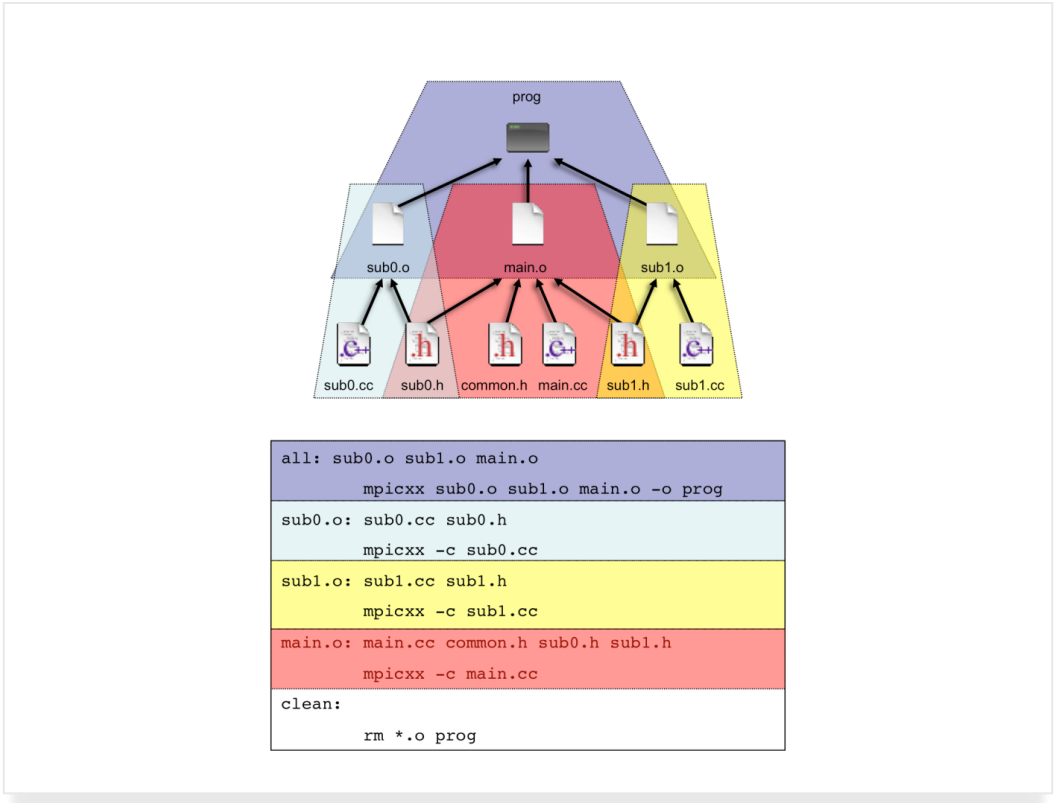
資料來源: https://mropengate.blogspot.com/2018/01/makefile.html

簡單學 makefile：makefile 介紹與範例程式

於 1/03/2018 05:28:00 下午 1 Comment
標籤： **Software Development-Linux/Unix**




```
all: sub0.o sub1.o main.o
        mpicxx sub0.o sub1.o main.o -o prog
sub0.o: sub0.cc sub0.h
        mpicxx -c sub0.cc
sub1.o: sub1.cc sub1.h
        mpicxx -c sub1.cc
main.o: main.cc common.h sub0.h sub1.h
        mpicxx -c main.cc
clean:
        rm *.o prog
```

2018.1.3 重整筆記

一、Make 簡介

在軟體開發中，make 是一個工具程式，經由讀取一個叫做 makefile 的檔案，自動化建構軟體。

編譯小型程式可用簡單的命令編譯或 shell script 編譯，但當程式很大且包含大量標頭檔和函式庫

時，就需要使用 makefile。makefile 會將程式分成好幾個模組，根據裡面的目標 (target)、規則 (rule) 和檔案的修改時間進行判斷哪些需要重新編譯，可以省去大量重複編譯的時間，這在大型程式中尤為有用。

另外，在閱讀大型程式碼時除了 Readme 檔案，Makefile 也能對整體架構有不錯的詮釋，先閱讀 makefile 是掌握程式碼架構一個良好的策略。

```
PATH  := node_modules/.bin:$(PATH)

source_files     := $(wildcard lib/*.ts)
build_files      := $(source_files:lib/%.ts=dist/%.js)
PACKAGE          := build/my-package-1.0.0.tgz

.PHONY: all

all: $(PACKAGE)

$(build_files): $(source_files) package.json
    npm i
    tsc

$(PACKAGE): $(build_files) .npmignore
    @mkdir -p $(dir $@)
    @cd $(dir $@) && npm pack $(CURDIR)
```

