CHUYÊN ĐỀ HỆ ĐIỀU HÀNH LINUX

Tuần 10: Tạo makefile (phần 2)

GVLT: NGUYỄN Thị Minh Tuyền

Nội dung

- Makefile (tiếp theo) :

1. **Các nguyên tắc về xây dựng quy tắc**
2. **Các chỉ thị (Directive)**
3. **Các biến tự động (Automatic Variables)**
4. **Khối lệnh**

Nội dung

- `Makefile` (tiếp theo) :

1. **Các nguyên tắc về xây dựng quy tắc**
2. Các chỉ thị (Directive)
3. Các biến tự động (Automatic Variables)
4. Khối lệnh

Các nguyên tắc (principal) của quy tắc/luật (rule)

- Nhắc lại : cú pháp chung của một quy tắc :

```
target :prerequisites ...
     recipe
     ...
     ...
```

- hoặc theo cách tương đương như sau:

```
target :prerequisites ; recipe
```

- make thực hiện qua 2 pha :

1. Đọc toàn bộ file và ghi nhớ các quy tắc
2. Tìm kiếm quy tắc được yêu cầu

Nguyên tắc 1

- Tất cả các phụ thuộc phải
  - Tồn tại như một target trong `Makefile`
  - Và/hoặc tương đương với một file đã tồn tại
- Ví dụ : ta tạo file `Makefile` này trong một thư mục rỗng:

  a : b c

  b : c

  `make` thất bại:

```
make: *** No rule to make target `c', needed by `b'.  Stop.
```

- Giải pháp :
  - thêm vào một target
  - hoặc tạo một file ".c" trong thư mục.

Nguyên tắc 2

- Nếu một target là một file đã tồn tại, và nếu target này không có phụ thuộc, lệnh không được thực thi.
- Ví dụ :

```
c :
 echo "do c"


$ make --silent
do c
$ touch c
$ make
make: 'c' is up to date.
```

Nguyên tắc 3

- Một target có thể không tương ứng với một file.
- Ví dụ : `all`, `clean`, `depend`
  ➔target "giả" (phony targets)
- Để chỉ rõ các target "giả" trong `Makefile` :

```
.PHONY : all clean
all : prog1 prog2
clean :
   rm *.o prog1 prog2
```

- Mục tiêu : không kiểm tra sự tồn tại của file
  - tránh xung đột với tên file
  - cải thiện performance

Nguyên tắc 4

- `make` hiển thị mỗi lệnh trước khi thực thi :

```
a:
    echo "do a"
```

```
$ make
echo "do a"
do a
```

- Ta có thể ngăn hiển thị lệnh bằng cách thêm tuỳ chọn `--silent`:

```
a:
    echo "do a"
```

```
$ make --silent
do a
```

- Hoặc sử dụng @ trước lệnh :

```
a:
    @echo "do a"
```

```
$ make
do a
```

Nguyên tắc 5

- Một target chỉ được thực hiện một lần ;
- Nếu nhiều phụ thuộc dựa vào cùng một target, lệnh chỉ được thực hiện lần đầu tiên.
- Ví dụ :

```
a : b c
     echo "do a"
b : c
    echo "do b"
c :
    echo "do c"
```

```
$ make --silent
do c
do b
do a
```

Nguyên tắc 6

- Nhiều target có thể chia sẻ cùng một quy tắc.
- Ví dụ :

```
a : b c
    echo "do a"
b c :
    echo "do b or c"

$ make --silent
do b or do c
do b or do c
do a
```

Nguyên tắc 7

- Một target có thể xuất hiện nhiều lần, với các phụ thuộc giống hoặc khác nhau, các phụ thuộc được nối lại với nhau.
- Tuy nhiên : chỉ một khối lệnh bởi một target.
- Ví dụ 1 :

```
a : b c
    echo "1"
a : c e
b c e :
```

Tương đương với

```
a : b c e
        echo "1"
b c e :
```

- Ví dụ 2:

```
a : b c
    echo "1"
a : c e
    echo "2"
b c e :
```

Sai :

có hai khối lệnh bởi một target "a".

