

## Invoking GNU Make

To invoke GNU make type the following command line:

```
make [-f makefile-name] [options] [targets]
```

The following file names will be searched for in the current directory automatically: GNUMakefile, Makefile, makefile.

By default, the first target will be invoked if not target are given.

## Rule

```
target : dependency [dependency ...]
        command
        [command]
```

where `target` is the result of the operation, `command` are the recipes to execute and `dependency` is the input of the operation. Beware of tabulations before commands!

## Dependency between rules

```
target : target1 target2
        ...
target1 : dependencies_1
        ...
target_2 : dependencies_2
        ...
```

make program builds a *dependency-tree* from these rules.

## Standard target names

Mainly taken from autotools, please use them:

<code>all</code>	build application
<code>install</code>	install what needs to be installed
<code>clean</code>	erase all files built by make <code>all</code>
<code>distclean</code>	erase also all configuration files

## Built-in target names

<code>.PHONY</code>	define targets which are not files (e.g. <code>clean</code> )
<code>.DEFAULT</code>	the default target
<code>.IGNORE</code>	ignore errors in prerequisites of this rule

## Variables

```
HEADER = prg.h
FILES  = $(HEADER)
```

then `$(FILES)` is expanded to `prg.h` running make program.

## Automatic variables

<code>\$\$</code>	the file name of the target of the rule
<code>\$(&lt;)</code>	the name of the first prerequisite
<code>\$(^)</code>	the names of all the prerequisites

`$(XD)` and `$(XF)` can be used to extract the directory and the file part of the name corresponding to `$(X)`. For instance, if `$$` is `src/foo.c`, then `$(@D)` is `src` and `$(@F)` is `foo.c`

## Suffix rules (aka pattern rules)

Rules used to process a dependency of two given types of files (defined by extensions).

```
%.o : %.c
        gcc -c $<
```

This will compile any C code, supposing it's extension is `.c`. Some rules are built-in, for instance for C compilation.

## Including another makefile

```
include PATH_TO_MAKEFILE
```

## Using functions

Functions can be called from a Makefile. To call a function `foo` with arguments `x` and `y`:

```
$(foo x,y)
```

Functions already defined (more to find in [1]):

<code>\$(subst FROM,TO,TEXT)</code>	replaces all occurrences of <code>FROM</code> by <code>TO</code> in <code>TEXT</code>
<code>\$(suffix NAMES...)</code>	extract the suffix of each file names in <code>NAMES</code>
<code>\$(suffix NAMES...)</code>	extract the suffix of each file names in <code>NAMES</code>
<code>\$(basename NAMES...)</code>	extract all but the suffix of each file names in <code>NAMES</code>

## Using conditionals

Using conditionals with the following constructs:

<code>ifeq (ARG1, ARG2)</code>	<code>ARG1</code> equals to <code>ARG2</code> ?
<code>ifneq (ARG1, ARG2)</code>	<code>ARG1</code> not equals to <code>ARG2</code> ?
<code>ifdef VAR-NAME</code>	<code>VAR-NAME</code> is defined?
<code>ifndef VAR-NAME</code>	<code>VAR-NAME</code> is not defined?

For instance in a command:

```
ifndef PROXY
    PROXY = proxy.isae.fr
endif
```

Conditional functions can be used (particularly in a functional context):

```
$(if CONDITION,THEN-PART[,ELSE-PART])
$(or CONDITION1,CONDITION2[,CONDITION3...])
$(and CONDITION1,CONDITION2[,CONDITION3...])
```

## References

[1] Free Software Foundation. *GNU Make*. 2014. URL: <http://www.gnu.org/software/make/>.

## Using shell for loop

To use shell for loop in a recipe, do not forget to add `\` at the end of each lines and to use `$$` to get the variables values:

```
target:
    for number in 1 2 3 4 ; do \
        echo $$number \
    done
```

## Foreach loop

The **foreach** function can be used to repeatedly use a piece of text:

```
$(foreach VAR,LIST,TEXT)
```

The following example sets the variable `C_FILES` to the list of all files with suffix `.c` in the directories specified in the list `DIRS` (the **wildcard** function allows to use wildcards in file names):

```
DIRS = ./src ./tests
C_FILES = $(foreach dir, DIRS, $(wildcard $(dir)/*.c))
```

## Ignoring errors in command

Put `"-"` before command to ignore potential errors, e.g.

```
clean :
    - rm *.o
```