一個常見 makefile 大致的樣式圖。

二、Make 的工作流程

一般來說，當我們輸入 make 命令，會執行以下內容：

1. make 會在當前目錄下按順序找尋文件名為 GNUmakefile、makefile 或 Makefile 的文件。
2. 在 makefile 文件中的找到**第一個目標文件（target）**，並把這個文件作為最終的目標文件。
3. 如果沒找到或目標文件所依賴的文件，或修改時間要比目標文件新，則 make 將執行後面所定義的命令來生成這個文件，如此遞迴下去找到文件彼此的依賴關係，直到最終編譯出第一個目標文件。

[用心去感覺] make 只在意依賴性

如果最後被依賴的文件找不到，那麼 make 就會直接退出並報錯，而對於所定義的命令的錯誤或是編譯不成功，make 不予理會。

二、Makefile 主要內容

Makefile 裡主要包含了五個東西：顯式規則、隱式規則、變量定義、文件指示和註釋。

- **顯式規則：**顯式規則表示如何生成一個或多個目標文件。
- **隱式規則：**比較簡略地書寫 Makefile 規則，例如規則中有 .o 文件，make 會自動的把 .c 文件也加入依賴關係中。
- **變數定義：**類似 C 語言中的 define，定義的變數都會置換到引用位置上。
- **文件指示：**
  - 類似 C 語言中的 include，一個 Makefile 中引用另一個 Makefile，如 include makefile.inc。
  - 類似 C 語言中的 預編譯 #if，根據某些情況指定 Makefile 中的有效部分。
- **註釋與換行：**Makefile 中只有行註釋，用 # 符號；換行則是使用 \ 符號 。

1. 顯式規則

最重要的是 Makefile 規則，其基本結構如下：

- **目標 (Target) :** 一個目標檔，可以是 Object 檔，也可以是執行檔，還可以是一個標籤。
- **依賴 (Dependency, Prerequisites) :** 要產生目標檔 (target) 所依賴哪些檔。
- **命令 (Command) :** 建立專案時需要執行的 shell 命令。命令部分的每行的縮進必須要使用 **Tab 鍵**而不能使用多個空格。

```
#用「井」號表明注釋。
target（要生成的文件）: dependencies（被依賴的文件）
#命令前面用的是「tab」而非空格。誤用空格是初學者容易犯的錯誤。
        命令1
        命令2
        命令3
          .
          .
          .
        命令n
#可以使用「\」表示續行。注意，「\」之後不能有空格。
```

[用心去感覺] 顯式 make 命令

像 clean 這種沒有被第一個目標文件直接或間接關聯，那麼它後面所定義的命令將不會被自動執行，不過我們可以顯式要求 make 執行。即 make clean。

[用心去感覺] 偽目標 .PHONY

.PHONY 會將目標設成假目標，使 make 目錄下沒有目標檔案或目標檔案為最新時，仍可執行 make <target>。.PHONY 寫法也可以讓程式設計師知道哪些工作目標不是針對檔案，增加可讀性。

Make 預設的假工作目標有 all, install, clean, distclean, TAGS, info 和 check。

一個常用的情況是 make clean，因為 clean 標籤下的 rm 命令並不產生 clean 文件：

```
.PHONY: clean
clean:
 rm *.o
```

2. 變數使用

變數宣告時要使用 = 或 := 給予初始值 (注意兩者在代換時稍有不同)，如 obj = hello.o foo.o，取用時寫成 $(obj) 或 ${obj}。如果我們想定義一些比較類似的文件，可以使用 Unix-like 的 * ， ? 和 ~ 。

一個範例如下：

```
objects = program.o foo.o utils.o
program : $(objects)
    cc -o program $(objects)
$(objects) : defs.h
```

在一些 make 中常使用自動化變數來簡寫規則（讓 makefile 難讀的兇手之一QQ）：

- $@：目前的目標項目名稱。
- $<：代表目前的相依性項目。
- $*：代表目前的相依性項目，但不含副檔名。
- $?：代表需要重建（被修改）的相依性項目。

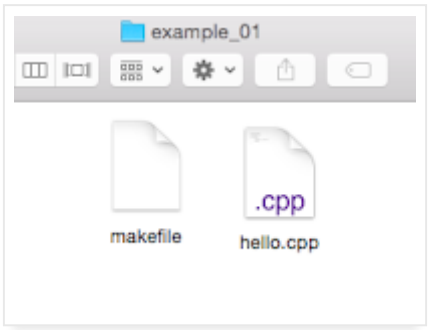
另外在 makefile 規則中所用的萬用配對字元是% ，因此一個使用各種變數代換技巧的 makefile 可能如下例所示：

