Bilaga A

UNIX-terminalen

I detta appendix ska några praktiska detaljer om hur man använder UNIX förklaras. UNIX är ett operativsystem som utvecklades på Bell Labs i början av 1970-talet. Det är det mest framgångsrika operativsystem för professionell datoranvändning och finns t.ex som MacOS X och Linux men kärnorna i både Android och iOS är baserade på UNIX. Det viktigaste operativsystemet som inte är baserat på UNIX, Microsoft Windows, stöder nu numera många Linux appar genom sitt Ubuntu subsystem. Det är dock ofta betydligt enklare att använda UNIX i en virtuell maskin i Windows, se t.ex. `virtualbox.org` och `debian.org`. Ett skäl till det är att Windows har begränsat stöd för att öppna nya fönster genom Ubuntu subsystem.

A.1 Starta en terminal

För att börja behöver du ett fönster som är en **terminal**. Vi antar att du använder Linux eller MacOS X när vi förklarar hur du ska starta en terminal. Bortsett från detta första steg ska resten, i princip, fungera även på Windows genom dess Ubuntu subsystem. Leta upp platsen där du kan söka efter kommandon och skriv `terminal`. Tryck på tangenten för ENTER eller RETURN på ditt tangentbord. Om detta inte fungerar är det bäst att fråga en kompis eller söka på nätet. När du har startat en terminal, skriv följande kommando och tryck sedan ENTER:

```
pwd
```

Du bör då få utdata i form av `/home/`*yourname* på Linux (och lite annorlunda på Mac). Terminalen är ett fönster och i det kör ett program som heter `bash` och det är detta som läser vad du skriver i terminalfönstret och utför dina kommandon. Bash är en så kallad **kommandoradstolk**, en typ av program som också kallas ett **skal** men vi kommer att använda **shell** istället vilket är vanligt även i Sverige. Bash står för **Bourne Again Shell**, och är en mer sofistikerad version av

den första kommandoradstolken, som skapades av den brittiske matematikern Stephen Bourne. Både Linux och MacOS X använder Bash. När du skrev `pwd` och tryckte ENTER, läste Bash vad du skrev och utförde sedan kommandot för att skriva ut vilken katalog du befinner dig i. Vi ska nu se vad en **katalog** är.

A.2 Filer och kataloger

Alla dokument i en dator, och även i telefoner, finns i någon katalog. I UNIX kallas ett dokument också en **fil**. Nästan allting är en fil i UNIX, inklusive kataloger, skrivare och USB minnen. Att filer är organiserade i kataloger gör det enklare att arbeta med dem. Varje användare i en UNIX dator har en **hemkatalog** vilken är roten av ett träd av kataloger som som den användaren äger. Det speciella tecknet tilde, ~, är en förkortning för din hemkatalog. En analogi till hemkataloger är rum i ett hus (och datorn är huset). I rummen kan det finnas hyllor där man kan lägga dokument. I en dator kan en katalog innehålla fler kataloger. Din hemkatalog har förmodligen katalogerna `Documents`, `Downloads` och `Desktop`. Eventuellt heter Desktop istället Skrivbord. Desktop är speciellt eftersom du kan se dess innehåll direkt på datorns bildskärm.

Från och med nu kommer vi inte att skriva ut att du ska trycka ENTER efter att du skrivit in ett kommando. Skriv nu kommandot:

```
cd Desktop
```

Bash kommer nu att använda din Desktopskatalog som sin arbetskatalog. Kontrollera detta genom attt ge kommandot `pwd` en gång till. Gör nu ett experiment och skriv kommandot:

```
ls
```

Utdata ska bli en lista över alla filer och det bör vara samma filer som du ser på bildskärmen på Desktop. Kommandot `ls` står för *list* (som i verbet). Ge nu kommandot (som utläses ”ls minus ell”):

```
ls -l
```

Denna så kallade flagga eller switch till `ls` säger åt `ls` att skriva ut mer information om varje fil, såsom t.ex. filens storlek i den femte kolumnen och när filen senast ändrades.

