

Sestavení programů, Make

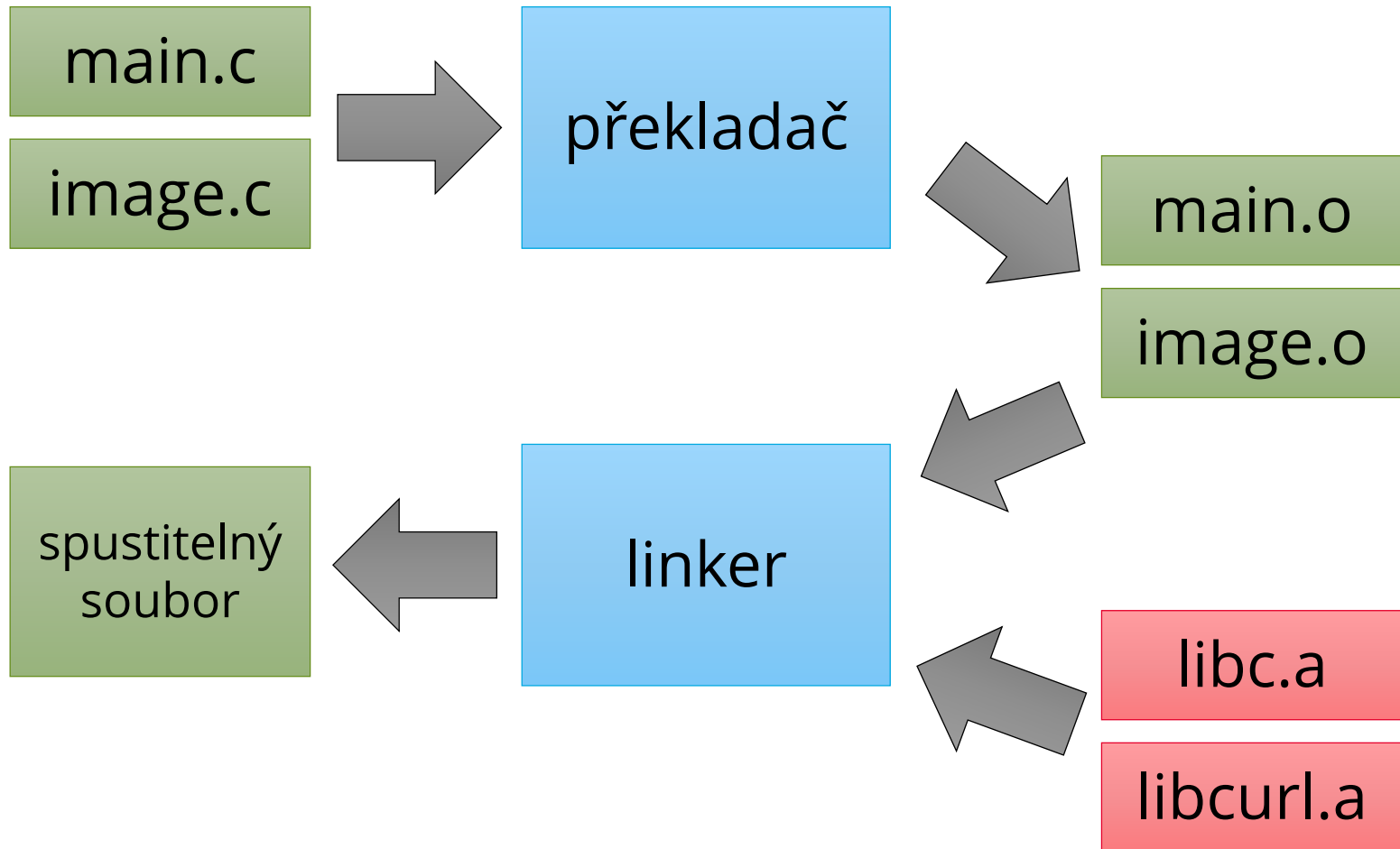
Michal Wiglasz

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



14. 3. 2017

Praktické aspekty vývoje software (IVS)



Proces sestavení programu je vhodné zautomatizovat:

- GNU Make
- Ant
- Maven
- Grunt
- Gulp
- a mnoho dalších...

