誰にでも使える make 講座
第 3 回
「まとめるマクロ」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : Makefile のマクロ。
root : ああ、そうだったね。この前の Makefile を見せてくれるかい？
yasuoka : はい。あ、ちょっとだけ変更したんですけど。

```
/home/yasuoka/src/kbanner% cat Makefile （ぽこ）
SHELL = /bin/sh

.SUFFIXES: .c .o

.c.o:
        cc -c $<

kbanner: kbanner.o font1.o font2.o font3.o
        cc kbanner.o font1.o font2.o font3.o -o kbanner
        strip kbanner

clean clear:
        @echo -n 'Really make $@? '
        @read YN ; case $$YN in y*) : ;; *) false ;; esac
        rm -f *.o core

install: kbanner
        cp kbanner $$HOME/bin/kbanner

font1.o font2.o font3.o: font.h
/home/yasuoka/src/kbanner%
```

root : よし、まずは 8 行目と 9 行目と最終行の font1.o font2.o font3.o を、ひとまとめにしよう。

```
SHELL = /bin/sh
FONTOBJ = font1.o font2.o font3.o

.SUFFIXES: .c .o

.c.o:
        cc -c $<

kbanner: kbanner.o $(FONTOBJ)
        cc kbanner.o $(FONTOBJ) -o kbanner
        strip kbanner

clean clear:
        @echo -n 'Really make $@? '
        @read YN ; case $$YN in y*) : ;; *) false ;; esac
        rm -f *.o core

install: kbanner
        cp kbanner $$HOME/bin/kbanner

$(FONTOBJ): font.h
```

（間）

yasuoka : できました。
root : じゃ、 make を実行してみてごらん。
yasuoka : はい。

```
/home/yasuoka/src/kbanner% make （ぽこ）
cc -c kbanner.c
cc -c font1.c
cc -c font2.c
cc -c font3.c
cc kbanner.o font1.o font2.o font3.o -o kbanner
```

```
strip kbanner
/home/yasuoka/src/kbanner% ■
```

前とおんなじですね。

root : うん。

マクロ = 文字列
　　マクロへの代入。文字列は複数書いてもよい。
$(マクロ)
　　マクロの読み出し。

yasuoka : 2 行目の FONTOBJ = font1.o font2.o font3.oって、変数の代入みたいですけど。

root : それが make のマクロなんだよ。$(FONTOBJ)ってのが、読み出しにあたるんだ。これを使えば、同じ内容のところをひとまとめに書けるんで、便利だね。

yasuoka : ええ。

root : でもシェルの変数なんかと違うところは、一度しか代入がきかないところだ。

yasuoka : え？ 途中で変更できないんですか？

root : そう。まあそういう意味では、変数と言うよりは定数に近いね。

yasuoka : ふーん。

root : これだけじゃ何だから、マクロを使って、もうちょっときれいな Makefile にしてみようか。

```
SHELL = /bin/sh
PROGRAM = kbanner
DESTDIR = $$HOME/bin
FONTOBJ = font1.o font2.o font3.o
OBJ = kbanner.o $(FONTOBJ)

.SUFFIXES: .c .o

.c.o:
        cc -c $<

$(PROGRAM): $(OBJ)
        cc $(OBJ) -o $(PROGRAM)
        strip $(PROGRAM)

clean clear:
        @echo -n 'Really make $@? '
        @read YN ; \
         case $$YN in \
         y*) : ;; \
         *) false ;; \
         esac
        rm -f *.o core

install: $(PROGRAM)
        cp $(PROGRAM) $(DESTDIR)/$(PROGRAM)

$(FONTOBJ): font.h
```

root : 行末に \ を入れておくと、複数行が 1 行だとみなされるのは、知ってて損はないよ。

yasuoka : そうなんですか。

```
/home/yasuoka/src/kbanner% make （ぽこ）
'kbanner' is up to date.
/home/yasuoka/src/kbanner% make clean （ぽこ）
Really make clean? yes （ぽこ）
rm -f *.o core
/home/yasuoka/src/kbanner% ls （ぽこ）
Makefile        font.h          font2.c         kbanner
README          font1.c         font3.c         kbanner.c
/home/yasuoka/src/kbanner% cd ../lfh （ぽこ）
/home/yasuoka/src/lfh% ls （ぽこ）
main.c  sub.c
/home/yasuoka/src/lfh% ■
```

