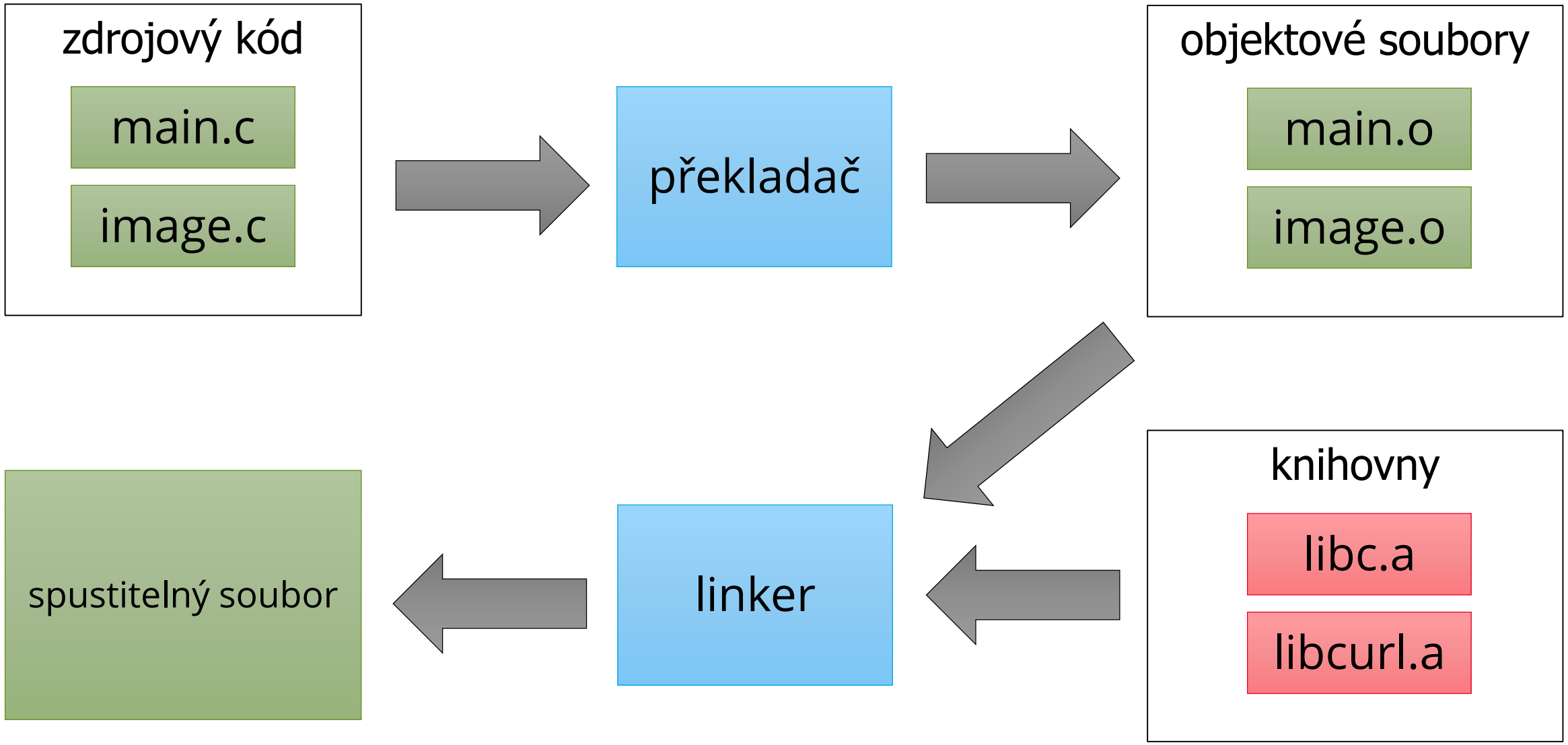
Sestavení programů, Make

Michal Wiglasz

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
iwiglasz@fit.vutbr.cz

12. 3. 2019
Praktické aspekty vývoje software (IVS)

Sestavení programu



Proces sestavení programu je rozumné zautomatizovat:

- stejně jako jakýkoliv opakující se proces
- protože je to rychlejší, spolehlivější a méně otravné

Spousta nástrojů:

- C/C++ – Make, CMake…
- Java – Ant, Maven, Gradle…
- JS – Grunt, Gulp, Yarn…

- (A to vše třeba na nějakém CI serveru a napojené na GIT – Jenkins, Travis…)

- Automatizace překladu (nejen) programů
- Nezávislý na jazyku, multiplatformní, prověřený časem (1977)
- Detekuje změny a přeloží pouze změněné části
- Pravidla v souboru `GNUmakefile`, `makefile`, `Makefile`

Makefile se skládá z:

