C Makefile cheatsheet

Automatic variables

automatic variables	descriptions
$@	The file name of the target
$<	The name of the first prerequisite
$^	The names of all the prerequisites
$+	prerequisites listed more than once are duplicated in the order

Makefile

```
.PHONY: all

all: hello world

hello world: foo foo foo bar bar
	@echo "== target: $@ =="
	@echo $<
	@echo $^
	@echo $+

foo:
	@echo "Hello foo"

bar:
	@echo "Hello Bar"
```

output

```
Hello foo
Hello Bar
== target: hello ==
foo
foo bar
foo foo foo bar bar
== target: world ==
foo
foo bar
foo foo foo bar bar
```

using `$(warning text)` check make rules (for debug)

```
$(warning Top level warning)

FOO := $(warning FOO variable)foo
BAR  = $(warning BAR variable)bar

$(warning target)target: $(warning prerequisite list)Makefile $(BAR)
	$(warning tagrget script)
	@ls
$(BAR):
```

output

```
Makefile:1: Top level warning
Makefile:3: FOO variable
Makefile:6: target
Makefile:6: prerequisite list
Makefile:6: BAR variable
Makefile:9: BAR variable
Makefile:7: tagrget script
Makefile
```

string functions

Makefile

```
SRC      = hello_foo.c hello_bar.c foo_world.c bar_world.c

SUBST    = $(subst .c,,$(SRC))

SRCST    = $(SRC:.c=.o)
PATSRCST = $(SRC:%.c=%.o)
PATSUBST = $(patsubst %.c, %.o, $(SRC))

.PHONY: all

all: sub filter findstring words word wordlist

sub:
        @echo "== sub example =="
        @echo "SUBST: " $(SUBST)
        @echo "SRCST: " $(SRCST)
        @echo "PATSRCST: " $(PATSRCST)
        @echo "PATSUBST: " $(PATSUBST)
        @echo ""

filter:
        @echo "== filter example =="
        @echo "filter: " $(filter hello_%, $(SRC))
        @echo "filter-out: $(filter-out hello_%, $(SRC))"
        @echo ""

findstring:
        @echo "== findstring example =="
        @echo "Res: " $(findstring hello, hello world)
        @echo "Res: " $(findstring hello, ker)
        @echo "Res: " $(findstring world, worl)
        @echo ""

words:
        @echo "== words example =="
        @echo "num of words: "$(words $(SRC))
        @echo ""


word:
        @echo "== word example =="
        @echo "1st word: " $(word 1,$(SRC))
        @echo "2nd word: " $(word 2,$(SRC))
        @echo "3th word: " $(word 3,$(SRC))
        @echo ""


wordlist:
        @echo "== wordlist example =="
        @echo "[1:3]:"$(wordlist 1,3,$(SRC))
        @echo ""
```

output

```
$ make
== sub example ==
SUBST:  hello_foo hello_bar foo_world bar_world
SRCST:  hello_foo.o hello_bar.o foo_world.o bar_world.o
PATSRCST:  hello_foo.o hello_bar.o foo_world.o bar_world.o
PATSUBST:  hello_foo.o hello_bar.o foo_world.o bar_world.o

== filter example ==
filter:  hello_foo.c hello_bar.c
filter-out: foo_world.c bar_world.c
```

```
== findstring example ==
Res:  hello
Res:
Res:

== words example ==
num of words: 4

== word example ==
1st word:  hello_foo.c
2nd word:  hello_bar.c
3th word:  foo_world.c

== wordlist example ==
[1:3]:hello_foo.c hello_bar.c foo_world.c
```

using `$(sort list)` sort list and remove duplicates

Makefile

```
SRC = foo.c bar.c ker.c foo.h bar.h ker.h

.PHONY: all

all:
        @echo $(suffix $(SRC))
        @echo $(sort $(suffix $(SRC)))
```

output

```
$ make
.c .c .c .h .h .h
.c .h
```

single dollar sign and double dollar sign

dollar sign	descriptions
$	reference a make variable using $
$$	reference a shell variable using $$

Makefile

```
LIST = one two three

.PHONY: all single_dollar double_dollar

all: single_dollar double_dollar

double_dollar:
        @echo "=== double dollar sign example ==="
        @for i in $(LIST); do \
                echo $$i;     \
        done

single_dollar:
        @echo "=== single dollar sign example ==="
        @for i in $(LIST); do  \
                echo $i;     \
        done
```

output

```
$ make
=== single dollar sign example ===
```

```
=== double dollar sign example ===
one
two
three
```

build executable files respectively

directory layout

```
.
|-- Makefile
|-- bar.c
|-- bar.h
|-- foo.c
`-- foo.h
```

Makefile

```
# CFLAGS: Extra flags to give to the C compiler
CFLAGS   += -Werror -Wall -O2 -g
SRC       = $(wildcard *.c)
OBJ       = $(SRC:.c=.o)
EXE       = $(subst .c,,$(SRC))

.PHONY: all clean

all: $(OBJ) $(EXE)

clean:
    rm -rf *.o *.so *.a *.la $(EXE)
```