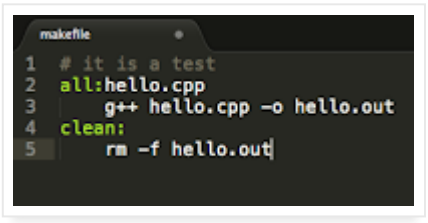
```
CC:=gcc
exe:=main
obj:=main.o a.o b.o c.o

all:$(obj)
 $(CC) -o $(exe) $(obj)
%.o:%.c
 $(CC) -c $^ -o $@

.PHONY:clean
clean:
 rm -rf $(obj) $(exe)
```

三、Makefile 簡單練習

Example 1：最簡單的 makefile 練習

1. 新建 makefile 和 hello.cpp 兩個檔案





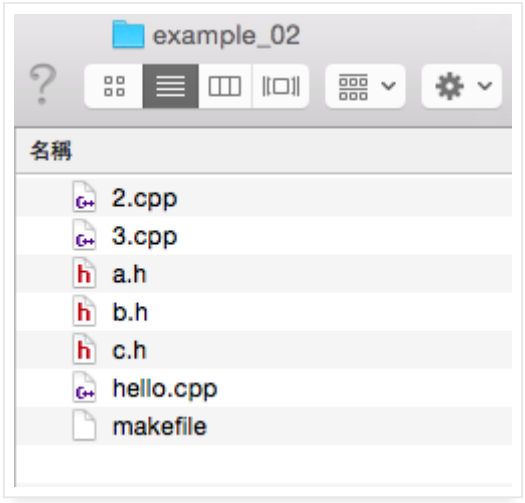
2. 於命令列執行 make 檔案

- 執行 make ：會把名為 hello.out 的執行檔編譯出來。
- 執行 make clean：會把名為 hello.out 的執行檔刪除。

```
opengate@Opengate-MacBook-Air:[~/Desktop]: ls
hello.cpp       makefile
opengate@Opengate-MacBook-Air:[~/Desktop]: make
g++ hello.cpp -o hello.out
opengate@Opengate-MacBook-Air:[~/Desktop]: ls
hello.cpp       hello.out       makefile
opengate@Opengate-MacBook-Air:[~/Desktop]: make clean
rm -f hello.out
opengate@Opengate-MacBook-Air:[~/Desktop]: ls
hello.cpp       makefile
opengate@Opengate-MacBook-Air:[~/Desktop]:
```

Example 2：檔案多一點的 makefile 練習

1. 新建 a.h b.h c.h hello.cpp 2.cpp 3.cpp makefile 六個檔案。hello.cpp 與之前相同，makefile 改成以下檔案內容。



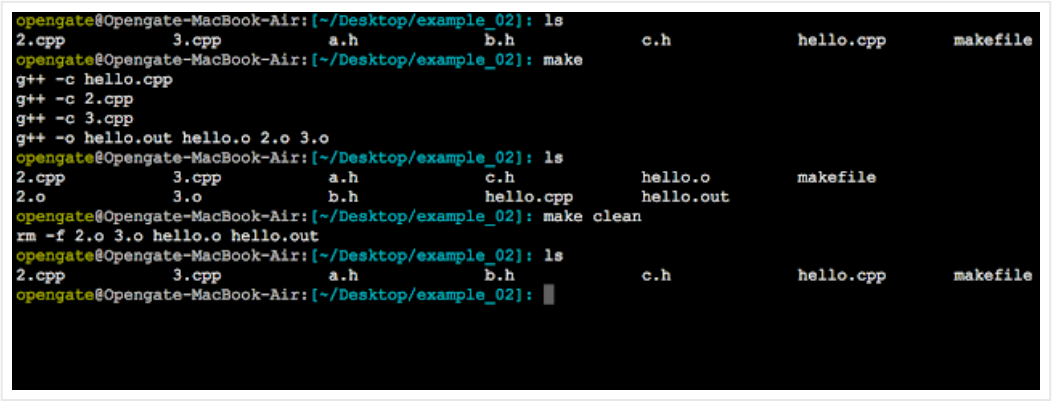

```
makefile
1  hello.out:hello.o 2.o 3.o
2      g++ -o hello.out hello.o 2.o 3.o
3  hello.o:hello.cpp a.h
4      g++ -c hello.cpp
5  2.o:2.cpp b.h
6      g++ -c 2.cpp
7  3.o:3.cpp a.h b.h c.h
8      g++ -c 3.cpp
9
10 clean:
11     rm -f 2.o 3.o hello.o hello.out
12
```