Kommandot `cd` som du använde tidigare kan också användas utan något argument, och betydelsen då är att byta till din hemkatalog. Gör det:

```
cd
```

Vi använder ofta `cd` som ett verb. Antag att du vill skapa en katalog för laborationerna i den här boken. För att skapa en katalog `algoritmer` ska du skriva:

```
mkdir algoritmer
```

Gå till denna nya katalog och skapa en ny katalog `labbar` och gör cd till den. Om du använder Linux och allt har blivit rätt ska du nu vara i följande katalog: `/home/`*yourname*`/algoritmer/labbar`.
Skriv nu:

```
ls -a
```

Du ska se två filnamn: ”punkt” och ”punkt punkt”, dvs. två filer med namnen `.` och `..` och dessa filer finns i varje katalog. Filer vars namn börjar med en punkt listas bara med `-a` flaggan. Filen `.` är namnet på den nuvarande katalogen och `..` är namnet på föräldrarkatalogen (dvs, den katalog som är ett steg upp). Skriv:

```
cd ..
```

Verifiera att du nu är i katalogen `/home/`*yourname*`/algoritmer`. Hur man kan ha nytta av filnamnet `.` ska vi se senare.

För att upprepa det senaste kommandot skriver man `!!` och för att upprepa det senaste kommandot som börjar med en viss sträng såsom `abc` kan man skriva `!abc` eller använda tangentbordets ”upp-pil” och leta upp det som börjar med `abc` — eller t.ex bara gå till det förra kommandot.

A.3 Kopiera filer

För att kopiera en fil använder man kommandot `cp`. Om du har en fil `graph.scala`, och vill kopiera den innan du börjar ändra i den (som backup), ska du skriva:

```
cp graph.scala annatnamn.scala
```

Eller något annat namn. Gå nu till din hemkatalog. Om du vill göra en kopia av en hel katalog, t.ex. `algoritmer`, ska du skriva:

```
cp -r algoritmer algoritmer-001
```

Flaggan `-r` står för rekusivt, dvs. alla kataloger i `algoritmer` kopieras också.

Att göra kopior som backup av dina filer är ytterst viktigt. Dels vill man undvika att man råkat göra felaktiga ändringar i en viss fil, dels få tillbaka en fil som man av misstag råkat ta bort, och dels få tillbaka filer när en hårddisk gått sönder. Om hårddisken går sönder försvinner alla filer på den och därför är det lämpligt att lägga kopior på en annan dator. I nästa paragraf ska ska vi se hur man kopierar filer mellan datorer.

A.4 Grundläggande SSH kommandon

Secure Shell eller **SSH** är ett antal kommandon som gör det möjligt att kopiera filer mellan datorer och även att arbeta på distans från en terminal på en dator, t.ex. en laptop, och logga in på en annan dator och arbeta i en terminal på den. Man kan även öppna nya fönster på den andra datorn som visas på den lokala datorn. Allt du skriver inklusive lösenord vid inloggningen krypteras. Först ska vi se hur man kopierar en katalog från din lokala dator till ett konto på en annan dator, som vi antar är `abc@login.student.lth.se`. Gå till katalogen där det du vill kopiera finns och skriv:

```
scp -r algoritmer abc@login.student.lth.se:
```

Första gången du loggar in på (eller kopierar till) en annan dator blir du tillfrågad om du litar på att det är rätt dator. Oftast är det säkert att svara `yes`. Du blir därefter tillfrågad om ditt lösenord på den andra datorn. Som sagt är det säkert att mata in det eftersom det inte skickas i klartext över nätet utan krypterat men om den andra datorn blivit hackad är givetvis inte SSH säkert. Notera att du måste skriva ett kolon efter den andra datorns namn. Detta kolon informerar `scp` att strängen före kolonet är en dator och inte en fil på den lokala datorn. Om du glömmer kolon kommer du att kopiera till din lokala dator istället.

