makefile 課題

学籍番号 045718D : 翁長絵美

平成 17 年 5 月 10 日

課題:全ての Level にトライし、レポートとしてまとめよ。

1 Level1

サンプルプログラムをダウンロードし、make を実行せよ。make 実行時、端末上にその動作が表示された内容を記録し、その内容が何を意味するのか説明せよ。

1.1 実行結果

```
[Emi-ONAGA:~/Desktop/make-sample] j04018% ls
0readme.txt add.c makefile2 multi.c
Makefile makefile1 makefile3 sample.c
[Emi-ONAGA:~/Desktop/make-sample] j04018% make
cc -c -o sample.o sample.c
cc -c -o add.o add.c
cc -c -o multi.o multi.c
cc sample.o add.o multi.o -o sample
[Emi-ONAGA:~/Desktop/make-sample] j04018% ls
0readme.txt add.o makefile3 sample
Makefile makefile1 multi.c sample.c
add.c makefile2 multi.o sample.o
```

1.2 考察

サンプルプログラムをダウンロードした直後に ls を実行してみると、8 つのファイルが make-sample の中に入ってることがわかる。make を実行してみると、実行結果から sample.c、add.c、multi.c というファイルを sample.o、add.o、multi.o というオブジェクトファイルにコンパイルして生成している。さらに sample.o、add.o、multi.o をまとめて sample という 1 つのファ

イルに生成していることがわかる。make を実行した後、ls を実行してみると、ちゃんと sample.o、add.o、multi.o、sample　が出来ていることが確かめられる。

2　Level2

どれか１つ以上のソースファイルを編集・保存し、make を実行せよ (コンパイルが通るのであれば編集内容は問わない)。編集対象毎に make の動作は異なるはずである。どのファイルを編集するとどう動作するのかを確認せよ。

2.1　実行結果

2.1.1　sample.c の場合

```
[Emi-ONAGA:~/Desktop/make-sample] j04018% make
cc -c -o sample.o sample.c
cc sample.o add.o multi.o -o sample
```

2.1.2　add.c の場合

```
[Emi-ONAGA:~/Desktop/make-sample] j04018% make
cc -c -o add.o add.c
cc sample.o add.o multi.o -o sample
```

2.1.3　multi.c の場合

```
[Emi-ONAGA:~/Desktop/make-sample] j04018% make
cc -c -o multi.o multi.c
cc sample.o add.o multi.o -o sample
```

2.2　考察

各ソースファイルを編集し、make を実行すると、編集したファイルだけ再びコンパイルされていることが実行結果から見てわかる。そして、それぞれまた sample という１つのファイルにしている。各ソースファイルを編集してわかったことは、sample.c は変数の値や出力の仕方を定義し、add.c は変数の加算の仕方、multi.c は変数の乗算の仕方をそれぞれ定義している。

3 Level3

touch コマンドにより１つ以上のソースファイルのタイムスタンプを更新し、make を実行せよ。この時の make の Level2 と比較して同じはずである。何故このような動作になっているのか考察せよ。

3.0.1 実行結果

```
[Emi-ONAGA:~/Desktop/make-sample] j04018% ls -l sample.c
-rw-r--r-- 1 j04018 j04018 232 10 May 01:09 sample.c
[Emi-ONAGA:~/Desktop/make-sample] j04018% touch sample.c
[Emi-ONAGA:~/Desktop/make-sample] j04018% ls -l sample.c
-rw-r--r-- 1 j04018 j04018 232 10 May 02:50 sample.c
[Emi-ONAGA:~/Desktop/make-sample] j04018% make
cc -c -o sample.o sample.c
cc sample.o add.o multi.o -o sample
```

3.0.2 考察

まず最初に ls -l sample.c を実行してみると、最終に更新したのは 10 May 01:09 となっている。そこで、touch sample.c を実行した後にもう一度 ls -l sample.c をしてみると 10 May 02:50 と、何も手を加えずに時間だけを最新にすることができた。この状態で make を実行すると、touch をした sample.c だけが再コンパイルされていることがわかる。これは touch コマンドをすることにより make をした時間よりも sample.c の方が新しい時間になるため、最新にコンパイルされてないと認識され、内容が変わっていなくても sample.c だけがまたコンパイルされると考えられる。

