Självstudiematerial 1

Detta är det första av två självstudiematerial som ni behöver använda för att komma igång med de verktyg och system som används i kursen. Materialet kräver att ni har VirtualBox installerat samt ha laddat ned och importerat maskinen NewTinyDebian i VirtualBox. Länkar till dessa finns på kurswebbsidan som ni hittar genom att googla på "HI1025 allt kursmaterial", kolla under VT 2016.

1. Om mjukvara och operativsystem

För att en dator överhuvudtaget ska kunna köra behövs instruktioner om vad maskinen ska göra. Dessa instruktioner styr varenda liten detalj i maskinens beteende och påverkar således hela vår upplevelse av handhavandet av maskinen. Instruktionerna kan bytas ut utan att vi byter ut någon maskindel i datorn. Vi brukar därför säga att instruktionerna är mjukvara, de kan lätt bytas ut. Själva de fysiska maskindelarna, skärm, tangentbord, moderkort, etc. kallas då hårdvara.

I de föregående kurserna har ni mest studerat mjukvara och att ta fram mjukvara kallas att *programmera*. Mjukvara kan också kallas "program", men vi ska göra en annan uppdelning av mjukvara, vi delar in mjukvara i två huvudkategorier:

- Användarprogram
- Systemprogram

Användarprogram är typiskt applikationer som till exempel *Microsoft Office* eller *OpenOffice*, två stora kontorsprogram. Olika användare kan använda olika användarprogram, en användare kanske gillar *Microsoft Word* och en annan kanske gillar *OpenOffice Writer*. Ett annat exempel på användarprogram kan vara *iTunes*, eller de små hjälpmedlena som ofta finns i form av kalkylatorer, ritprogram för att göra eller hantera bilder och, förstås, webbläsare typ Firefox eller Internet Explorer.

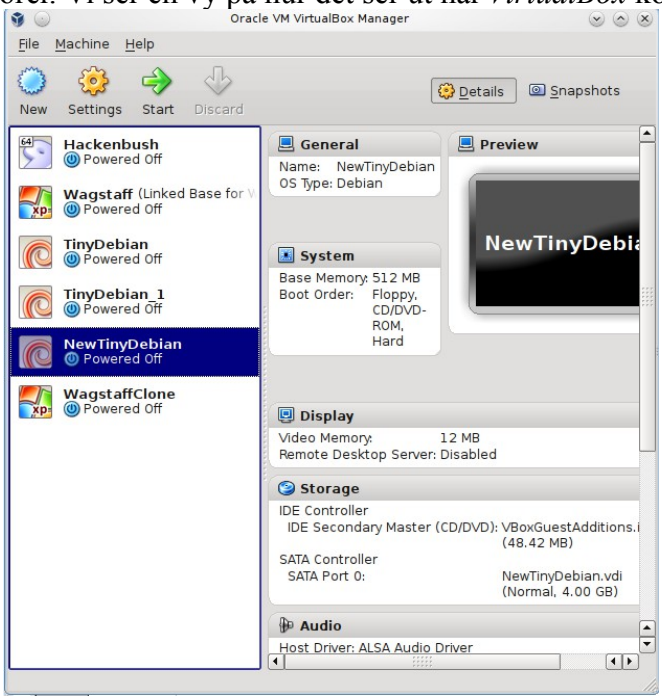
Systemprogram är då program som är nödvändiga för att datorn överhuvudtaget ska köra. Den viktigaste systemprogramvaran kallas *operativsystemet* och vi kan (generellt) inte köra en dator idag utan ett operativsystem (OS). Uppgiften hos ett OS är att administrera datorns resurser så att alla användare som kör på datorn får del av datorkraften. Utan OS kör alltså inte datorn idag. Det betyder att *alla* användare måste ha ett operativsystem, de kan ha olika användarprogram (ordbehandlare eller ritprogram eller webbläsare) men alla *måste* ha ett operativsystem.

Det mest utspridda operativsystemet är *Windows*. Det finns så många varianter så man ska snarare tala om *Windows*-familjen. Tyvärr lider *Windows* av olika växtsjukdomar. En mer pålitlig familj av operativsystem är *UNIX*-systemen och vi ska faktiskt använda ett *UNIX*-system under denna kurs och flera framöver. Det system vi ska avända heter *Debian* – ett av de bästa *UNIX*-systemen som faktiskt är helt gratis.