- cílů (targets)
- závislostí (prerequisites)
- příkazů pro vytvoření cílů ze závislostí (recipes)

Příklady použití:

`make [cíl1] [cíl2]`

- vykoná `cíl1` a `cíl2` (výchozí je první cíl zapsaný v `Makefile`)

`make --always-make [cíl]` / `make -B [cíl]`

- vykoná vše, tj. přeloží i nezměněné soubory

`make --just-print [cíl]` / `make --dry-run [cíl]` / `make -n [cíl]`

- pouze vypíše, co by se dělo, ale nic reálně nespustí

`make --jobs [počet] [cíl]` / `make -j [počet] [cíl]`

- spouští příkazy paralelně (až daný počet příkazů, případně bez omezení)

`make --print-data-base [cíl]` / `make -p [cíl]`

- navíc vypíše proměnné, pravidla…

`make --makefile rules.mk [cíl]` / `make -f rules.mk [cíl]`

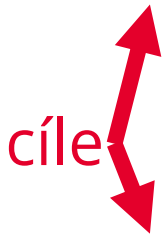
- použije pravidla z jiného souboru než `Makefile`

Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile

all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

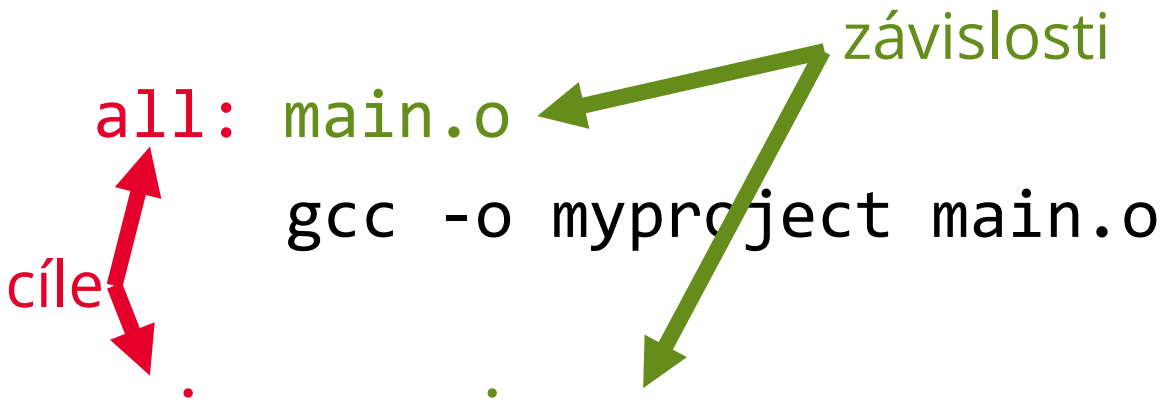
Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile

all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```



Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile

all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```



Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile

all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile

all: main.o
     gcc -o myproject main.o

main.o: main.c
     gcc -c main.c
```

Obecně:

```
cíl: závislost1 závislost2 …
     příkaz1
     příkaz2
```

Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile

all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

závislosti

cíle

příkazy

Tady musí být tabulátor!

Obecně:

```
cíl: závislost1 závislost2 …
	příkaz1
	příkaz2
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky `all`)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

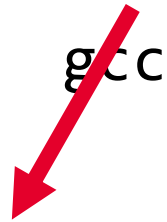
1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```



Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Co se stane po spuštění `make`?

1. Pokud není zadán cíl, najdi první (typicky all)
2. Aktualizuj jeho závislosti
3. Pokud se alespoň jedna závislost změnila, vykonej uvedené příkazy

```
all: main.o
	gcc -o myproject main.o

main.o: main.c
	gcc -c main.c
```

Každý řádek příkazů se spouští v samostatném shellu

- případné komentáře make neřeší – celý řádek předá shellu
- změna adresáře, proměnných apod. se nepřenese do dalšího příkazu
  - je nutné je zřetězit

Smaže soubory *.a v aktuální složce:

```
clean-libs:
    cd libs
    rm -f *.a
```

Smaže soubory *.a v podsložce libs:

```
clean-libs:
    cd libs && rm -f *.a
```

Trochu složitější (ale stále jednoduchý) Makefile

```
all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

Trochu složitější (ale stále jednoduchý) Makefile

```
all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

Trochu složitější (ale stále jednoduchý) Makefile

```
all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

Trochu složitější (ale stále jednoduchý) Makefile

```
all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

PROMĚNNÉ
UNIVERZÁLNÍ A IMPLICITNÍ PRAVIDLA

```
CC = gcc
CFLAGS = -Wall --std=c11
CXX = g++
CXXFLAGS = -Wall --std=c++11
LDLIBS = -lzip -lz

EXECUTABLE = hello
OBJS = main.o util.o log.o

# konkatenace
CFLAGS += -O3
```

