# Kom-igång-hjälp med Linux

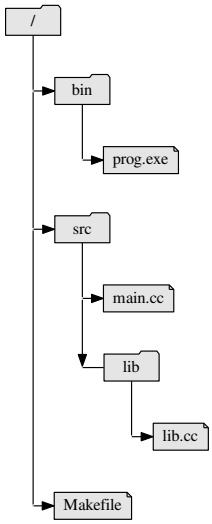
Erik Bergenholtz

26 mars 2017

## 1 Grundläggande kommandon för Linux kommandotolk

Här finns tabeller över ett par grundläggande kommandon för Linux terminal. Att kunna använda dessa kommer underlätta arbetet med Linux. Den text i tabellerna som är *kursiv* markerar att den ska ersättas med något. T.ex. `cd` ***`MAPP`*** innebär att kommandot `cd` ska följas av ett namn på en mapp, t.ex. `cd src`. Eftersom väldigt många kommandon tar en sökväg som argument, och alla sökvägar kan vara antingen relativa eller absoluta, förklaras inledningsvis hur relativa och absoluta sökvägar fungerar.

### 1.0.1 Sökvägar




En sökväg är den sekvens av mappar som krävs för att nå en viss fil eller mapp. Det finns två typer av sökvägar: relativa och absoluta.

I expemplen nedan används ett filsystem med strukturen till vänster.

Absolut sökvägar börjar ifrån filsystemets rot och listar sedan alla mappar ner till filen eller mappen man letar efter. Exempelvis är den absoluta sökvägen till filen *main.cc* `/src/main.cc`.

En relativ sökväg börjar ifrån mappen man står i just nu. Undermappar till den nuvarande mappen kan skrivas med bara namn, medan om man vill upp en nivå skriver man "`..`". Detta innebär att om man står i mappen `/src/` blir den relativa sökvägen till filen "*lib.cc*" `lib/lib.cc`, medan den relativa sökvägen till "*prog.out*" blir `../bin/prog.exe`.

## 1.1 Navigering i filsystemet

| Kommando | Förklaring |
|---|---|
| `pwd` | Skriver ut absolut sökväg till nuvarande mapp |
| `cd` *MAPP* | Gå in i mappen *MAPP* |
| `ls` | Listar innehållet i nuvarande mapp |
| `ls` *MAPP* | Lista innehållet i *MAPP* |

## 1.2 Filhantering

| Kommando | Förklaring |
|---|---|
| `file` *FIL* | Skriver ut filtypen för *FIL* |
| `touch` *FIL* | Skapar filen *FIL* |
| `rm` *FIL* | Tar bort filen *FIL*[1] |
| `cp` *FIL1 FIL2* | Kopierar *FIL1* till *FIL2*.[2] |
| `mv` *FIL1 FIL2* | Flyttar *FIL1* till *FIL2*.[2] |
| `cat` *FIL* | Skriver ut innehållet i *FIL* |
| `head` *FIL* | Skriver ut de första 10 raderna i *FIL* |
| `tail` *FIL* | Skriver ut de sista 10 raderna i *FIL* |
| `vim` *FIL* | Öppnar *FIL* för redigering i vim' |
| `nano` *FIL* | Öppnar *FIL* för redigering i nano |

## 1.3 Mapphantering

| Kommando | Förklaring |
|---|---|
| `mkdir` *MAPP* | Skapar mappen *MAPP* |
| `rmdir` *MAPP* | Tar bort mappen *MAPP*. Detta kräver att mappen är tom |
| `rm -r` *MAPP* | Tar bort mappen *MAPP* och alla dess filer och undermappar.[1] |
| `cp -r` *MAPP1 MAPP2* | Kopierar alla filer och mappar i *MAPP1* till *MAPP2*[2] |
| `mv` *MAPP1 MAPP2* | Flyttar *MAPP1* till *MAPP2*[2] |

## 1.4 Kompilering

| Kommando | Förklaring |
|---|---|
| `gcc` *FIL* | Kompilerar C-filen *FIL* till den körbara filen *a.out* |
| `g++` *FIL* | Kompilerar C++-filen *FIL* till den körbara filen *a.out* |
| `make` | Kör bygginstruktioner för att kompilera ett projekt. Kräver att instruktionerna är definierade i en fil som heter *Makefile* |

[1] Notera att detta *inte* innebär att den lägger sig i en papperskorg där man kan återskapa den, utan att den helt plockas bort ifrån filsystemet

