誰にでも使える make 講座
第 2 回
「@ と - と @-」

安岡孝一

yasuoka : root さん、root さん。
root : 何だい？
yasuoka : make clean がうまく動かないんです。
root : え？
yasuoka : こんな感じなんですけど。

```
/home/yasuoka/src/kbanner% make clean （ぽこ）
echo -n 'Really make clean? '
Really make clean? read YN
Make: Cannot load read.  Stop.
/home/yasuoka/src/kbanner% 
```

root : どういうこと？
yasuoka : えっとですね。

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

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

clean:
        echo -n 'Really make clean? '
        read YN
        if test $$YN != y
        then exit 1
        fi
        rm -f *.o core

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

kbanner.o: kbanner.c
        cc -c kbanner.c

font1.o: font1.c font.h
        cc -c font1.c

font2.o: font2.c font.h
        cc -c font2.c

font3.o: font3.c font.h
        cc -c font3.c
/home/yasuoka/src/kbanner% 
```

make clean を実行した時に、いきなりファイルを消しちゃうのは何か恐いから、Really make clean? と訊いて y を答えたら消そうと。
root : じゃあ、rm -f *.o core の -f のところを、-fi にしたらいいんじゃないの？
yasuoka : それだと、1 ファイルごとに訊いてくるでしょう？ そうじゃなくって、1 回だけ訊きたいんです。
root : そうか。
yasuoka : でもうまくいきません。
root : make ではコマンドは、1 行 1 行別々のシェルにかけられるからね。同じシェル変数を参照してる部分とか、if と then と fi とかは、1 行に書かなきゃいけない。
yasuoka : そうなんですか。
root : それから、この echo -n 'Really make clean? ' とかは、最初に @ を付けた方がいいな。
yasuoka : @って？
root : 「コマンドの実行を表示しない」って意味だよ。echo -n 'Really make clean? ' と Really make clean? の両方が表示されるのは、悲しいだろ？
yasuoka : そうですね。だとすると…。

（間）

yasuoka ： こんなもんでしょうか？

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

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

clean:
        @echo -n 'Really make clean? '
        @read YN ; if test $$YN != y ; then exit 1 ; fi
        rm -f *.o core

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

kbanner.o: kbanner.c
        cc -c kbanner.c

font1.o: font1.c font.h
        cc -c font1.c

font2.o: font2.c font.h
        cc -c font2.c

font3.o: font3.c font.h
        cc -c font3.c
/home/yasuoka/src/kbanner% 
```

root ： ま、ためしに make clean してごらん。

yasuoka ： はい。

```
/home/yasuoka/src/kbanner% make clean （ぽこ）
Really make clean? 
```

今度はちゃんと訊いてきた。echo -n 'Really make clean? ' も出てないや。

```
Really make clean? n （ぽこ）
*** Exit 1

Stop.
/home/yasuoka/src/kbanner% 
```

うまくいってるみたい。

root ： そうかな？ 今度は y を答えてごらん。

yasuoka ： え？

```
/home/yasuoka/src/kbanner% !! （ぽこ）
make clean
Really make clean? y （ぽこ）
*** Exit 1

Stop.
/home/yasuoka/src/kbanner% 
```

あれ？ rm されない。どうしてですか？

root ： だって y を答えた時には test $YN != y のエグジットステイタスは 1 になっちゃうだろ。それを make はエラーだと思って、実行を中止してしまうからね。

yasuoka ： エグジットステイタスが 0 以外でも、それを無視して、実行を続けるようにはできないんですか？

root ： @ のところに - を書けばできるけど。

yasuoka ： @ の代わりに - ですか？

root ： @ と併用するなら @-、しないなら単に - だよ。でもたぶんそれじゃ、うまくいかないだろうな。

yasuoka ： どうしてですか？

root ： ま、ものはためしだ。やってみてごらん。

（間）

yasuoka ： できました。

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

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

clean:
        @echo -n 'Really make clean? '
        @-read YN ; if test $$YN != y ; then exit 1 ; fi
        rm -f *.o core

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

kbanner.o: kbanner.c
        cc -c kbanner.c

font1.o: font1.c font.h
        cc -c font1.c

font2.o: font2.c font.h
        cc -c font2.c

font3.o: font3.c font.h
        cc -c font3.c
/home/yasuoka/src/kbanner% ■
```

これで make cleanっと。

```
/home/yasuoka/src/kbanner% make clean（ぽこ）
Really make clean? y（ぽこ）
*** Exit 1 (ignored)
rm -f *.o core
/home/yasuoka/src/kbanner% ■
```

お、ignored で無視された。rm も実行されてるし、うまくいってるみたいですよ。

root ： そうかな？ じゃ、n を答えてみてごらん。

yasuoka ： n ですか？

```
/home/yasuoka/src/kbanner% !!（ぽこ）
make clean
Really make clean? n（ぽこ）
*** Exit 1 (ignored)
rm -f *.o core
/home/yasuoka/src/kbanner% ■
```

あら、無視されて rm が実行されちゃった。

root ： だろ？ 最初に - を付けると、exit 1 だろうと何だろうと、とにかく無視して実行を続けてしまうからね。

yasuoka ： どうしたらいいんですか？

root ： そもそも、エグジットステイタスを無視しようなんて考えが、よくなかったんだよ。y を答えた時にはエグジットステイタスが 0、それ以外なら 1 になるようにすれば、よかったんだ。