Automatické proměnné	
$@	Cíl
$<	První závislost
$?	Všechny závislosti novější než cíl
$^	Všechny závislosti bez duplicit
$+	Všechny závislosti vč. duplicit
$(@D) $(<D) $(^D) $(+D)	Jména složek
$(@F) $(<F) $(^F) $(+F)	Jména souborů

Proměnná se dá použít prakticky kdekoliv:

```
all: $(EXECUTABLE)
$(EXECUTABLE): $(OBJS)
	$(CC) $(CFLAGS) -o $@ $^ $(LDLIBS)
```

Obsah proměnné se vyhodnocuje až při použití:

```
CFLAGS = $(include_dirs)
include_dirs = -Ifoo
```

Ale pozor na rekurzi:

```
CFLAGS = $(CFLAGS) -Ibar
```

Operátor `:=` vyhodnotí obsah proměnné ihned:

```
CFLAGS := $(CFLAGS) -Ibar
```

https://www.gnu.org/software/make/manual/html_node/Flavors.html

Použití proměnných

```
all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

Použití proměnných

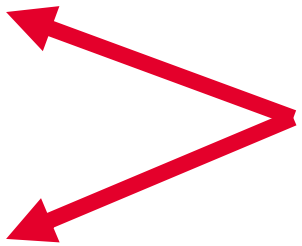
```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

Použití proměnných

```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: $(EXECUTABLE)

$(EXECUTABLE): main.o util.o
	$(CC) $(CFLAGS) -o $@ $^

main.o: main.c
	$(CC) $(CFLAGS) -c $^

util.o: util.c
	$(CC) $(CFLAGS) -c $^

clean:
	rm -f $(EXECUTABLE) *.o
```

Použití proměnných

```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: $(EXECUTABLE)

$(EXECUTABLE): main.o util.o
	$(CC) $(CFLAGS) -o $@ $^

main.o: main.c
	$(CC) $(CFLAGS) -c $^

util.o: util.c
	$(CC) $(CFLAGS) -c $^

clean:
	rm -f $(EXECUTABLE) *.o
```



tato pravidla jsou de facto stejná

Lze psát univerzální pravidla pomocí znaku %

```
%.o: %.c
	$(CC) $(CFLAGS) -c $<
```

Make má v sobě zabudovanou sadu implicitních pravidel:

- `%.c → %.o` `$(CC) $(CPPFLAGS) $(CFLAGS) -c`
- `%.cc, %.cpp, %.C → *.o` `$(CXX) $(CPPFLAGS) $(CXXFLAGS) -c`
- `%.o → %` `$(CC) $(LDFLAGS) n.o $(LOADLIBES) $(LDLIBS)`

Seznam:
https://www.gnu.org/software/make/manual/html_node/Catalogue-of-Rules.html

Použití univerzálních pravidel

```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: $(EXECUTABLE)

$(EXECUTABLE): main.o util.o
	$(CC) $(CFLAGS) -o $@ $^

main.o: main.c
	$(CC) $(CFLAGS) -c $^

util.o: util.c
	$(CC) $(CFLAGS) -c $^

clean:
	rm -f $(EXECUTABLE) *.o
```

Použití univerzálních pravidel

```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: $(EXECUTABLE)

$(EXECUTABLE): main.o util.o
	$(CC) $(CFLAGS) -o $@ $^

%.o: %.c
	$(CC) $(CFLAGS) -c $^



clean:
	rm -f $(EXECUTABLE) *.o
```

Použití implicitních pravidel

```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: $(EXECUTABLE)

$(EXECUTABLE): main.o util.o



# s využitím zabudovaných pravidel není třeba psát skoro nic



clean:
	rm -f $(EXECUTABLE) *.o
```

Srovnání původního a zkráceného Makefile

```
all: hello

hello: main.o util.o
	gcc -g -Wall --std=c11 -o hello main.o util.o

main.o: main.c
	gcc -g -Wall --std=c11 -c main.c

util.o: util.c
	gcc -g -Wall --std=c11 -c util.c

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

```
EXECUTABLE = hello
CC = gcc
CFLAGS = -g -Wall --std=c11

all: $(EXECUTABLE)

$(EXECUTABLE): main.o util.o

clean:
	rm -f $(EXECUTABLE) *.o
```

DALŠÍ VYCHYTÁVKY

Volání funkcí:

```
$(function arg1,arg2,arg3 ...)
```

Například:

```
SOURCES = $(wildcard *.c)
OBJS = $(patsubst %.c,%.o,$(SOURCES))

USERNAME := $(shell whoami)
HOSTNAME := $(shell hostname)
```

https://www.gnu.org/software/make/manual/html_node/Functions.html

Program poskytující informace o nainstalovaných knihovnách, umožňuje automaticky nastavit překladač.

Použití v Makefile:

```
CFLAGS += $(shell pkg-config --cflags libzip)
LDLIBS += $(shell pkg-config --libs libzip)
```

Do CFLAGS přidá:

```
-I/usr/lib/libzip/include
```

Do LDLIBS přidá:

```
-lzip -lz
```

Speciální cíl .PHONY umožňuje definovat, které cíle se mají provést vždy, bez ohledu na existenci souboru.