[2] Notera att du inte blir varnad om *FIL2* redan finns, utan den skrivs över

## 2 Koppla upp sig till Raspberry Pi

Labbdatorerna i labbsalen har var sin Raspberry Pi uppkopplad till sig. För att arbeta med dem används lämpligen ett program som heter `ssh`. För att köra `ssh` öppnar du terminalen och skriver följande:

```
ssh <user>@<IP>
```

Exempelvis skulle detta kunna se ut såhär:

```
ssh pi@10.0.0.2
```

`ssh` kommer då försöka logga in på datorn med användarnamnet *pi*. Om användaren har ett lösenord kommer du bli ombedd att mata in det. Per default är användarnamnet *pi* och lösenordet *raspberry* på alla Raspberry Pi's.

För att hitta IP-addressen till din Raspberry Pi i labbsalen kan du skriva

```
arp -a | grep -i "<MAC-ADDRESS>"
```

där `<MAC-ADDRESS>` byts ut mot det som står på Raspberry Pi-stationen.

På It's Learning finns det ett program som heter *setup.sh*. Detta kan köras för att underlätta inloggningen på Raspberry Pi. Ladda ner scriptet och kör

```
chmod +x setup.sh ; ./setup.sh <IP> [NEW|OLD]
```

för att köra programmet, där `[NEW|OLD]` byts ut mot `NEW` eller `OLD` baserat på om du redan kört scriptet mot det SD-kortet elle inte. Du kommer behöva skriva in lösenord ett par gånger, men bör inte behöva det fler gånger när scriptet kört klart. Efter det räcker det med att du skriver

```
ssh raspberrypi
```

för att logga in.

Väl inloggad får du tillgång till en terminal på din Raspberry Pi. Den skiljer sig inte ifrån den "vanliga" terminalen på något sätt, så referensen ovan går att använda på Raspberry Pi också.

# 3 Textredigering

När du kodar på Raspberry Pi:n via `ssh` kommer du inte ha ett skrivbord att jobba med. Det innebär att du måste använda terminalbaserade texteditorer. Det finns två alternativ förinstallerade: `vim` och `nano`. Båda två går bra att använda, men de är väldigt olika. `vim` är mycket mer kraftfullt än `nano`, men det är också svårare att lära sig. Här kommer en väldigt kort introduktion till de båda.

## 3.1 `vim`

För att öppna en fil i vim skriver du följande i terminalen:

```
vim [FILNAMN]
```

Exempelvis:

```
vim main.c
```

`vim` startar då i s.k. "normal mode". Detta innebär att det du skriver på tangentbordet tolkas som kommandon. För att börja skriva text måste du gå in i "insert mode", vilket låter dig mata in text till filen. Nedan ser du ett par väldigt grundläggande kommandon som är essentiella för att kunna arbeta med `vim`. Vill du göra något mer komplext är Google svaret.

| Kommando | Mode | Förklaring |
|---|---|---|
| i | Normal | Går in i insert mode |
| I | Normal | Hoppar till början av raden, går in i insert mode |
| a | Normal | Går in i insert mode och stegar fram markören ett steg |
| A | Normal | Hoppar till slutet av rader, går in i insert mode |
| o | Normal | Går in i insert mode, lägger till en ny rad under nuvarande |
| O | Normal | Går in i insert mode, lägger till en ny rad ovarnför nuvarande |
| u | Normal | Ångra ändringar |
| CTRL+r | Normal | Gör om ändringar |
| dd | Normal | Klipper ut nuvarande rad och lägger i buffern |
| yy | Normal | Kopierar nuvarande rad och lägger i buffern |
| p | Normal | Klistrar in bufferinnehållet efter markören |
| P | Normal | Klistrar in bufferinnehållet innan markören |
| / | Normal | Söker i filen |
| n | Normal | Stega till nästa träff |
| N | Normal | Stega till förra träffen |
| :w | Normal | Spara |
| :q | Normal | Stäng av `vim` |
| :wq | Normal | Spara och stäng |
| :q! | Normal | Stäng `vim` utan att spara |
| ESC | Insert | Går till normal mode |

## 3.2 `nano`

För att öppna en fil med nano skriver du följande i terminalen:

```
nano [FILNAMN]
```

Exempelvis:

```
nano main.c
```

`nano` öppnar då filen för redigering, inget mer behöver göras för att kunna skriva i filen. Nedan ser du ett par viktiga och användbara kommandon för `nano`:

| **Kommando** | **Förklaring** |
|---|---|
| CTRL+o | Spara fil |
| CTRL+x | Stäng `nano` |
| CTRL+w | Sök i fil |
| CTRL+k | Klipp ut rad |
| CTRL+u | Klistra in rad |
| CTRL+g | Visa hjälpfönster |

# 4 GDB - Gnu Debugger

## 4.1 Kompilera för debugging

Om man ska använda debuggern gdb för att följa programexekveringen och för att felsöka behövs ett särskilt kompileringsdirektiv, -g, för att generera symbolinformation till debuggern.