4 Level4

サンプルには幾つかの Makefile 例があり、makefile2 は詳細動作まで記述した例である。makefile2 の source や command を削除し、make を実行せよ。この例では全く同一の動作になっているはずである。これは make に推論機能が実装されているためである。各自削除した source/command について、具体的に何が推論により補われているのかを示せ。

→ソースとして「hoge.o」を指定した場合、「hoge.c」が
あれば make はそのソースが cc によりコンパイルします。
これは make がそのような関係をしっているからですが、
知らないものに対してはどのように教えればよいのか？

4.1 実行結果

4.1.1 makefile2 の編集

```
[Emi-ONAGA:~/Desktop/make-sample] j04018% emacs -nw makefile2
CC = gcc
CFLAGS = -Wall -O2

sample:  sample.o add.o multi.o
sample.o:
add.o:
multi.o:
clean:
        rm -f *.o sample
```

4.1.2 make の実行

```
[nw0418:~/Desktop/make-sample] j04018% make -f makefile2
gcc -Wall -O2 -c -o sample.o sample.c
gcc -Wall -O2 -c -o add.o add.c
gcc -Wall -O2 -c -o multi.o multi.c
gcc sample.o add.o multi.o -o sample
```

4.2 考察

makefile2 の最初のターゲットである sample の source は指定するために必要なので消せないが、それ以下のターケッドはそれぞれのオブジェクトファイルを生成するための手順なので source と command を省略しても、同じ結果を得ることが出来る。各自削除した source/command について、具体的にはそれぞれのオブジェクトファイルの生成の仕方が補われている。そして、もしソースとして「hoge.o」を指定した場合、「hoge.c」があれば make が cc によりコンパイルするという関係を知らないものに対しては、ソースとコマンドをきちんと定義するべきだと思う。

5 Level5

clean は大抵の makefile に定義されている内容である (コマンドやその引数は必要に応じてマクロを使うことが多い)。これは何をしているのか。Level1 でダウンロードした Makefile に類似の機能を記述し、実行せよ。その際、clean 実行前後で何がどう変わるのかを示せ。

5.1 実行結果

5.1.1 clean 実行前の結果

```
[nw0418:~/Desktop/make-sample] j04018% make
cc -c -o sample.o sample.c
cc -c -o add.o add.c
cc -c -o multi.o multi.c
cc sample.o add.o multi.o -o sample
[nw0418:~/Desktop/make-sample] j04018% ls
#multi.c# add.c makefile2 multi.o sample.o
0readme.txt add.o makefile3 sample texput.log
Makefile makefile1 multi.c sample.c
```

5.1.2 Makefile の書き換え

```
sample:  sample.o add.o multi.o
clean:
        rm -f a.out *.o *
```

5.1.3 clean 実行後の結果

```
[nw0418:~/Desktop/make-sample] j04018% make clean
rm -f a.out *.o *
[nw0418:~/Desktop/make-sample] j04018% ls
#multi.c# add.c makefile3 sample.c
0readme.txt makefile1 multi.c texput.log
Makefile makefile2 sample
```

5.2 考察

clean の実行前と実行後の ls を見てみると make コマンドで生成したファイルが削除されていることがわかる。つまり、clean をすることによって不必要なファイルなどを削除することが出来ると考えられる。

6 まとめ

makefile をすることによって作業の効率がよくなるってことがわかった。ふだん覚えるのがめんどくさいコマンドも makefile を使ってコマンドを簡単化できることを知り、是非やろう！と思った。

参考文献

[1] Makefile の書き方

http://www.tk.airnet.ne.jp/nagae/jp/comp/make/makefile.html

[2] make の第一歩

http://lagendra.s.kanazawa-u.ac.jp/ogurisu/manuals/make-intro/index.html