```
.PHONY: clean

clean:
	rm -f $(EXECUTABLE) *.o
```

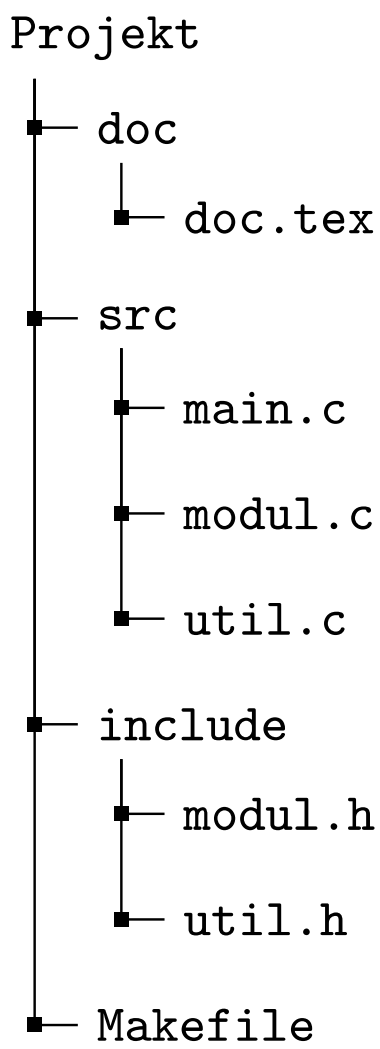
Pokud by chyběla definice `.PHONY` a existoval by soubor clean, příkaz by se nevykonal.

https://www.gnu.org/software/make/manual/html_node/Special-Targets.html

vpath

Říká, kde se mají hledat **závislosti** cílů.

```
# všechny typy souborů
VPATH = doc src include

# jen některé typy
vpath %.c src
vpath %.h include
vpath %.tex doc

modul.o: modul.c modul.h util.h
	$(CC) $(CFLAGS) -c $<
```



http://make.mad-scientist.net/papers/how-not-to-use-vpath/

gcc umí vygenerovat cíle a závislosti pro Makefile

```
gcc -MM *.c
```

Výstup je například:

```
dtsp.o: dtsp.c dtsp.h random.h config.h load.h
load.o: load.c config.h load.h dtsp.h random.h
main.o: main.c dtsp.h random.h config.h load.h
```

Google: *make auto dependency gcc mm*

Příklad Makefile

```
SERVER = merlin.fit.vutbr.cz    # nebo xwigla00@merlin.fit.vutbr.cz
SERVER_DIR = ~/KRY/proj1
ZIP_FILE = xwigla00.zip
.PHONY: all pack run clean upload

all: $(EXECUTABLE)
pack: xwigla00.zip
run: $(EXECUTABLE)
	./$(EXECUTABLE) $(CMDLINE)
$(EXECUTABLE): $(OBJS)
	$(CC) $(CFLAGS) -o $@ $^
clean:
	rm -f $(EXECUTABLE) *.o xwigla00.zip
xwigla00.zip:
	zip -j $(ZIP_FILE) *.c *.h Makefile doc/doc.pdf
upload: xwigla00.zip
	scp $(ZIP_FILE) $(SERVER):$(SERVER_DIR)
	ssh $(SERVER) "cd $(SERVER_DIR) && unzip xwigla00.zip && make"
```

Příklad Makefile – další užitečné cíle

```
valgrind: $(EXECUTABLE)
	valgrind ./$(EXECUTABLE) $(CMDLINE)

gdb: $(EXECUTABLE)
	gdb -ex ./$(EXECUTABLE) --args $(CMDLINE)

leaks: $(EXECUTABLE)
	valgrind --track-origins=yes --leak-check=full \
	--show-reachable=yes ./$(EXECUTABLE) $(CMDLINE)
```

Odkazy

http://www.fit.vutbr.cz/~martinek/clang/make.html

https://www.gnu.org/software/make/manual/html_node/

http://make.mad-scientist.net/papers/rules-of-makefiles/

https://www.cmcrossroads.com/article/basics-vpath-and-vpath

http://make.mad-scientist.net/papers/how-not-to-use-vpath/

iwiglasz@fit.vutbr.cz