Anta att de programdelar som ska länkas ihop inför körning heter Mprov.s och bibliotek.s. Då genereras en körbar fil (kompileras och länkas) för debugging som heter test med:

```
gcc -g -o test Mprov.s bibliotek.s
```

## 4.2 Starta och stoppa körning

När programmet är kompilerat och länkat kan debuggern aktiveras med:

```
gdb test
```

För att köra programmet skriver man run (r kort), men att göra bara det är inte mycket mening eftersom programmet nu uppför sig exakt som det gör direkt vid Linux-prompten. Vitsen med att använda debuggern är att man nu kan sätta brytpunkter, så att programexekveringen stannar på väl valda ställen. Det är lätt att sätta brytpunkter med kommandot break (b kort) vid etiketter (eng. labels) i koden. Om det finns en etikett i koden där funktionen inimage börjar, kan en brytpunkt sättas där med kommandot:

```
b inimage
```

Om en sådan brytpunkt skapats kan programmet köras med kommandot run. Nu stannar programmet vid den angivna etiketten, och man kan titta på vad som finns i register och minne om man vill.

Om programmet kört till en brytpunkt och man vill titta närmare på vad som händer steg för steg i programmet kan stegvis exekvering användas. Kommandot step (s kort) kör en instruktion och om en siffra anges efter kommandot körs så många instruktioner som siffran anger innan programmet avbryts igen. Det finns en variant på kommandot som heter next (n kort). Skillnaden är att next inte går in i funktioner instruktionsvis, utan funktionen ses som en instruktion när programmet stegas igenom.

## 4.3 Titta på registerinnehåll

Om man vill titta på innehållet i cpu-registren kan kommandot info användas. Registerinnehållen (utom stackpekare m fl. som bara innehåller adresser) presenteras då både på hexadecimal och decimal form:

```
info registers
```

Vill man bara titta på innehållet i ett specifikt register kan man använda kommandot print. Där kan man också ange en formatkod för vilket format man vill se innehållet på (decimal är default). Koden är d för decimal form, x för hexadecimal, t för binär form och c för ASCII-tecken. Exempel för att visa innehållet i register rax på hexadecimal form:

```
print/x $rax
```

Om man frekvent är intresserad av innehållet i ett specifikt register kan man använda kommandot display. Det register man angivit kommer då att visa sitt innehåll varje gång programmet avbryts. Exempel:

```
display $rax
```

Man kan upprepa detta för flera register, vilka då hamnar i en numrerad lista. Vill man sluta visa registret skriver man bara undisplay. Om man vill ta bort ett specifikt register anger man dess listnummer. Utan argument kan man ta bort alla samtidigt.

## 4.4 Titta på minnesinnehåll

Minnesinnehåll kan undersökas med kommandot x. Här kan man också ange hur många byte man vill se från och med en viss adress och på vilket format. Anta att en etikett inbuff definierats för en buffert i minnet. Om man vill titta på 12 byte framåt i minnet därifrån som ASCII-tecken bytevis kan ett kommando för det skrivas som:

```
x/12cb &inbuff
```

12 är hur många enheter i minnet som ska granskas c betyder att innehållet ska tolkas som ASCII-tecken (se formatkoderna under rubriken ”Titta på registerinnehåll”) och b betyder att de 12 enheterna är av typen byte. Andra valmöjligheter där är h (16-bits ord) och w (32-bits ord).

`&inbuff` betyder helt enkelt adressen till etiketten inbuff.

## 4.5 Mer hjälp

Om du behöver kolla upp hur något fungerar som inte finns skrivet i detta dokumentet kan man använda kommandot `help`. För att få en lista över alla kommandon av en viss typ, kör

```
help kommandotyp
```

För att få hjälp med ett specifikt kommando, använd:

```
help kommando
```

Vill du ha en mer generell manual till GBD kan man i Linux-terminalen (alltså inte i gdb's promtp) skriva `man gdb`.