2. Virtualisering

För att kunna köra en dator måste vi alltså ha ett operativsystem som installeras på den datorn. Vi ska alla använda systemet *Debian* och då ska vi faktiskt skapa varsinn egen dator för detta. Vi ska inte skapa en riktig *fysisk* dator utan den dator som vi ska köra ska vara *virtuell*. Det betyder att vi använder en speciell programvara (ett användarprogram) som heter *VirtualBox* som är ett program

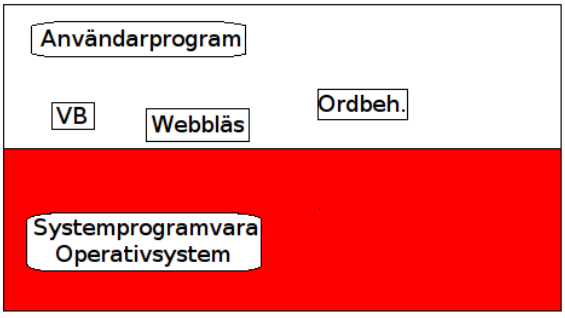
som låtsas köra flera datorer. Vi ser en vy på hur det ser ut när *VirtualBox* kör:



Det här är kontrollfönstret hörande till *VirtualBox* (VB) och det finns för närvarande 6 stycken virtuella maskiner som den håller reda på. Maskinerna har olika namn, *Hackenbush*, *Wagstaff*, *TinyDebian*, *TinyDebian_1*, *NewTinyDebian* (som är den vi ska använda) och *WagstaffClone*. Maskinerna har olika OS installerade, *Hackenbush* har ett system som heter *Gentoo Linux*, De båda *Wagstaff*-maskinerna har *Windows XP* och de maskiner som har "debian" som del av sina namn har just systemet *Debian* i sig. *Debian* är en så kallad *Linux*-distribution (*Gentoo* är en annan). Till höger om kolumnen med maskiner ser man information om den innevarande maskinens egenskaper. Vi ser till exempel att primärminnet är 512 MB, hårddisken är på 4.00 GB osv. Det finns flera saker som man ser till höger, men vi ska inte titta på allihop.

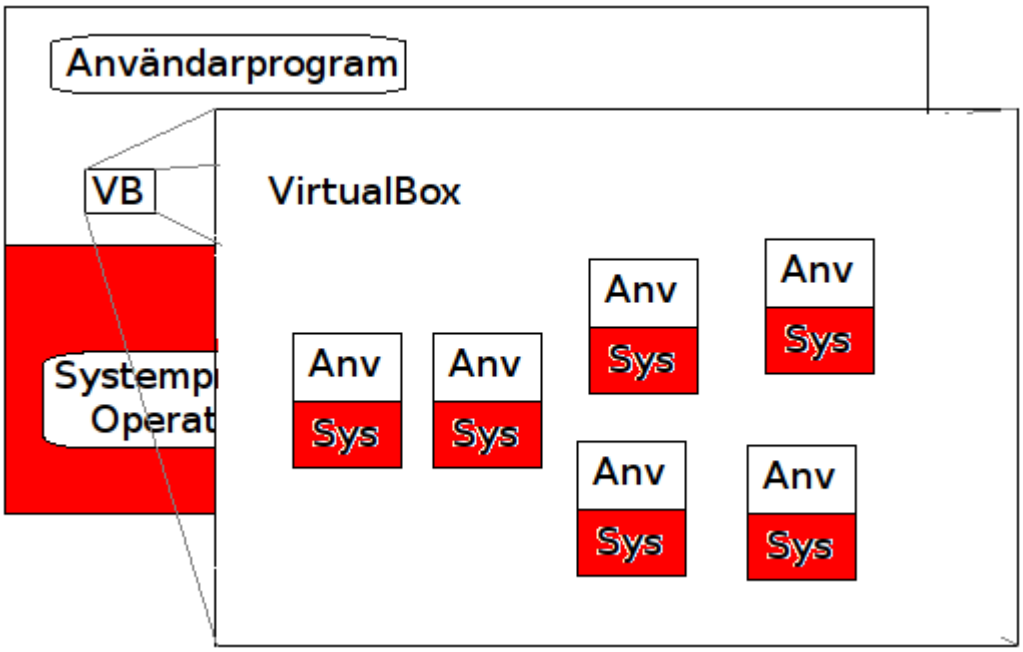
VirtualBox är en av de mest fantastiska programvaror som någonsin skrivits (dessutom gratis). Det innebär att vi på våran vanliga dator, kan köra många möjliga varianter av datorer och operativsystem – utan att skaffa flera fysiska maskiner. Vi kommer att använda oss av VB mycket i utbildningen. *VirtualBox* är en virtualiseringsprogramvara och vi ska beskriva hur det fungerar.