output

```
$ make
cc -Werror -Wall -O2 -g   -c -o foo.o foo.c
cc -Werror -Wall -O2 -g   -c -o bar.o bar.c
cc   foo.o   -o foo
cc   bar.o   -o bar
```

using `$(eval)` predefine variables

without `$(eval)`

```
SRC = $(wildcard *.c)
EXE = $(subst .c,,$(SRC))

define PROGRAM_template
$1_SHARED = lib$(strip $1).so
endef

.PHONY: all

$(foreach exe, $(EXE), $(call PROGRAM_template, $(exe)))

all:
        @echo $(foo_SHARED)
        @echo $(bar_SHARED)
```

output

```
$ make
Makefile:11: *** missing separator.  Stop.
```

with `$(eval)`

```
CFLAGS  += -Wall -g -O2 -I./include
SRC = $(wildcard *.c)
EXE = $(subst .c,,$(SRC))

define PROGRAM_template
$1_SHARED = lib$(strip $1).so
endef

.PHONY: all

$(foreach exe, $(EXE), $(eval $(call PROGRAM_template, $(exe))))

all:
        @echo $(foo_SHARED)
        @echo $(bar_SHARED)
```

output

```
$ make
libfoo.so
libbar.so
```

build subdir and link together

directory layout

```
.
|-- Makefile
|-- include
|   `-- foo.h
`-- src
    |-- foo.c
    `-- main.c
```

Makefile

```
CFLAGS  += -Wall -g -O2 -I./include
SRC     = $(wildcard src/*.c)
OBJ     = $(SRC:.c=.o)
EXE     = main

.PHONY: all clean

all: $(OBJ) $(EXE)

$(EXE): $(OBJ)
        $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

clean:
        rm -rf *.o *.so *.a *.la $(EXE) src/*.o src/*.so src/*a
```

output

```
$ make
cc -Wall -g -O2 -I./include -c src/foo.c -o src/foo.o
cc -Wall -g -O2 -I./include -c src/main.c -o src/main.o
cc  -o main src/foo.o src/main.o
```

build shared library

directory layout

```
.
|-- Makefile
|-- include
|   `-- common.h
`-- src
    |-- bar.c
    `-- foo.c
```

Makefile

```
SONAME    = libfoobar.so.1
SHARED    = src/libfoobar.so.1.0.0
SRC       = $(wildcard src/*.c)
OBJ       = $(SRC:.c=.o)

CFLAGS    += -Wall -Werror -fPIC -O2 -g -I./include
LDFLAGS   += -shared -Wl,-soname,$(SONAME)

.PHONY: all clean

all: $(SHARED) $(OBJ)

$(SHARED): $(OBJ)
        $(CC) $(LDFLAGS) -o $@ $^

%.o: %.c
        $(CC) $(CFLAGS) -c $^ -o $@

clean:
        rm -rf src/*.o src/*.so.* src/*.a src/*.la
```

output

```
$ make
cc -Wall -Werror -fPIC -O2 -g -I./include -c src/foo.c -o src/foo.o
cc -Wall -Werror -fPIC -O2 -g -I./include -c src/bar.c -o src/bar.o
cc -shared -Wl,-soname,libfoobar.so.1 -o src/libfoobar.so.1.0.0 src/foo.o src/bar.o
```

build shared and static library

directory layout

```
.
|-- Makefile
|-- include
|   |-- bar.h
|   `-- foo.h
`-- src
    |-- Makefile
    |-- bar.c
    `-- foo.c
```

Makefile

```
SUBDIR = src

.PHONY: all clean $(SUBDIR)

all: $(SUBDIR)

clean: $(SUBDIR)

$(SUBDIR):
        make -C $@ $(MAKECMDGOALS)
```

src/Makefile

```
SRC      = $(wildcard *.c)
OBJ      = $(SRC:.c=.o)
LIB      = libfoobar

STATIC   = $(LIB).a
SHARED   = $(LIB).so.1.0.0
SONAME   = $(LIB).so.1
SOFILE   = $(LIB).so

CFLAGS  += -Wall -Werror -g -O2 -fPIC -I../include
LDFLAGS += -shared -Wl,-soname,$(SONAME)

.PHONY: all clean

all: $(STATIC) $(SHARED) $(SONAME) $(SOFILE)

$(SOFILE): $(SHARED)
        ln -sf $(SHARED) $(SOFILE)

$(SONAME): $(SHARED)
        ln -sf $(SHARED) $(SONAME)

$(SHARED): $(STATIC)
        $(CC) $(LDFLAGS) -o $@ $<

$(STATIC): $(OBJ)
        $(AR) $(ARFLAGS) $@ $^

%.o: %.c
        $(CC) $(CFLAGS) -c -o $@ $<

clean:
        rm -rf *.o *.a *.so *.so.*
```

