Make

Eric Elfving

Institutionen för datavetenskap (IDA)

GNU Make

- Används vanligen för att skapa körbara binärer från källkod men går att använda till mycket mer
- Då UNIX/Linux bygger på öppen källkod får man ofta källkoden för de program man vill installera
- Detta för att få ett program väl anpassat för just ditt system samt slippa skicka stora binärfiler
- I programkodens root-mapp ingår ofta en körbar fil vid namn ”configure”
- Detta script går igenom systemuppsättningen och skapar en ”Makefile” utefter just ditt system

Programinstallation

- Om configure går bra kan man sedan kompilera programmet med ”make” och därefter köra ”make install” för att installera programmet i systemkatalogerna (kräver ofta root-access)

```
tar –xf emacs.21.4a.tar.gz
cd emacs.21.4a
./configure
make
sudo make install
```

Makefile

- En Makefile består av ett antal regler för hur mål ska utföras

```
Mål1: [beroende, …]
    kommando1
    kommando2
    …
    kommandoN

Mål2: [beroende, …]
    kommando
```

- make kontrollerar om några ändringar gjorts i de filer (eller mål) det valda målet beror på
  - Om inga ändringar skett: Klart!
  - Annars: kontrollera om något beroende måste göras om och ordna det och utför därefter kommandona som hör till detta mål
- Varje kommando måste inledas med ett tab-tecken (inget annat)

En enkel make-fil

```
program: program.cc List.o
	g++ -o program program.cc List.o

List.o: List.cc List.h
	g++ -c List.cc
```

- `make` (utan parametrar) kör det första målet i filen `Makefile`
- `List.o` är ett beroende till `program` och utförs därför först
  - `List.cc` kompileras till en objektfil
- Objektfilen används för att kompilera (och länka) `program`

Varje kommando körs i ett eget skal

```
Test:
        cd Mapp
        ls
```

- Kommer skriva ut innehållet i den mapp make körs från
- Lösning:

```
Test:
        cd Mapp; \
        ls
```

- \ gör så att nästa rad ingår i samma kommando

Makron (variabler)

- Vi kan om vi vill införa variabler för att hålla koll på t.ex. flaggor

```
GXX = gccfilter -c -a g++
CCFLAGS += -std=c++11 -pedantic -Wall -Wextra
LDFLAGS += -L/sw/gcc-${GCC4_V}/lib -static-libstdc++
COMPILE = $(GXX) $(CCFLAGS) -c
LINK = $(GXX) $(LDFLAGS)

program: program.cc List.o
	$(COMPILE) program.cc
	$(LINK) program.o List.o -o program

List.o: List.cc List.h
	$(COMPILE) List.cc
```

Vanliga mål

- Clean och zap används för att ta bort ”skräpfiler” och binärer

```
clean:
        @\rm -f *.o *~

zap: clean
        @\rm -f program
```

- Vanligtvis skrivs kommandot ut i skalet, @ tar bort utskriften

Automatiska mål

```
OBJECTS= List.o Word.o program.o
BINARY = program

$(BINARY): $(OBJECTS)
	g++ $(OBJECTS) -o $@

%o: %cc
	g++ -c $< -o $@
```

- `$@` = målets namn
- `$<` = målets beroenden

Läs mer

- http://www.gnu.org/software/make/manual/make.html
- http://en.wikipedia.org/wiki/Make_(software)

Linköpings universitet

expanding reality

www.liu.se