Nguyên tắc 8

- Để đặt nhiều khối lệnh cho cùng một target, ta chỉ cần thêm "::" vào chỗ ":" cho tất cả các lần xuất hiện của target.
- Ví dụ:

```
a :: b c
	echo "a1"
a :: c e
	echo "a2"
b : ; echo "b"
c : ; echo "c"
e : ; echo "e"
```

```
$ make --silent
b
c
a1
e
a2
```

→ Các khối lệnh được thực hiện theo thứ tự xuất hiện.

Nguyên tắc 9

- "::" tạo một target "giả".
  Có hai cách để khai báo một target "giả":

```
.PHONY : a
a : ....
    ....
```

hoặc

```
a :: ....
     ....
```

Nguyên tắc 10

- Mặc định, lệnh make tự động dừng lại khi một lệnh thất bại. Ta có thể ngăn việc tự động dừng bằng cách thêm "-" trước khối lệnh.

```
a : b c
    @echo "a"
b :
    @echo "b"
c :
    false
```

Nội dung

- Makefile (tiếp theo) :

1. Các nguyên tắc về xây dựng quy tắc
2. **Các chỉ thị (Directive)**
3. Các biến tự động (Automatic Variables)
4. Khối lệnh

Các chỉ thị

- Lệnh make cho phép việc sử dụng các chỉ thị :
  - include các file
  - các biến
  - các điều kiện
  - ...
- Luôn đặt ở đầu dòng, không bao giờ đặt trong khối lệnh.

Include file

`include file ...`

- Include file, thất bại khi thiếu file.

`-include file ...`

- Include file; nếu thiếu file, không báo lỗi cũng không đưa ra cảnh báo.

Định nghĩa biến trong make

- Định nghĩa một biến và giá trị của nó, cho phép khoảng trắng

  `variable = value`

- Nối giá trị vào cuối

  `variable += value`

- Thay thế giá trị :

  `$(variable)` hoặc `${variable}` hoặc `$v` nếu là một ký tự

- Ta có thể cập nhật một biến trong một khối lệnh.

Các biến hữu ích

```
SHELL = /bin/bash # Quan trọng : shell được sử dụng bởi make
CC    = gcc
RM    = rm -f
CFLAGS = -Wall -O2
CPATHS = -I.
LFLAGS = -lm -lmylibrary
LPATHS = -L.
EXEC = myprogram
OBJS = module1.o module2.o
$(EXEC) : $(OBJS)
    $(CC) $(OBJS) $(LPATHS) $(LFLAGS) -o $(EXEC)
```

- Chú ý: $(LFLAGS) đặt sau $(OBJS)

Câu điều kiện tồn tại định nghĩa biến

```
ifdef variable     # hoặc ifndef
    ....
else               # option
    ....
endif
```

- Bao gồm các dòng .... tuỳ theo biến được định nghĩa hay chưa.
- Ví dụ:

```
CFLAGS = -Wall -O2
# Bỏ chú thích để thực hiện debug với gdb
#DEBUG =
ifdef DEBUG
    CFLAGS += -g
endif
toto.o : toto.c
     gcc $(CFLAGS) -c toto.c
```

Câu điều kiện so sánh giá trị [1]

```
ifeq "value1" "value2"    # hoặc ifneq
....
else            # option
....
endif
```

- Include các dòng `....` tuỳ theo các giá trị có bằng nhau hay không.
- Biến thể : `ifeq (value1, value2)`

Câu điều kiện so sánh giá trị [2]

- Ví dụ:

```
CFLAGS = -Wall -W
# CVER = ansi ou c99
CVER = c99
ifeq "$(CVER)" "c99"
   CFLAGS += -std=c99
else
   CFLAGS += -ansi
endif
toto.o : toto.c
   gcc $(CFLAGS) -c toto.c
```

Nội dung

- Makefile (tiếp theo) :

1. Các nguyên tắc về xây dựng quy tắc
2. Các chỉ thị (Directive)
3. **Các biến tự động (Automatic Variables)**
4. Khối lệnh