2. 於命令列執行 make 檔案

- 執行 make：會把名為 hello.out 的執行檔編譯出來。
- 執行 make clean：會把名為 hello.out 的執行檔給刪除。

make 會去找第一行的必要條件：hello.o 2.o 3.o，當在目錄裡卻沒這三個檔，所以 make 會在 makefile 往下找 hello.o, 2.o 和 3.o，分別把這三個檔案給編出來。

如果在前後兩次編譯之間，hello.cpp 和 a.h 均沒有被修改，而且 hello.o 還存在的話，就 make 執行時就不會再重新編譯這些檔案。



Example 3：常見的 makefile 使用方式

1. 使用的檔案與 example2 相同，makefile 改成下圖。這個範例和平常別人撰寫的 makefile 檔案比較像，可以稍微用心琢磨。

gcc 有使用許多的參數，意思如下：

- -c :編譯但不進行鏈結
- -ansi：程式要求依據 ansi 標準，增加可移植性。
- -I：追加 include 檔案的搜尋路徑
- -Wall：編譯時顯示所有的警告訊息
- -g：編入除錯資訊 (要使用 GDB 除錯一定要加)。
- -O ：表示最佳化的程度，預設是 -O1，可以指定 -O2 或 -O3，數字越大最佳化程度越高。

makefile

```
all: hello.out

#declare variables
CC = g++
INSTDIR = /usr/local/bin
INCLUDE = .
CFLAGS = -g -Wall -ansi
LIBS += -framework CoreFoundation


hello.out: hello.o 2.o 3.o
    $(CC) -o hello.out hello.o 2.o 3.o
hello.o: hello.cpp a.h
    $(CC) -I$(INCLUDE) $(CFLAGS) -c hello.cpp
2.o: 2.cpp a.h b.h
    $(CC) -I$(INCLUDE) $(CFLAGS) -c 2.cpp
3.o: 3.cpp b.h c.h
    $(CC) -I$(INCLUDE) $(CFLAGS) -c 3.cpp

clean:
    rm -f hello.o hello.out 2.o 3.o

install: mytest
    @if[   -d $(INSTDIR)   ]; \
    then   \
        cp hello.out $(INSTDIR);\
        chmod a+x $$(INSTDIR)/hello.out; \
        chmod og-w $(INSTDIR)/hello.out; \
        echo "Installed in $(INSTDIR)";\
    else \
        echo "Sorry, $(INSTDIR) does not exist";\
    fi
```

2. 於命令列「make」檔案

- 執行 make：會把名為 hello.out 的執行檔編譯出來。
- 執行 make clean：會把名為 hello.out 的執行檔給刪除。

```
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]: ls
2.cpp           3.cpp           a.h             b.h             c.h             hello.cpp       makefile
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]: make
g++ -I. -g -Wall -ansi -c hello.cpp
g++ -I. -g -Wall -ansi -c 2.cpp
g++ -I. -g -Wall -ansi -c 3.cpp
g++ -o hello.out hello.o 2.o 3.o
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]: ./hello.out
Example_03 is correct in hello.cpp!!
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]: ls
2.cpp           3.cpp           a.h             c.h             hello.o         makefile
2.o             3.o             b.h             hello.cpp       hello.out
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]: make clean
rm -f hello.o hello.out 2.o 3.o
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]: ls
2.cpp           3.cpp           a.h             b.h             c.h             hello.cpp       makefile
opengate@Opengate-MacBook-Air:[~/Desktop/example_03]:
```

References

Maxsolar's Linux Blog - Makefile 範例教學
http://maxubuntu.blogspot.tw/2010/02/makefile.html

跟我一起寫Makefile:MakeFile介紹
http://wiki.ubuntu.org.cn/index.php?title=%E8%B7%9F%E6%88%91%E4%B8%80%E8%B5%B7%E5%86%99Makefile:MakeFile%E4%BB%8B%E7%BB%8D&variant=zh-hant

csie - makefile
http://www.csie.nuk.edu.tw/~kcf/course/97_Spring/Embedded%20System/8-Makefile.pdf

(make的参数) - Driver - Embeded Linux
http://www.360doc.com/content/10/0918/18/3444631_54660975.shtml