Då en dator kör mjukvara är dessa som sagt av två slag, användarprogram eller systemprogramvara, i skissen nedan är systemprogramvaran färgad röd för att den är så viktig, vi kan inte köra användarprogram om inte systemprogramvaran finns där. Vi ser tre stycke användarprogram, VB = *VirtualBox* samt en webbläsare och en ordbehandlare:



Det fantastiska med *VirtualBox* är att den är en emulator som gör det möjligt för oss att emulera

(låtsas) köra flera datorer samtidigt med olika operativsystem Vi kan då se VB som ett rum fullt med datorer som kör olika operativsystem samtidigt och oberoende av varandra:



Så i användarprogrammet VB (*VirtualBox*) finns alltså flera andra system körandes som var för sig innehåller systemprogramvara och användarprogram. Den dator som kör VB kallas då *värd* och de virtuella datorerna som kör inuti VB kallas då *gäster*. (Engelska: *host*/*guest*).

3. Delade mappar

Eftersom de datorer (virtuella datorer) kör helt oberoende av varandra så har de egentligen inte några inbyggda möjligheter att kommunicera med omgivningen. Det betyder att det är svårt att skicka data till dessa maskiner. Lösningen på detta är att vi installerar någonting som heter *Guest Additions* som möjliggör att kataloger på värddatorn kan synliggöras för gästsystemen. Vi ska se hur det går till i detalj lite senare, vi ska först börja med att undersöka ett *UNIX*-system.

4. Maskinavbildningar

En möjlighet som virtualisering ger är att vi kan packa ihop en hel maskin, med installerad system- och användarprogramvara i en stor fil som innehåller allt som behövs för att kunna köra denna maskin. Det är precis så vi ska göra för att dela ut den maskin som är anpassad för genomförandet av den här kursen och även kursen i grundläggande programmering. Vi tar tag i filen `NewTinyDebian_2012_04_26.ova` (eller vad den heter, det kan hända att det finns en med ett senare datum när den här kursen ges) och importerar filen med funktionen "*Import Applicance*" i *VirtualBox* (under menyn *File*). Det är viktigt att ni kan göra detta så snart som möjligt. Hur denna fil distribueras till er är inte klart i skrivande stund, det kan hända att jag har DVD-skivor med mig på mötet.

5. Grundläggande kommandon och *UNIX* arkitektur

Vi ska arbeta väldigt mycket textmässigt när vi programmerar. Vi är kanske vana vid ett grafiskt användargränssnitt och det är ganska bekvämt. Men när vi programmerar ger det ofta en bättre förståelse för vad som händer om vi arbetar med textmässiga kommandon. Vi ska börja med att undersöka de klassiska grundläggande kommandona i *UNIX* för att hantera filer och kataloger.

Då vi startar *NewTinyDebian* får vi en inloggningsruta. Vi ska först ange användarnamn och sedan lösenord. Detta är det klassiska sättet att starta en arbetsstund på en *UNIX*-maskin. På *NewTinyDebian* har jag skapat en speciell användare som har namnet `me`, alltså *mig*, på svenska. För att inte krångla till det har jag även satt lösenordet till `me`. Det här är inte så bra, men vi ska inte använda vår maskin i några kritiska situationer så det duger för oss.