```
SHELL = /bin/sh

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

clean:
        @echo -n 'Really make clean? '
        @read YN ; test "$$YN" = y
        rm -f *.o core

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

kbanner.o: kbanner.c
        cc -c kbanner.c

font1.o: font1.c font.h
        cc -c font1.c
```

```
font2.o: font2.c font.h
        cc -c font2.c

font3.o: font3.c font.h
        cc -c font3.c
```

yasuoka : ああ、そうか。これなら大丈夫ですね。
root : どれどれ、もうできたかい？
yasuoka : はい。

```
/home/yasuoka/src/kbanner% !m（ぽこ）
make clean
Really make clean? y（ぽこ）
rm -f *.o core
/home/yasuoka/src/kbanner% !!（ぽこ）
make clean
Really make clean? n（ぽこ）
*** Exit 1

Stop.
/home/yasuoka/src/kbanner%
```

お、完璧。
root : ま、こんなもんだな。
yasuoka : もう一つやりたいことがあるんですけど、いいですか？
root : 何だい？
yasuoka : clean の代わりに make clear ってしたときも、ファイルを消したいんですけど。
root : ああ、それは簡単だよ。
yasuoka : ただしその時は、 Really make clear?って訊いてほしいんです。
root : うーん。

```
SHELL = /bin/sh

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 ; test "$$YN" = y
        rm -f *.o core

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

kbanner.o: kbanner.c
        cc -c kbanner.c

font1.o: font1.c font.h
        cc -c font1.c

font2.o: font2.c font.h
        cc -c font2.c

font3.o: font3.c font.h
        cc -c font3.c
```

（間）

yasuoka : これでできるんですか？

```
/home/yasuoka/src/kbanner% make clear（ぽこ）
Really make clear? y（ぽこ）
rm -f *.o core
/home/yasuoka/src/kbanner% make clean（ぽこ）
Really make clean? y（ぽこ）
rm -f *.o core
/home/yasuoka/src/kbanner%
```

うまくいってる。どうなってるんですか？
root : 7 行目の clean のところに、 clear ってターゲットを加えただけだよ。これで make clean か make clear ってした時に、 8 行目から 10 行目が実行されるわけだ。

yasuoka : 8 行目もちょっと変わってるみたいですけど、この $@って何ですか？
root : この $@ は、実行する時にはターゲットに置き換えられる。つまり 8 行目は、ターゲットが clean だったら @echo -n 'Really make clean? ' に、clear なら @echo -n 'Really make clear? ' に解釈される。
yasuoka : 実行するコマンドが、全く同じか、あるいはよく似てる時には、ひとまとめに書けるんですね。
root : まあ、そうかな。
yasuoka : じゃあ、この font1.o から font3.o の部分も、よく似てますけど、これはひとまとめにできないんですか？
root : それは、全然違う方法を使わないと、無理だな。

```
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 ; test "$$YN" = y
        rm -f *.o core

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

font1.o font2.o font3.o: font.h
```

yasuoka : えらく短くなりましたねー。どうなったんですか？
root : 「何とか.c」から「何とか.o」を作るところを、ひとまとめにしたんだ。
yasuoka : って言いますと？
root : 5 行目と 6 行目がそれ。ここは「『何とか.c』から『何とか.o』を作るには『cc -c 何とか.c』を実行する」っていう風に読むんだ。
yasuoka : $<って何ですか？
root : そういう作り方をする時の、元になる方のファイル。

$$	$
$@	ターゲット
$?	必要なファイルのうちターゲットより新しいもの全部
$<	サフィクスルールでの元のファイル、サフィクスルールでだけ使用可
$*	ターゲットからサフィクスを除いたもの、サフィクスルールでだけ使用可

root : 例えば kbanner.o を作る時には、5 行目から 6 行目は「kbanner.o を作るには kbanner.c が必要で、作るためのコマンドは cc -c kbanner.c」っていう意味になるんだ。
yasuoka : font1.o を作る時には？
root : 「font1.o を作るには font1.c が必要で、作るためのコマンドは cc -c font1.c」っていう意味になる。
yasuoka : あれ？ 前の Makefile じゃ「font1.o を作るのに必要なのは、font1.c と font.h」っていう風になってませんでした？
root : うん。だから最終行にそう追加してある。この行は「font1.o か font2.o か font3.o を作るには font.h も必要」って意味だからね。そしてここには作り方が書いてないから、実際に作る時には 6 行目の cc -c $<が実行される。
yasuoka : 最後の行の後ろに、作り方を書いた時は？
root : そっちが優先される。
yasuoka : うーん、すごい。
root : ただし、このサフィクスルールって機能を使うには、3 行目みたいに、使うサフィクスの一覧を.SUFFIXES で書いとかなきゃいけない。この行は「『何とか.c』や『何とか.o』については、サフィクスルールを見て下さい」って意味だよ。さて、もうこんな時間だ。あとマクロの話をしてないけど、今日はこのへんで終わりにしようか。
yasuoka : マクロ？
root : Makefile の中に同じことが何回も書いてある時に、それらをひとまとめにする方法だよ。
yasuoka : そんな方法があるんですか。次回は必ず教えて下さいね。