För att kopiera en katalog från din hemkatalog på en annan dator, t.ex. `algoritmer/labbar/lab1` till din lokala dator skriver du:

```
scp -r abc@login.student.lth.se:algoritmer/labbar/lab1 .
```

Notera användningen av punkt: vi säger åt `scp` att kopiera `lab1` till katalogen vi är i (som heter `.`). Vi ser också att utöver kontonamn och dator kan vi ange en katalog som i det här fallet måste finnas i din hemkatalog på den andra datorn.

För att arbeta på distans på en annan dator kan du skriva:

```
ssh -Y abc@login.student.lth.se
```

Här ska inget kolon användas. Flaggan `-Y` är användbar om du vill öppna ett fönster på den andra dator som ska synas på din lokala dator. Oftast blir detta ganska långsamt men det kan fungera i viss utsträckning. Du kan vilja prova en modern editor såsom `atom` eller `sublime` men editorer som utvecklats för terminaler är oftast mycket snabbare speciellt om man kör dem över ett nätverk. Prova t.ex `vi`, `emacs`, eller `nano` (men åtminstone `vi` är ganska komplicerad och tar lite tid att lära sig ordentligt). Du loggar ut från den andra dator genom att skriva `exit` eller trycka CTRL-D.

För att slippa behöva mata in lösenord varje gång kan man skapa en speciell fil och kopiera den till den andra datorn. Skriv:

```
ssh-keygen -t rsa
```

Programmet kommer att fråga efter ett hemligt meddelande (passphrase) som krypteras. Om du går till din hemkatalog och sedan gör cd till `.ssh` kommer du att se två filer: `id_rsa` och `id_rsa.pub` och det är den senare som ska kopieras till den andra datorn och för det finns ett speciellt kommando:

```
ssh-copy-id abc@login.student.lth.se
```

När du nästa gång loggar in på den andra dator är det möjligt att din lokala dator frågar efter ditt hemliga meddelande och om din dator ska komma ihåg detta. Om det är din egen dator är det i princip lämpligt att svara ”yes” eller ”forever” eller något liknande. Efter dessa åtgärder ska du kunna logga in till och kopiera filer till och från den andra datorn utan att behöva ange lösenord.

Efter en stund kan SSH märka att du inte varit aktiv och därför avbryta en inloggning, vilket kan vara störande men om du lägger till följande rad i filen `~/.ssh/config`:

```
ServerAliveInterval 60
```

så slipper du det problemet.

A.5 Filnamnsexpansion

Antag att du vill göra cd till `Downloads` från din hemkatalog. Då kan du skriva `cd Dow` och trycka TAB. Om `Dow` är ett unikt prefix för filnamn i din hemkatalog kommer Bash att expandera det till `Downloads`.

En annan form av filnamnsexpansion är $*$ och `?`. Strängen $*$`.scala` expanderar till alla filnamn som slutar på `.scala`. Tecknet `?` matchar ett tecken, så `?.scala` expanderar till alla filnamn med ett tecken följt av `.scala`, som t.ex `a.scala`.

A.6 Ta bort filer

För att ta bort filer används kommandot `rm`:

```
rm filnamn
```

För att ta bort en katalog, lägg till flaggan `-r`. Många UNIX användare lär sig ”den hårda vägen” att kommandon som:

```
rm * .class
```

har ett blanktecken efter $*$ vilket betyder att alla filnamn matchas och tas bort varefter du informeras om att det inte finns någon fil som heter `.class`.

A.7 Scala och C programmering

För att kompilera en Scala fil `graph.scala` skriver du:

```
scalac graph.scala
```

För att köra `main` metoden i `object graph` skriver du:

```
scala graph
```

Det är ofta bättre att använda `fsc` istället för `scalac` eftersom det startar en server som minskar kompileringstiden något. För att istället kompilera ett C program `graph.c` skriver du:

```
cc graph.c
```

För att köra det skriver du:

```
./a.out
```

För att använda ett stort antal verktyg för C och C++ hänvisar vi till boken [21].

