Variables

`[define] variable`	Define

`[define] variable =`
`[define] variable :=`
`[define] variable ::=`
	Assignment
	`=` recursively expanded
	`:=,::=` simply expanded

`[define] variable +=`	Append
`[define] variable ?=`	Only if not exists
`endef`	Multi-line closure
`undefine variable`	Undefinition
`override var-assign`	Explicitly overrides all
`export`	All variables to child process
`export <var|var-assign>`	To child process
`Unexport variable`	Not export
`private var-assign`	Don't inherited by prerequisites

Directives

`include file` `<-include|sinclude> file`	Inclusion
`vpath [pattern] [path]`	Search paths specification|removal
`<ifdef| ifndef> variable`	Value not empty (It doesn't expand)

`<ifeq|ifneq> (a,b)`
`<ifeq|ifneq> "a" "b"`
`<ifeq|ifneq> 'a' 'b'`
	Equality

`else`	Else statement
`endif`	Closure

Functions

`$(subst f,to,t)`	Replace *f* with *to* in *t*
`$(patsubst p,r,t)` `$(t:p=r)`	Replace words matching *p* with *r* in *t*
`$(strip s)`	Remove excess spaces from *s*
`$(findstring s,t)`	Locate *s* in *t*
`$(filter p…,t)`	Words in *t* that match one *p* words
`$(filter-out p…,t)`	Words in *t* that *don't* match *p* words
`$(sort l)`	Sort *l* lexicographically, removes dup.
`$(word n,t)`	Extract the *n*th word (one-origin) of *t*
`$(words t)`	Count the number of words in *t*
`$(wordlist t,s,e)`	List of words in *t* from *s* to *e*
`$(firstword ns…)`	Extract the first word of *ns*
`$(lastword ns…)`	Extract the last word of *ns*
`$(dir ns…)`	Directory part of each file name
`$(notdir ns…)`	Non-directory part of each file name
`$(suffix ns…)`	Deletes until the last '.' in every *ns*
`$(basename ns…)`	Base name (no suffix) in every *ns*
`$(addsuffix sf,ns…)`	Append *sf* to each word in *ns*
`$(addprefix pf,ns…)`	Prepend *pf* to each word in *ns*
`$(join l1,l2)`	Join two parallel lists of words
`$(wildcard p…)`	Find files matching a pattern (*not* '%')
`$(realpath ns…)`	Absolute name (no ., .., nor symlinks)
`$(abspath ns…)`	Absolute name (no. or ..)Preserves symlinks
`$(error t…)`	make fatal error with the message *t*
`$(warning t…)`	make warning with the message *t*
`$(info t…)`	make info with the message *t*
`$(shell c)`	Execute a shell cmd, returns output
`$(origin v)`	Origin of variable *v*
`$(flavor v)`	Flavor of variable *v*
`$(foreach v,w,t)`	Evaluate *t* with *v* bound to each word in *w*, and concatenate the results
`$(if c,then-part[,else-part])`	Evaluates *c*; if it's non-empty substitute by *then-part* otherwise by *else-part*
`$(or c1[,c2[,c3…]])`	Evaluate each *cN*; substitute the first non-empty expansion.
`$(and c1[,c2[,c3…]])`	Evaluate each *cN*; if any is empty substitution is empty. Expansion of the last *condition otherwise*
`$(call v,p,…)`	Evaluates *v* replacing any references to $(1), $(2) with the first, second, etc. *p* values
`$(eval t)`	Evaluate *t* and read the results as makefile commands
`$(file op f,t)`	Open the file *f* using mode *op* and write *t* to that file
`$(value v)`	Evaluates *v*, with no expansion

Special targets (T: As target, P: As prerequisite)

`.PHONY`	T: make runs it's prereq. unconditionally
`.SUFFIXES`	T: list of suffixes used in checking for suffix rules
`.DEFAULT`	T: Used for any target for which no rules are found
`.PRECIOUS`	P: Target and intermediate files are not deleted
`.INTERMEDIATE`	P: treated as intermediate files
`.SECONDARY`	P: target is intermediate but not deleted
`.SECONDEXPANSION`	T: prerequisites expand twice
`.DELETE_ON_ERROR`	T: make will delete the target of a rule if it has changed and its recipe exits with a nonzero exit status
`.IGNORE`	P: Ignore errors T: Ignore all errors
`.LOW_RESOLUTION_TIME`	P: Considered as generates low resolution time stamps
`.SILENT`	P: Muted T: Mute all
`.EXPORT_ALL_VARIABLES`	T: as export
`.NOTPARALLEL`	T: make will run serially
`.ONESHELL`	As target: All lines of a recipe runs on one shell
`.POSIX`	As target: makefile will be parsed and run in POSIX-conforming mode

Special variables

`MAKEFILE_LIST`	List of parsed makefiles
`.DEFAULT_GOAL`	Defines the default target (instead the first)
`MAKE_RESTARTS`	Number of restarts
`MAKE_TERMOUT`	stdout terminal
`MAKE_TERMERR`	stderr terminal
`.RECIPEPREFIX`	Recipe prefix instead of tab
`.VARIABLES`	List of global variable names (Read only)
`.FEATURES`	List of features (Read only)
`.INCLUDE_DIRS`	Makefiles directories inclusion
`MAKEFILES`	Makefiles to be read
`VPATH`	Directory search path
`SHELL`	System default command interpreter
`MAKESHELL`	(MS-DOS only) command interpreter
`MAKE`	Name with which make was invoked
`MAKE_VERSION`	Version number of the GNU make program
`MAKE_HOST`	Host that GNU make was built to run on
`MAKELEVEL`	Number of levels of recursion
`MAKEFLAGS`	Flags given to make
`GNUMAKEFLAGS`	Other flags parsed by make
`MAKECMDGOALS`	Targets given to make
`CURDIR`	Absolute pathname of the CWD
`SUFFIXES`	Suffixes before make
`.LIBPATTERNS`	Naming of the libraries make searches for

Automatic variables

`$@`	`$(@D)`	`$(@F)`	Name of the target
`$%`	`$(%D)`	`$(%F)`	Target member name, when target is an archive member
`$<`	`$(<D)`	`$(<F)`	First prerequisite
`$?`	`$(?D)`	`$(?F)`	Prerequisites newer than the target
`$^` `$+`	`$(^D)` `$(+D)`	`$(^F)` `$(+F)`	Names of all the prerequisites, with spaces between them. $^ omits duplicates
`$*`	`$(*D)`	`$(*F)`	Implicit rule stem

Rules and prerequisites

`target: ps | oops` `  recipes`	*recipes* for the *target* file will run if any *ps* is modified or rebuilt. *oops* are treated as *ps* but they will not trigger the *recipe* if those are modified.
`<pf>%<sf>: ps | oops` `  recipes`	Implicit rule. Same as explicit. % represents any string. Matching of % is called stem.
`target-pat: var-assign`	Target/pattern specific variables

contact@eduardolezcano.com – edupo@github