ところで root さん。

root : 何だい？

yasuoka ： このlfhっていうプログラムにも、Makefileを作りたいな、って思ってるんですけど。

root ： main.cとsub.cを、単純にコンパイルするだけかい？

yasuoka ： ええ。ただ、この先.cで終わるファイルが増えるかもしれないんです。で、その時にもMakefileを書き換えずにすませられたらなぁ、と思うんですけど。

root ： うーん。

```
SHELL = /bin/sh
PROGRAM = lfh
DESTDIR = $$HOME/bin

.SUFFIXES: .c .o

.c.o:
        cc -c $<

all:
        @make OBJ="`echo *.c | sed 's/\.c/.o/g'`" $(PROGRAM)

$(PROGRAM): $(OBJ)
        cc $(OBJ) -o $(PROGRAM)
        strip $(PROGRAM)

clean:
        rm -f *.o core

install: all
        cp $(PROGRAM) $(DESTDIR)/$(PROGRAM)
```

（間）

root ： まずは最低限、こんなところかなぁ。

yasuoka ： OBJへの代入がないように思えるんですけど。

root ： うん。まずはmake OBJ='main.o sub.o' lfhを実行してごらん。

yasuoka ： はい。

```
/home/yasuoka/src/lfh% make OBJ='main.o sub.o' lfh（ぽこ）
cc -c main.c
cc -c sub.c
cc main.o sub.o -o lfh
strip lfh
/home/yasuoka/src/lfh% ls（ぽこ）
Makefile        main.c          sub.c
lfh             main.o          sub.o
/home/yasuoka/src/lfh%
```

あ、うまくいった。make OBJ='main.o sub.o' lfhってのは、どういう意味なんですか？

root ： OBJっていうマクロにmain.o sub.oを代入しながら、lfhを作るって意味だよ。マクロへの代入はMakefileの中だけじゃなくて、makeを実行する時にもできるんだ。

yasuoka ： へーえ。

root ： 今度はmake cleanしてから、もう一度、単にmakeしてごらん。

yasuoka ： はい。

```
/home/yasuoka/src/lfh% make clean（ぽこ）
rm -f *.o core
/home/yasuoka/src/lfh% make（ぽこ）
cc -c main.c
cc -c sub.c
cc main.o sub.o -o lfh
strip lfh
/home/yasuoka/src/lfh%
```

root ： うまくいったろう？

yasuoka ： ええ、どうなってるんですか？

root ： 単にmakeした時は、いちばん最初のallが作られるだろ。すると 11 行目のmake OBJ="`echo *.c | sed 's/\.c/.o/g'`" $(PROGRAM)が実行される。

yasuoka ： はい。

root ： ここでecho *.c | sed 's/\.c/.o/g'の結果は

```
/home/yasuoka/src/lfh% echo *.c | sed 's/\.c/.o/g'（ぽこ）
main.o sub.o
```

```
/home/yasuoka/src/lfh% ■
```

だし、 $(PROGRAM) は lfh だから、結局 make OBJ="main.o sub.o" lfh が実行されたのと、同じになる。

yasuoka : すいません、その sed 's/\.c/.o/g'って何ですか？

> sed -e 文字列 ファイル名
> 文字列を sed の命令とみなして、 sed を実行する。「-e 文字列」は複数書いてもよい。「-e 文字列」がただ一つしかない場合は、 -e を省略してもよい。入力はファイル、出力は標準出力。ただしファイル名が省略された場合は、入力は標準入力。

root : 入力中の.cって文字列を、全部.o に置き換えてるんだ。詳しくは「誰にでも書ける #! /bin/sed -f 講座」を読むように。

yasuoka : わかりました。

root : これで、カレントディレクトリの.c ファイルが増えても、ちゃんと必要な.o ファイルが OBJ に代入されるわけだ。

yasuoka : ええっと

```
/home/yasuoka/src/lfh% echo *.c （ぽこ）
main.c sub.c
/home/yasuoka/src/lfh% !! | sed 's/\.c/.o/g' （ぽこ）
echo *.c | sed 's/\.c/.o/g'
main.o sub.o
/home/yasuoka/src/lfh% ■
```