Các biến tự động

- Các biến định nghĩa bởi make tuỳ theo ngữ cảnh
  - $@ target hiện tại
  - $* tiền tố của $@ (rỗng nếu phần mở rộng không được nhận diện)
  - $< phụ thuộc đầu tiên
  - $^ danh sách các phụ thuộc
- Chỉ sử dụng trong khối lệnh

Ví dụ

```
mod1.o : mod1.c
    $(CC) $(CPATHS) $(CFLAGS) -c mod1.c

mod2.o : mod2.c
    $(CC) $(CPATHS) $(CFLAGS) -c mod2.c

prog1 : mod1.o mod2.o
    $(CC) mod1.o mod2.o $(LPATHS) $(LFLAGS) -o prog1
```

- Ta có thể viết lại như sau :

```
mod1.o : mod1.c

mod2.o : mod2.c

mod1.o mod2.o :
    $(CC) $(CPATHS) $(CFLAGS) -c $*.c

prog1 : mod1.o mod2.o
    $(CC) $^ $(LPATHS) $(LFLAGS) -o $@
```

Quy tắc chung

- Target đặc biệt `.c.o :` (cú pháp cũ)

  tất cả `.o` phụ thuộc vào `.c` có cùng tên

- Trong ví dụ trước :

```
mod1.o : mod1.c
mod2.o : mod2.c
mod1.o mod2.o :
    $(CC) $(CPATHS) $(CFLAGS) -c $*.c
```

- ta có thể viết lại thành :

```
.c.o :
    $(CC) $(CPATHS) $(CFLAGS) -c $*.c
```

Target với pattern

- Cú pháp GNU-make (phổ biến hơn) :
  - Tất cả các target có thể chứa một "%" (nhiều nhất), tương ứng với tất cả các chuỗi không rỗng.
- Ứng dụng : %.o : %.c rõ ràng hơn .c.o :
- Ví dụ ở slide trước trở thành :
-
```
%.o : %.c
	$(CC) $(CPATHS) $(CFLAGS) -c $*.c
```

Nội dung

- Makefile (tiếp theo) :

1. Các nguyên tắc về xây dựng quy tắc
2. Các chỉ thị (Directive)
3. Các biến tự động (Automatic Variables)
4. **Khối lệnh**

Khối lệnh

- Trong khối lệnh của một quy tắc:
  - mỗi dòng không rỗng bắt đầu bởi một `tab`
  - ta có thể nhảy dòng
  - ta có thể chú thích với #
- Shell được sử dụng được định nghĩa bởi SHELL :

```
SHELL = /bin/bash
```

Thay thế trong shell

"$" đã được sử dụng bởi các biến của make
→ để thực hiện các thay thế của shell, ta sử dụng "$$" .

```
SHELL = /bin/bash
FOO = bar
try ::
    @echo "FOO = $(FOO)"
    @a="hello" ; echo "$$a"
    @echo "3 + 4 = $$((3+4))"
    @echo "today is $$(date)"
    @echo "PID is $$$$"
```

Một shell bởi một dòng

- Trong một khối lệnh, `make` thực hiện mỗi dòng trong một shell mới.
- Yếu điểm : mất các biến shell

```
SHELL = /bin/bash
try ::
        @a="hello"
        @echo "a = $$a"
```

```
$ make try
a=
```

Giải pháp : nối các dòng lại với nhau bởi "\"

```
SHELL = /bin/bash
try ::
        @a="hello" ;\
        echo "a = $$a"
```

```
$ make try
a = hello
```

Makefile với thư mục con

- Cho các thư mục con Bim, Bam et Boum ; mỗi thư mục chứa một Makefile với các target all, clean và depend.
- Makefile tổng quát :

```
SHELL = /bin/bash
MAKE = make
DIRS = Bim Bam Boum
all clean depend ::
@for D in $(DIRS) ; do \
    (cd "$$D" && $(MAKE) $@) ;\
done
```

Xem thêm

- https://www.gnu.org/software/make/manual/html_node/index.html#SEC_Contents
- https://www.coursera.org/lecture/introduction-embedded-systems/6-make-18etg

Câu hỏi ?