code>	Selects words in <code>text</code> that do not match any of the <code>pattern</code> words.
<code>\$(sort list)</code>	Sorts the words in <code>list</code> lexicographically, removing duplicates.
<code>\$(dir names...)</code>	Extracts the directory-part of each file name in <code>names</code> .
<code>\$(notdir names...)</code>	Extracts the non-directory part of each file name in <code>names</code> .
<code>\$(realpath names...)</code>	Returns an absolute name (does not contain “.” or “..” or symlinks) for each file name in <code>names</code> .
<code>\$(suffix names...)</code>	Extracts the suffix (everything starting with the last period) of each file name in <code>names</code> .
<code>\$(basename names...)</code>	Extracts all but the suffix of each file name in <code>names</code> .
<code>\$(addsuffix suffix,names...)</code>	Appends <code>suffix</code> to each word in <code>names</code> .
<code>\$(addprefix prefix,names...)</code>	Prepends <code>prefix</code> to each word in <code>names</code> .
<code>\$(join list_1,list_2)</code>	Join two parallel lists of words.
<code>\$(word n,text)</code>	Extracts the n^{th} word of <code>text</code> .
<code>\$(words text)</code>	Counts the number of words in <code>text</code> .
<code>\$(wordlist i,j,text)</code>	Returns the list of words in <code>text</code> from <code>i</code> to <code>j</code> .
<code>\$(firstword names...)</code>	Extracts the first word in <code>names</code> .
<code>\$(wildcard pattern...)</code>	Returns the file names matching (a shell file name) <code>pattern</code> (not a “%” pattern).
<code>\$(error text...)</code>	When the function is evaluated a fatal error with the message <code>text</code> is generated.
<code>\$(warning text...)</code>	When the function is evaluated a warning with the message <code>text</code> is generated.
<code>\$(shell command)</code>	Execute a shell command and return its output.
<code>\$(origin variable)</code>	Describes where <code>variable</code> came from. Do not use “\$” or parentheses around <code>variable</code> unless you want the name not to be constant and provide a variable reference.
<code>\$(foreach var,words,text)</code>	Evaluate <code>text</code> with <code>var</code> bound to each word in <code>words</code> , and concatenate the results.
<code>\$(call var,param,param,...)</code>	Evaluate <code>var</code> replacing any references to <code>\$(1)</code> , <code>\$(2)</code> with the first, second, etc. <code>param</code> values.

BUILT-IN TARGET NAMES

<code>.PHONY</code>	prerequisites of a <code>.PHONY</code> target are never considered to be up to date.
<code>.INTERMEDIATE</code>	the targets which <code>.INTERMEDIATE</code> depends on are treated as intermediate files and they are deleted once no longer needed for a rule.
<code>.SECONDARY</code>	the targets which <code>.SECONDARY</code> depends on are treated as intermediate files but they are not automatically deleted.
<code>.IGNORE</code>	errors encountered while executing recipes for the prerequisites of <code>.IGNORE</code> are ignored.
<code>.EXPORT_ALL_VARIABLES</code>	allows variables exported in parent Makefile to be available to rules in child processes. It uses no prerequisites.