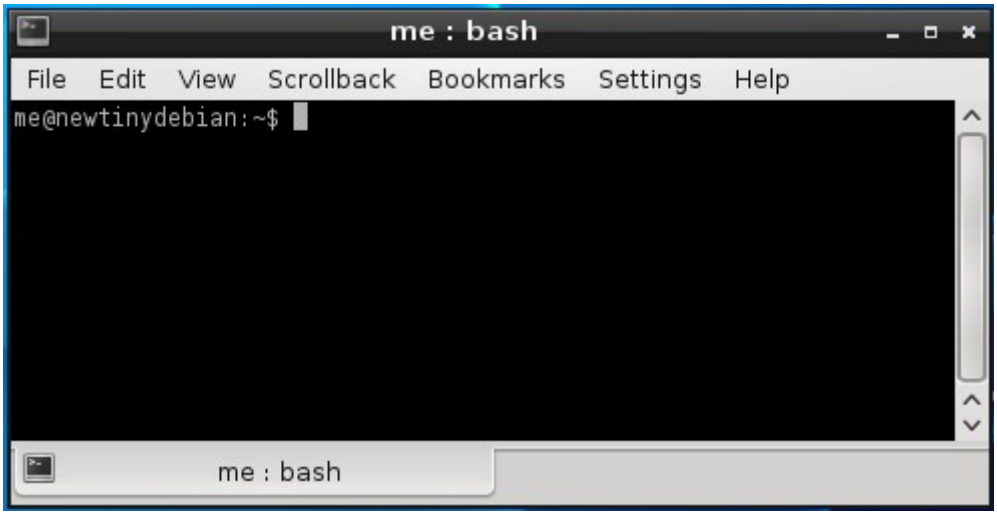
Efter inloggningrutan startar en grafisk miljö som ser ut ungefär så här:

Vi har en skrivbordmiljö som liknar *Windows*, ett grafiskt användargränssnitt som vi nu ska bekanta oss en del med. En skillnad mellan Windows och denna miljö är att listen som man hittar menyer på är i fönstrets överkant istället för nederkanten. Detta kan man ställa in efter sina egna önskemål. Vi börjar med att titta på en förstoring av menylisten i övre vänstra hörnet, vi har här lagt in ett par pilar numrerade från 1 – 8 så att vi ska kunna förklara varje enskilt menyelement.

1 – *LXDE:s huvudmeny*, härifrån kan man finna det mesta man behöver för att göra olika inställningar men också starta program, liknar *Windows* startmeny väldigt mycket.
2 – *LXDE:s file manager*, liknar *Windows utforskaren* (*explorer*).
3 – *En kommandotolk*. Detta kommer att vara ett av våra huvudsakliga verktyg. När vi startar denna

får vi ett fönster där vi kan ge våra textkommandon. Vi kommer in i detalj på det nedan. Den kommandotolk som är installerad är inte den reguljära tolken som hör till LXDE, jag har tagit KDEs "konsole" eftersom man kan förstora/förminska texten i den med en enkel knapptryckning, ctrl och +/-.
4 – *Code::Blocks*, en integrerad utvecklingsmiljö som lämpar sig ganska väl för *C*-programmering. Ett populärt verktyg som ni kan använda om ni vill. Jag rekommenderar dock *Emacs*.
5 – *Grafisk debugger*, `ddd`, ett verktyg som ni kan använda för att detaljstudera era program när det är mycket svårt att hitta fel i dem.
6 – *Emacs*, en mycket bra textredigerare som lämpar sig mycket väl för *C*-programmering. Att arbeta i Emacs har fördelen att vi inte bakar ihop allting i en integrerad miljö, det är många gånger lättare att hantera projekt om de inte är i en integrerad miljö.
7 – *Förminska-alla-fönster-knappen*, om man har hela skärmen full med fönster och man vill se lite bättre kan man trycka på den här knappen. Den stänger inga fönster men förminskar dem bara.
8 – *Flera skrivbord*, en mycket bra egenskap är möjligheten att hantera flera skrivbord samtidigt. Det här är kontrollverktyget för att växla mellan 5 olika skrivbord. Man kan ha olika fönster öppna på olika skrivbord och det är, tycker jag, en mycket stor fördel.