ああ、そうか。なかなかうまくできてますね。

root : まあね。あと make veryclean を付け加えたら完璧かな。

```
SHELL = /bin/sh
PROGRAM = lfh
DESTDIR = $$HOME/bin

.SUFFIXES: .c .o

.c.o:
        cc -c $<

.DEFAULT:
        @case $@ in \
         *clean) : ;; \
         *) echo "Make:  Don't know how to make $@." ; \
            false ;; \
         esac
        rm -f *.o $(PROGRAM) core

all:
        @make OBJ="`echo *.c | sed 's/\.c/.o/g'`" $(PROGRAM)

$(PROGRAM): $(OBJ)
        cc $(OBJ) -o $(PROGRAM)
        strip $(PROGRAM)

clean:
        rm -f *.o core

install: all
        cp $(PROGRAM) $(DESTDIR)/$(PROGRAM)
```

（間）

yasuoka : どういうことですか？

root : まあ、ためしに make veryclean を実行してみてごらん。

yasuoka : はい。

```
/home/yasuoka/src/lfh% make veryclean （ぽこ）
rm -f *.o lfh core
/home/yasuoka/src/lfh% ■
```

あれ、 make clean が実行された。

root : いや、よく見てごらん。 make clean とは違うだろ？

yasuoka : え？

```
/home/yasuoka/src/lfh% make clean （ぽこ）
rm -f *.o core
```

```
/home/yasuoka/src/lfh% ■
```

あ、本当だ。lfh が rm されてる分が違う。どこを実行してるんですか？

root ： .DEFAULT だよ。

yasuoka ： .DEFAULTって？

root ： make は Makefile の中に作り方が書いてないものには、普通エラーを出すけど、もし.DEFAULTっていうターゲットがあったなら、代わりにそれを作るんだ。

yasuoka ： ふーん。

root ： で、この Makefile では $@ で末尾が clean のものだけをより分けて、それに対してだけ rm -f *.o lfh core を実行してる。

yasuoka ： それが 11 行目から 16 行目ですね。でもどうして、末尾が clean のものだけなんですか？

root ： 本当は verycleanっていう、実行ファイルも全部消してしまうターゲットを書きたいんだけど、人によっては realclean だとか、 moreclean じゃなきゃいやだとか、色々な流儀があるんだよ。そこでとにかく clean で終わるものなら、きれいにしようと。

yasuoka ： 普通にきれいにする時は、 clean でいいんですか？

root ： うん、それは大体共通の認識になってる。でも「もっときれいにしたい」時のターゲットには、各流派あるみたいだから。ま、最近ではこう書くのが普通みたい。

yasuoka ： ふーん。

```
/home/yasuoka/src/lfh% make realclean （ぽこ）
rm -f *.o lfh core
/home/yasuoka/src/lfh% make moreclean （ぽこ）
rm -f *.o lfh core
/home/yasuoka/src/lfh% make OBJ=obj.o lfh （ぽこ）
Make:  Don't know how to make obj.o.
*** Exit 1

Stop.
/home/yasuoka/src/lfh% ls （ぽこ）
Makefile        main.c          sub.c
/home/yasuoka/src/lfh% ■
```

make の時に代入してるマクロに、 Makefile の中でも代入があったら、どうなります？

root ： 今の lfh だと、例えば Makefile の中に OBJ = main.o sub.o とか書いてあった場合だね？

yasuoka ： はい。

root ： make 実行時に代入した方が勝つ。

yasuoka ： そうなんですか。じゃ、複数のサフィクスルールは？

root ： 複数のサフィクスルールって言うと？

yasuoka ： 例えば.c から.o を作るルールと、.f から.o を作るルールの両方がある時には、どちらから.o が作られるんですか？

root ： ああ、そういう場合か。.SUFFIXES に書かれてる順番が、前のものほど優先される。例えば obj.o を作る時に

```
.SUFFIXES: .c .f .o
```

となっていたら、まず obj.c があるかどうか確かめて、あれば.c.o のルールを使う。なければ obj.f から.f.o のルールを使う。逆に

```
.SUFFIXES: .f .c .o
```

なら obj.f、 obj.c の順番に調べる。

yasuoka ： だいたいわかりました。それから、 Makefile にはコメントは書けないんですか？

root ： そういえば、一度も書かなかったね。シェル・スクリプトなんかと同じで、 # から始まる行がコメントになるよ。

yasuoka ： そうですか。

root ： make については、このぐらいのことを知っておけば、まあ困ることはないだろうな。例として C のプログラムを取り上げてきたけど、別にどんなコマンドでも実行できるから、同じ作業を何回も何回も繰り返す時なんかには、軽く Makefile でも書いて、あとは毎回 make 一発ってのがとってもおしゃれだね。

yasuoka ： そうですね。これからはそうするようにします。

root ： よし、じゃこれで make については終わりにしよう。

yasuoka ： はい。どうもありがとうございました。