output

```
$ make
make -C src
make[1]: Entering directory '/root/test/src'
cc -Wall -Werror -g -O2 -fPIC -I../include -c -o foo.o foo.c
cc -Wall -Werror -g -O2 -fPIC -I../include -c -o bar.o bar.c
ar rv libfoobar.a foo.o bar.o
ar: creating libfoobar.a
a - foo.o
a - bar.o
cc -shared -Wl,-soname,libfoobar.so.1 -o libfoobar.so.1.0.0 libfoobar.a
ln -sf libfoobar.so.1.0.0 libfoobar.so.1
ln -sf libfoobar.so.1.0.0 libfoobar.so
make[1]: Leaving directory '/root/test/src'
```

build recursively

directory layout

```
.
|-- Makefile
|-- include
|   `-- common.h
|-- src
|   |-- Makefile
|   |-- bar.c
|   `-- foo.c
`-- test
    |-- Makefile
    `-- test.c
```

v: latest

Makefile

```
SUBDIR = src test

.PHONY: all clean $(SUBDIR)

all: $(SUBDIR)

clean: $(SUBDIR)

$(SUBDIR):
	$(MAKE) -C $@ $(MAKECMDGOALS)
```

src/Makefile

```
SONAME   = libfoobar.so.1
SHARED   = libfoobar.so.1.0.0
SOFILE   = libfoobar.so

CFLAGS  += -Wall -g -O2 -Werror -fPIC -I../include
LDFLAGS += -shared -Wl,-soname,$(SONAME)

SRC      = $(wildcard *.c)
OBJ      = $(SRC:.c=.o)

.PHONY: all clean

all: $(SHARED) $(OBJ)

$(SHARED): $(OBJ)
	$(CC) $(LDFLAGS) -o $@ $^
	ln -sf $(SHARED) $(SONAME)
	ln -sf $(SHARED) $(SOFILE)

%.o: %.c
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -rf *.o *.so.* *.a *.so
```

test/Makefile

```
CFLAGS    += -Wall -Werror -g -I../include
LDFLAGS   += -Wall -L../src -lfoobar

SRC        = $(wildcard *.c)
OBJ        = $(SRC:.c=.o)
EXE        = test_main

.PHONY: all clean

all: $(OBJ) $(EXE)

$(EXE): $(OBJ)
	$(CC) -o $@ $^ $(LDFLAGS)

%.o: %.c
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -rf *.so *.o *.a $(EXE)
```

output

```
$ make
make -C src
make[1]: Entering directory '/root/proj/src'
cc -Wall -g -O2 -Werror -fPIC -I../include -c foo.c -o foo.o
cc -Wall -g -O2 -Werror -fPIC -I../include -c bar.c -o bar.o
```

```
cc -shared -Wl,-soname,libfoobar.so.1 -o libfoobar.so.1.0.0 foo.o bar.o
ln -sf libfoobar.so.1.0.0 libfoobar.so.1
ln -sf libfoobar.so.1.0.0 libfoobar.so
make[1]: Leaving directory '/root/proj/src'
make -C test
make[1]: Entering directory '/root/proj/test'
cc -Wall -Werror -g -I../include -c test.c -o test.o
cc -o test_main test.o -Wall -L../src -lfoobar
make[1]: Leaving directory '/root/proj/test'
$ tree .
.
|-- Makefile
|-- include
|   `-- common.h
|-- src
|   |-- Makefile
|   |-- bar.c
|   |-- bar.o
|   |-- foo.c
|   |-- foo.o
|   |-- libfoobar.so -> libfoobar.so.1.0.0
|   |-- libfoobar.so.1 -> libfoobar.so.1.0.0
|   `-- libfoobar.so.1.0.0
`-- test
    |-- Makefile
    |-- test.c
    |-- test.o
    `-- test_main

3 directories, 14 files
```

replace current shell

```
OLD_SHELL := $(SHELL)
SHELL = /usr/bin/python

.PHONY: all

all:
        @import os; print os.uname()[0]
```

output

```
$ make
Linux
```

one line condition

syntax: `$(if cond, then part, else part)`

Makefile

```
VAR =
IS_EMPTY = $(if $(VAR), $(info not empty), $(info empty))

.PHONY: all

all:
        @echo $(IS_EMPTY)
```

output

```
$ make
empty
```

```
$ make VAR=true
not empty
```

Using define to control CFLAGS

Makefile

```
CFLAGS += -Wall -Werror -g -O2
SRC     = $(wildcard *.c)
OBJ     = $(SRC:.c=.o)
EXE     = $(subst .c,,$(SRC))

ifdef DEBUG
CFLAGS += -DDEBUG
endif

.PHONY: all clean

all: $(OBJ) $(EXE)

clean:
        rm -rf $(OBJ) $(EXE)
```

output

```
$ make
cc -Wall -Werror -g -O2   -c -o foo.o foo.c
cc   foo.o   -o foo
$ make DEBUG=1
cc -Wall -Werror -g -O2 -DDEBUG   -c -o foo.o foo.c
cc   foo.o   -o foo
```