A.8 Enkel shell programmering

Bash är ett programspråk med variabler och satser såsom `if` och `for` vilka gör det möjligt att skriva program i Bash. Ett shellprogram kallas också ett **skript**. Ett av de mest användbara skripten vid programmering (i andra språk) är skript för att automatisera testning. Antag att du vill kompilera ett Scala program och sen köra det om kompileringen lyckades. Då kan man skriva:

```
scalac graph.scala && scala graph
```

Alla program returnerar ett heltal till Bash[1] och det finns en konvention som säger att returvärdet noll betyder att ett program lyckades utan fel och ett returvärde skiljt från noll indikerar någon form av fel. När ditt program anropar `exit(0)` eller returnerar ett tal från `main` metoden, är det detta tal som används av Bash. I normala fall ignorerar Bash returvärdet men i kommandot ovan körs `scala graph` endast om `scalac` returnerade noll.

Antag att du vill testa om utdata från ditt program är korrekt. Det kan du göra genom att ha en fil, t.ex. `correct` och spara ditt programs utdata i en fil `output`. Kommandot `diff` jämför två filer och om de är identiska returnerar det noll. Skapa först filen med korrekt utdata och skriv sedan:

```
scalac graph.scala && scala graph > output && diff output correct
```

[1] För ett vara mer exakt, alla program returnerar ett heltal till föräldrarprogrammet (normalt det som startade barnet) och Bash är ofta föräldern men vilket program som helst kan starta nya program och kolla returvärdet.

Då kommer `diff` att skriva ut skillnaden, så kortfattat som möjligt, på de två filerna. Om `diff` inte skriver ut något är de identiska. Kommandot `echo` skriver helt enkelt ut sina argument och det kan användas så här:

```
scalac graph.scala &&
scala graph > output &&
diff output correct &&
echo PASS
```

Genom att placera && på slutet av varje rad fortsätter Bash på nästa rad.

Om du lägger ovanstående kommandon i en fil, t.ex med namnet `t` kan du skriva:

```
sh t
```

Denna fil `t` är ett exempel på ett shellprogram, eller skript. Antag nu att du har flera tusen testfall som par av indata och korrekt utdata, namngivna som *testname*`.in` och *testname*`.correct`. Du kan använda en for-loop för att utföra alla tester:

```
for x in *.in
do
        echo testing $x
        y=$(basename $x .in)
        scala graph $x > output
        if diff output $y.correct
        then
                echo test failed for $x
                exit 1
        fi
done
```

Detta illustrerar vad vi har lärt oss hittills och introducerar variabler. Vi ser att `*.in` matchar alla filnamn som slutar på `.in` och att for-loopen kommer att iterera genom dessa med variabeln `x` tilldelad ett sådant namn per iteration. Det första loopen gör är att skriva ut vilken fil som testas. Som vi kan se får man värdet av en variabel genom att använda ett dollartecken. Utan dollartecknet kan inte Bash skilja på variabler och filnamn. Nästa sats ser lite konstigt ut men sparar testnamnet utan ändelsen `.in` i en annan variabel `y`. När man placerar ett kommando inne i `$()` produceras en sträng som t.ex kan sparas i en variabel. För att skapa filnamnet på den korrekta utdatan använder vi `$y.correct` Om testfallet heter `a.in` blir resultatet `a.correct`. Om nu `diff` returnerar ett värde som är skiljt från noll kommer Bash att märka det i if-satsen och avsluta testningen. Om du inte vill avsluta utan forsätta med fler testfall kan du byta till `# exit 1` för att kommentera bort den raden.

Slutligen, för att se vilka kommandon ett skript utför kan du skriva:

```
sh -x t
```

Med det avslutar denna korta introduktion till UNIX-terminalen.