Mittuniversitetet
MID SWEDEN UNIVERSITY

DT028G Introduktion till programmering

Laboration:
Vim, GNU make och Hello World!

Daniel Bosk*

vimake.tex 778 2013-03-19 16:05:27Z danbos

Innehåll

1 Introduktion

För att kunna redigera din källkod krävs att du har tillgång till och kan använda en bra editor. När du sedan är klar med kodskrivandet underlättar det att ha ett byggskript för att kompilera din kod till exekverbar kod. Detta underlättar avsevärt när din kodbas växer, och det är därför bra att börja i tid.

2 Syfte

Syftet med laborationen är att du ska bekanta dig med en bra editor och att du ska lära dig att skriva enkla byggskript för att hantera dina projekt.

3 Läsanvisningar

För att genomföra laborationen behöver du orientera dig i språket C++, du bör alltså ha läst det första kapitlet i kursboken [1].

Utöver detta ska du bekanta dig med GNU make(1). Du börjar med detta genom att läsa de inledande delarna av dokumentationen [2, avsnitt 1.1 till och med 2.2].

Texteditorn vim(1) lär du dig genom att köra vimtutor(1), mer om detta under avsnitt 4.

Som referens för de olika kommandona har du även manualsidorna för de olika kommandona; make(1), vim(1) och vimtutor(1). Du öppnar dem genom kommandoraderna

```
1 $ man 1 make
2 $ man 1 vim
```

som du skriver direkt i terminalen.

4 Uppgift

Denna uppgift är uppdelad i två delar. Den första handlar om att lära sig använda en riktig editor, nämligen vim(1). Den andra delen handlar om att lära sig använda make(1) för att skapa byggskript som automatiskt kompilerar källkoden till objektkod och länkar dessa till en exekverbar fil.

4.1 Vim

Det första du ska göra är att genomföra hela vimtutor(1). Det är en praktisk tutorial för att snabbt komma igång med vim(1). Du startar den genom att köra följande kommandorad:

```
1 $ vimtutor
2 $
```

4.2 GNU make

Ett byggskript för GNU make(1) är en vanlig textfil, vanligtvis döpt till *Makefile*[1]. Filen innehåller vad man kallar för mål[2]. Målet är en fil som make(1) kommer att försöka skapa. Varje mål har vad man kallar för förutsättningar eller beroenden[3] som behövs för att skapa målet. Därefter har målet ett recept[4] för att skapa målet utifrån beroendena. Receptet består av skalkod, det vill säga vanliga kommandon som kan köras i terminalen. Strukturellt ser det ut som följer.

[1] Detta underlättar eftersom att make(1) automatiskt tittar efter en fil döpt till *Makefile*, notera att namnet ska inledas med versal.

[2] *Eng.* target.

[3] *Eng.* prerequisites, depends.

[4] *Eng.* recipe.

```
target: depend1 depend2
  shell commands to produce target from depends
  shell commands ...
```

Notera att skalkommandona måste vara indenterade med *ett* tabbtecken.

make(1) kommer att undersöka om målet finns, om det inte finns ska det skapas. Om målet finns och är äldre än någon av filerna som det beror på kommer det att skapas på nytt. Om målet däremot finns och är nyare än alla beroenden kommer make(1) att säga att målet är färskt och inte behöver uppdateras. På detta sätt behöver man inte bygga om ett helt projekt bara för att en källfil ändrats, det räcker då att skapa ny objektkod för den filen och sedan länka alla objektfiler igen.

För att skapa den exekverbara filen `program` från källfilen `main.cpp` kan vi ha följande kod i filen `Makefile`:

```
program: main.cpp
  g++ -o program main.cpp
```

Därefter för att bygga vårt program och testköra skriver vi och får tillbaka följande rader i terminalen:

```
$ make
g++ -o program main.cpp
$ ./program
Hello World!
$ make
make: 'program' is up to date.
$
```

Notera att när vi försöker bygga programmet igen säger make(1) att `program` inte behöver byggas igen.

Att make(1) först tittar om målet finns som fil kan utnyttjas till andra saker. Exemeplvis om man har ett mål vars recept aldrig skapar filen med målets namn kommer receptet att köras varje gång. Detta gör att man kan skapa mål för att genomföra testning. För att få make(1) att köra ett givet mål anges målets namn som argument till make(1), exempelvis:

```
$ make test
[kör receptet för test]
$
```

Om inget mål anges som argument används det översta målet i `Makefile`.

Det du ska göra är att skriva ett ”Hello World!”-program och skriva en tillhörande `Makefile` med ett mål för programmet och ett mål med namnet *test* för att testköra det.

5 Examination

Din lösning ska redovisas muntligen för en lärare vid något av kursens redovisningstillfällen. När du redovisat och fått godkänt laddar du upp din källkod (och byggskript) till inlämningslådan i lärplattformen.

Referenser

[1] Deitel, Paul J. och Deitel, Harvey M. *C++ : How To Program.* Pearson/Prentice Hall, Harlow, 8:e internationella utgåvan, 2012.

[2] *GNU 'make'.* GNU. URL `http://www.gnu.org/software/make/manual/make.html`.