NOTES ON MAKEFILES

Frédéric GALLIANO

Université Paris-Saclay, Université Paris Cité, CEA, CNRS, AIM, 91191, Gif-sur-Yvette, France

January 5, 2024

Contents

---

1 GENERAL

Makefiles are files performing certain actions useful to compile a program or library, archive a document, update a website, *etc.* These actions are executed only depending on the last modification date of the various source files.

A makefile is composed of several rules with the form:

```
<cible>: <dependency>
         <commands>
```

The commands must be preceded by a TAB. Comments are indicated with `#`.

1.1 Executing

The general command to execute a makefile is:

```
make -f <mymakefile>.mk <rules>
```

- If no file name is given, `make` will look for a `makefile` file in the current directory.
- If no rule is given, `make` executes the first rule in the file.

1.2 Rule Intrication

- If a dependency is the target of another rule, this rule is then executed.
- If the target does not exist or is older than one dependency, the command of the rule is executed.

1.3 Specific Rules

1.3.1 The rule `all`

It should appear first. Its dependencies are all the executables to be produced.

1.3.2 The rule `clean`

It suppresses the intermediary files.intermédiaires.

1.3.3 The rule `mrproper`

It suppresses everything generated by the makefile.

1.3.4 The rule `.PHONY`

With a line such as:

```
.PHONY: <target1> ... <targetN>
```

the targets `<target1>` to `<targetN>` are systematically rebuilt, even if they are more recent than their dependencies.

1.3.5 The rule `.PRECIOUS`

With a line such as:

```
.PRECIOUS: %.o
```

the makefile does not suppress the `.o` files, which should still be suppressed as intermediary files.

1.4 Silent Commands

To make a command silent, it needs to be preceded by `@`. Only error messages are printed in this case.

---

2 VARIABLES

2.1 Variable Definitions

One can, in the first part of a makefile, define several variables, such as: `<var> = <value>`, `<value>` can eventually be empty. Most of the time, those are compilation options or file lists, such as:

```
FC=h5fc
CFLAGS=-O3 - g
LDFLAGS=
MODS=<mod1>.f90 \
     ...
     <modN.f90
```

One can also import variables from another file: `include <path>/<make_common.mk>`.

2.2 Internal Variables

$@ → the target;
$< → the first dependency;
$^ → the list of dependencies;
$? → the list of dependencies more recent than the target;
$* → the file without extension.

2.3 Inference Rules

Example of building several `.o` files from their `.f90`:

```
%.o: %.f90
       <commands>
```

2.4 Generating a List of Files

A declaration such as:

```
OBJ=$(SRC:.f90=.o)
```

attributes to the variable `$(OBJ)` the list of `$(SRC)` files, but replacing the extension `.f90` by `.o`.

2.5 Using Metacharacters

One can use UNIX metacharacters with the command `wildcard`, such as:

```
SRC=$(wildcard *.f90)
```

2.6 Variables `shell`

One can use shell variables with the `shell` command, such as:

```
HOST=($shell HOSTNAME)
```

---

3 ADVANCED FUNCTIONALITIES

3.1 Decision Scheme

One can implement a classic decision scheme such as:

```
ifeq ($(VAR),value1)
       <command1>
else ifeq ($(VAR),value2)
       <command2>
else
      <command3>
endif
```

3.2 Search Directories

One can specify in which directories searching for targets and dependencies, by declaring the variable `VPATH = <dir1>:...:<dirN>`.

One can also specify more precisely the search with the directive `vpath`, such as:

```
vpath %.f90 ../Programs:../Tests
vpath %.o ../Programs
vpath lib%.a ../Programs
```
