A make használata

Baráth Áron

Eötvös Loránd Tudományegyetem
Informatikai Kar

2014. március 17.

Tartalom

A feladat

- Egy (nagy) projekt buildelése minimális újrafordítással.
- Függőségek leírása.
- Fordítás automatizálása.
- Shell szkripttel túl körülményes a dátumok kezelése!

Kézi fordítás: 1 cpp

```
$ g++ hello.cpp -o hello
```

Kézi fordítás: 2 cpp

```
$ g++ -c egyik.cpp -o egyik.o
$ g++ -c masik.cpp -o masik.o
$ g++ -o hello egyik.o masik.o
```

Kézi fordítás: 2 cpp, kapcsolókkal

```
$ g++ -c egyik.cpp -Wall -Werror -ansi -pedantic \
  -D MACRO -o egyik.o

$ g++ -c masik.cpp -Wall -Werror -ansi -pedantic \
  -D MACRO -o masik.o

$ g++ -o hello egyik.o masik.o
```

Shell script

```
if [ egyik.cpp -nt egyik.o ]; then
  g++ -c egyik.cpp -Wall -Werror -ansi -pedantic \
    -D MACRO -o egyik.o
fi
if [ masik.cpp -nt masik.o ]; then
  g++ -c masik.cpp -Wall -Werror -ansi -pedantic \
    -D MACRO -o masik.o
fi
if [ egyik.o -nt hello ] || [ masik.o -nt hello ]
then
  g++ -o hello egyik.o masik.o
fi
```

Tartalom

Felépítés

```
cel: elofeltetel1 elofeltetel2 ...
<TAB>  parancs1
<TAB>  parancs2
<TAB>  ...
```

Felépítés

- Szabályok
- Kommentek
- Változók
- Direktívák

Szabályok

Explicit szabály: egy fájl elkészítése.

```
hello: hello.cpp
    g++ hello.cpp -o hello
```

A legelső cél az alapértelmezett.

```
$ make hello
$ ./hello
```

Egyszerű Makefile

```
hello: egyik.o masik.o
    g++ egyik.o masik.o -o hello

egyik.o: egyik.cpp
    g++ -c egyik.cpp -Wall -Werror -ansi -pedantic \
      -D MACRO -o egyik.o

masik.o: masik.cpp
    g++ -c masik.cpp -Wall -Werror -ansi -pedantic \
      -D MACRO -o masik.o
```

Jó ez így?

Szabályok

Implicit szabály: több fájl elkészítése minta-szabályok (pattern rules) segítségével.

```
hello: egyik.o masik.o
    g++ egyik.o masik.o -o hello

%.o: %.cpp
    g++ -c $< -o $@
```

- A % helyére bármi bekerülhet.
- A % ugyanazt jelenti az előfeltételben és a célban is.

Szabályok

Nem-fájl (phony) szabályok: mindig out-of-date állapotúak.

```
clean:
    rm -f *.o
```

Ne legyen clean nevű fájlunk!
Tipikus phony szabályok:

- `default` (ezt szokták a Makefile elejére tenni)
- `all`
- `install`
- `clean`
- `info`

„Dupla-kettőspont” szabályok

- Angolul: *double-colon rules*
- Ha előfeltételben szerepel, mindegyik előfordulást végrehajtja.
- A „normál” szabálynak egyedinek kell lennie, ha mégsem az, akkor hibát kapunk.
- Nem lehet keverni egy szabály típusát.
- Csak akkor használjuk, ha feltétlen szükséges!

```
rule::
    echo Egyik

rule:
    echo Masik
```

„Dupla-kettőspont” szabályok

```
rule::
	@echo Egyik

rule::
	@echo Masik

$ make rule
Egyik
Masik
```

Kommentek

A #-tól a sor végéig.

```
# a hello program elkészítése
hello: egyik.o masik.o
    g++ egyik.o masik.o -o hello

# a cpp források fordítása
%.o: %.cpp
    g++ -c $< -o $@
```

Változók

```
OBJS = egyik.o masik.o
CXXFLAGS = -Wall -Werror -ansi -pedantic

hello: $(OBJS)
    g++ $(OBJS) -o hello

%.o: %.cpp
    g++ -c $< $(CXXFLAGS) -o $@
```

Automatikus változók

`$@` a szabály céljának neve

`$<` az első feltétel neve

```
OBJS = egyik.o masik.o

hello: $(OBJS)
    g++ $(OBJS) -o hello

%.o: %.cpp
    g++ -c $< -o $@
```

