Makefile od základu až po pokročilé

Karel Kočí

14.10.2018

Co je Makefile?

- Předpis pro program make
- Sada skriptů popisujících vznik souborů
- Sada závislostí mezi soubory
- A vše ostatní ...

Pozor: gmake vs make!

Jednoduchý Makefile

```
FILE=pres

$(FILE).pdf: $(FILE).tex $(patsubst %.svg,%.pdf,$(wildcard *.svg))
	pdflatex -shell-escape $<

%.pdf: %.svg
	inkscape -D -z --file=$< --export-pdf=$@ --export-latex

clean:
	ls | grep -v -E "($(FILE).tex|makefile|scheme|svg|png|eps)$$" \
		| xargs rm -rf
```

Základní syntaxe

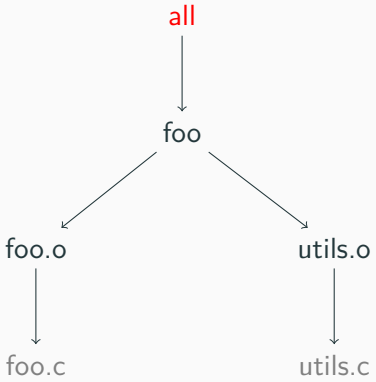
```
.PHONY: all
all: foo
        ;

foo: utils.o foo.o

foo:
        cc -o $@ $^

foo.o utils.o: %.o: %.c
        cc -c -o $@ $<
```

Závislosti a rebuild



Automatické proměnné

```
foo: main.o
foo: foo.o utils.o
        cc -o $@ $^
```

- `$@`
  foo
- `$<`
  foo.o
- `$^`
  utils.o foo.o main.o

`www.gnu.org/software/make/manual/html_node/Automatic-Variables.html`

Proměnné

```
SRC := main.c foo.c
OBJ = $(patsubst %.c,%.o,$(SRC))
SRC += utils.c
```

- FOO = přiřazení ale doplnění až při použití
- FOO := přiřazení a okamžité doplnění
- FOO += přidání obsahu do proměnné
- FOO ?= přiřaď pokud proměnná neexistuje

Generická pravidla

```
%.h: %.bin
        xxd -i $< > $@

goo.h: %.h: %.bin
        sed 's/foo/goo/g' $< | xxd -i - > $@

conf.h: conf.h.m4
        m4 $< > $@
```

```
$ make foo.h
$ make goo.h
$ make conf.h
```

Podmínky

```
ifeq ($(DEBUG),y)
        CFLAGS += -ggdb
endif

ifdef DEBUG
        CFLAGS += -ggdb
endif
```

Funkce

```
SRC := $(wildcard *.c)
OBJ := $(patsubst %.c,%.o,foo.c bar.c)
```

www.gnu.org/software/make/manual/html_node/Functions.html#Functions

define, call a eval

```
define PROJECT
$(1): $(1)_SRC
        cc $$($(1)_LDFLAGS) -o $@ $^
$(1)_SRC: %.o: %.c
        cc $$($(1)_CFLAGS) -c -o $@ $<
endef

foo_SRC := foo.c utils.c
foo_LDFLAGS := -lm
foo_CFLAGS := -Iinclude
$(eval $(call PROJECT,foo))
```

Doporučení a tipy

MAKEFLAGS a deaktivace vestavěných pravidel

```
MAKEFLAGS += --no-builtin-rules --no-builtin-variables
```

Q a skrývání příkazů

```
Q ?= @

clean:
        @echo " CLEAN build"
        $(Q)$(RM) -r build
```

O a výstupní složka

```
O ?= .
$(O)/foo: foo.c
        @mkdir -p $(@D)
        cc -o $@ $^
```

```
FOO_PATH = "../foo-project"
MAKEARGS := -C "$(FOO_PATH)" O="$(shell pwd)" --no-print-directory
Q ?= @
.PHONY: all $(MAKECMDGOALS)
all $(MAKECMDGOALS):
        $(Q)$(MAKE) $(MAKEARGS) $@
```

Generované závislosti

```
DEP := $(patsubst %.c,$(O)/%.d,$(SRC))

ifeq (,$(filter clean,$(MAKECMDGOALS)))
-include $(DEP)

$(DEP): $(O)/%.d: src/%.c
        @mkdir -p "$(@D)"
        @echo " DEP   $@"
        $(Q)$(CC) -MM -MG -MT '$*.o $@' $(CFLAGS) $< -MF $@

endif
```

Přepisovatelné nástroje

```
PREFIX ?= avr-
CC ?= $(PREFIX)gcc
LD ?= $(PREFIX)ld
```

HOST vs TARGET

```
HOST ?=
TARGET ?= avr
HOST_CC ?= $(HOST)gcc
TARGET_CC ?= $(TARGET)gcc
```

Automatické odstranění mezi-souborů!

```
pres.pdf: pres.tex $(patsubst %.svg,%.pdf,$(wildcard *.svg))
	pdflatex -shell-escape $<

%.pdf: %.svg
	inkscape -D -z --file=$< --export-pdf=$@ --export-latex

%.svg: %.dot
	dot -Tsvg $< > $@
```

Pár bodů na závěr

- Implementujte help target
- Vyvarujte se definici targetů v define
- Nikdy nepředpokládejte pořadí provádění
- Počátejte s cross-kompilací
- A pokud má jazyk který používáte již existující build systém tak použijte ten

Děkuji za pozornost.

`www.gnu.org/software/make/manual/make.html`