- Automatizace překladu (nejen) programů
- Nezávislý na jazyku
- Multiplatformní
- Detekuje změny a přeloží pouze změněné části

Makefile se skládá z:

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

Příklady použití:

`make [cíl1] [cíl2]`

- vykoná `cíl1` a `cíl2`

`make -B [cíl]`

- vykoná vše, tj. přeloží i nezměněné soubory

`make -n [cíl]`

- *dry run*: pouze vypíše, co by se dělo

`make -j [jobs] [cíl]`

- spouští příkazy paralelně

`make -p [cíl]`

- navíc vypíše proměnné, pravidla...

`make -f rules.mk [cíl]`

- použije pravidla z jiného souboru než Makefile

Obsahuje definice cílů, předpokladů 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
```

Diagram illustrating dependencies and targets:

- cíle** (targets) points to `all:` and `main.o:`.
- závislosti** (dependencies) points to `main.o` in the first rule and `main.c` in the second rule.

Obsahuje definice cílů, závislostí a příkazů

```
# MyProject Makefile
```

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

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

The diagram illustrates the components of a Makefile rule. Red arrows labeled 'cíle' (targets) point to the target names 'all:' and 'main.o:'. Green arrows labeled 'závislosti' (dependencies) point to the dependency names 'main.o' and 'main.c'. Blue arrows labeled 'příkazy' (commands) point to the command lines 'gcc -o myproject main.o' and '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 mai
```

Diagram annotations:

- Red arrows labeled "cíle" point to "all:" and "main.o:".
- Green arrows labeled "závislosti" point to "main.o" in the first rule and "main.c" in the second rule.
- A blue arrow labeled "příkazy" points to the command "gcc -o myproject main.o".

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
```

Diagram annotations:

- Red arrows labeled "cíle" point to `all:` and `main.o:`.
- Green arrows labeled "závislosti" point from `main.o` to `main.c` and from `all:` to `main.o`.
- Blue arrows labeled "příkazy" point to the command lines: `gcc -o myproject main.o` and `gcc -c main.c`.

Obecně:

```
cíl: závislost1 závislost2 ...
```

```
    příkaz1
```

```
    příkaz2
```

Tabulátor!

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

- změna adresáře či proměnných se nepřenesse 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
```

```
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
```

```
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
```

```
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
```



```
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
```

```
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
```

```
CFLAGS += -O3
```

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

Automatické proměnné

<code>\$@</code>	Cíl
<code>\$<</code>	První závislost
<code>\$?</code>	Všechny závislosti novější než cíl
<code>\$^</code>	Všechny závislosti bez duplicit
<code>\$+</code>	Všechny závislosti vč. duplicit
<code>\$(@D)</code>	Jen adresář
<code>\$(@F)</code>	Jen jméno souboru

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

Lze psát univerzální pravidla pomocí znaku %

```
%.o: %.c
```

```
$(CC) $(CFLAGS) -c $<
```

Navíc pravidla pro některé typy souborů má make zabudovaná

Seznam:

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

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.

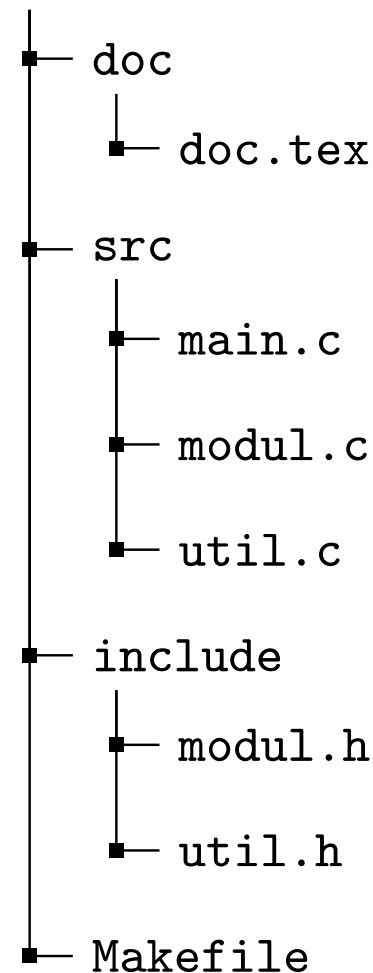
Říká, kde se mají hledat závislosti cílů.

```
# 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 $<
```

Projekt



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
```

```
SERVER=merlin.fit.vutbr.cz
```

```
SERVER_DIR=~ /KRY/proj1
```

```
.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 xwigla00.zip *.c *.h Makefile doc/doc.pdf
```

```
upload: xwigla00.zip
```

```
scp xwigla00.zip $(SERVER):$(SERVER_DIR)
```

```
ssh $(SERVER) "cd $(SERVER_DIR) && unzip xwigla00.zip && make"
```

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)
```

<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