Automatikus változók

- `$@` a szabály céljának neve
- `$%` a szabály célpontja, ha archív
  foo.a(bar.o)
  $% = bar.o
  $@ = foo.a
- `$<` az első feltétel neve
- `$?` az összes előfeltétel, amelyek a célnál fiatalabbak
- `$+` ugyanaz, mint a $?, de megengedi a duplikátumokat
- `$*` a minta (vagyis a % értéke)
- `$(@D)` a cél könyvtár része (relatív)
- `$(@F)` a cél fájl része (általában u.a., mint $@)

Változó manipuláció

Minta alapú helyettesítés. Tartalmazhat % karaktert, ami egy tetszőleges szöveget helyettesít.

```
SRC = egyik.cpp masik.cpp
OBJS = $(SRC:.cpp=.o)

hello: $(OBJS)
    g++ $(OBJS) -o hello

%.o: %.cpp
    g++ -c $< -o $@
```

Eredmény: `egyik.o masik.o`

Változó manipuláció

Helyettesítés függvénnyel.

```
SRC = egyik.cpp masik.cpp
OBJS = $(SRC:.cpp=.o)
INC = dir1 dir2

hello: $(OBJS)
    g++ $(OBJS) -o hello

%.o: %.cpp
    g++ -c $< $(patsubst %,-I%,$(INC)) -o $@
```

Eredmény: `-Idir1 -Idir2`
Ekvivalens: `$(INC:%=-I%)`

Változó manipuláció

```
SRC = egyik.cpp
SRC += masik.cpp
OBJS = $(SRC:.cpp=.o)

hello: $(OBJS)
    g++ $(OBJS) -o hello

%.o: %.cpp
    g++ -c $< -o $@
```

Függvények

`$(function arguments...)`
`${function arguments...}`

$(subst from,to,text) egyszerű csere
$(strip str) fehér szóközök „rendberakása”
$(findstring find,in) az értéke `find`, ha szerepel a `in`-ben
$(sort list) rendezés
$(word n,text) *n*-edik elem
$(words text) szavak száma

- `$(word 1,list)`
- `$(word $(words list),list)`
- `$(firstword list)`, `$(lastword list)`

$(wildcard pattern...) fájlok, amelyek megfelelnek a mintáknak
$(shell command) shell parancs végrehajtása (nem célszerű)

Direktívák

```
include fájl1 fájl2

CFLAGS += -Wall -Werror

ifeq ($(TARGET),Linux)
    CFLAGS += -D SEPARATOR='/'
else
    CFLAGS += -D SEPARATOR='\\'
endif
```

Futtatás: `make TARGET=Linux`

- Lehet szabály is a feltételes részben.
- A feltételben lehet „lokális” változó is.

Működés

- Ha egy parancs végrehajtása sikertelen, akkor megáll a `make` futása is.
- Saját program esetén: a `main` visszatérési értéke legyen 0 (a nem 0 hibát jelent).
- A végrehajtott parancsokat a `make` kiírja a standard kimenetre. Ezt a `@` jellel el lehet nyomni.

Működés

- A parancs eredményét figyelmen kívül is lehet hagyni a - (mínusz) jellel.
  - Ezzel lehet opcionális `include`-ot megvalósítani.

```
-include nem_letezo_file # nem okoz hibát

rule:
    -g++ -c -xxxxx hello.cpp
```

Több Makefile

`$ make` – az alábbiak közül választ: `GNUmakefile`, `makefile`, `Makefile`

```
$ make -f masik_makefile.mk
```

vagy

```
$ make --file=masik_makefile.mk
```

vagy

```
$ make --makefile=masik_makefile.mk
```

További kapcsolók

`$ make -n` Kiírja a parancsokat, de nem hajt végre semmit.

`$ make -d` Debug infó kiírása.

`$ make -j N` *N* szálú végrehajtás (*N* opcionális).

`$ make -C dir` Futtatás a *dir* könyvtárban. A `cd` helyett használjuk.

~~`cd dir && make && cd ..`~~

```
make -C dir
```

gcc/g++ támogatás

- A gcc/g++ segítségével lehet a függőségeket generálni.
- Csak az idézőjelek közé írt include-ok!

```
$ g++ -MM hello.cpp
hello.o: hello.cpp
```

- Célszerű átirányítani egy file-ba.
- Utána lehet `include`-olni a `Makefile`-ban.

```
$ g++ -MM hello.cpp > hello.dep
```

Automatizálás

- makedepend
- Imake
- autoconf
- automake

Vége