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).

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

SPECIAL BUILT-IN TARGET NAMES

`.PHONY` prerequisites of a `.PHONY` target are never considered to be up to date.

`.INTERMEDIATE` the targets which `.INTERMEDIATE` depends on are treated as intermediate files and they are deleted once no longer needed for a rule.

`.SECONDARY` the targets which `.SECONDARY` depends on are treated as intermediate files but they are not automatically deleted.

`.PRECIOUS` like `.SECONDARY`, and files will not be deleted if `make` is aborted.

`.IGNORE` errors encountered while executing recipes for the prerequisites of `.IGNORE` are ignored.

`.EXPORT_ALL_VARIABLES` allows variables exported in parent Makefile to be available to rules in child processes. It uses no prerequisites.

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 Makefiles are missing. The minus sign generally forces `make` to ignore errors.

FUNCTIONS

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

Inspired by `http://www.schacherer.de/frank/technology/tools/make.html` and `https://www.gnu.org/software/make/manual/`
Created by Maria Xenophontos after many valuable comments from Konrad Rudolph