Vi ska nu se närmare på 3:an, kommandotolken. Om vi klickar där får vi upp ett fönster med följande utseende:



Systemet skriver ut en prompt, det är texten `me@tinydebian:~$` och omedelbart efteråt finns markören. Markören finns alltid på skärmen i en kommandotolk och det är genom att skriva in kommandon som man får saker att ske (man ger kommandon som utförs). Dock är inte alltid en kommandotolk beredd att ta emot kommandon eftersom tolken kanske arbetar med att utföra ett tidigare kommando. Först när prompten kommer tillbaka är tolken redo att ta emot ett nytt kommando. Vi kan se hur detta fungerar genom att ge kommandot `sleep 10`. Det ger en order om att ta en paus på 10 sekunder. Om vi gör det så försvinner prompten under 10 sekunder och under denna tid kan inte tolken ta emot andra kommandon.

```
me@newtinydebian:~$ sleep 10
me@newtinydebian:~$ []
```

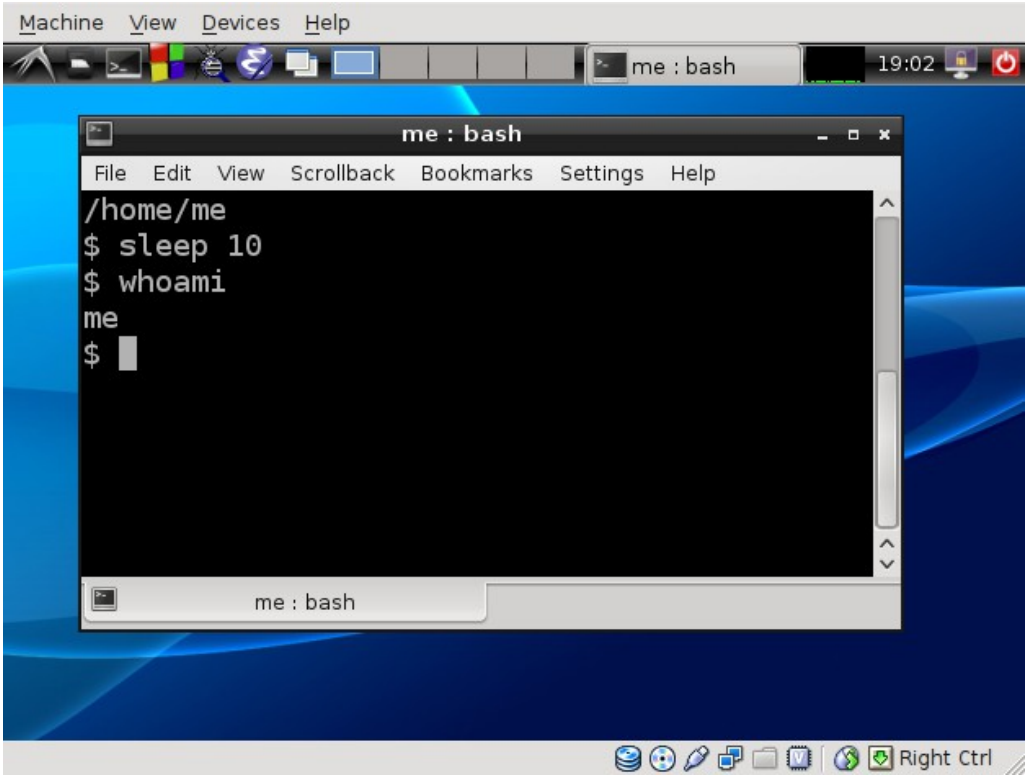
Den andra raden kommer fram först efter 10 sekunder. (`[]` symboliserar en markör). Vi ska nu undersöka de grundläggande kommandona och miljön i det här systemet.

Vi kan börja med kommandon som hjälper till med orienteringen. Det första kommandot heter `whoami` och om man ger det får vi följande resultat:

```
me@newtinydebian:~$ whoami
me
me@newtinydebian:~$ []
```

Tolken skriver ut vilket användarnamn man är inloggad som, och det är `me`, (engelska för *mig*) och efter det så får man tillbaka prompten.

Vi ska ta och byta prompt till en enklare prompt som bara består av ett dollartecken, vi skriver då `. sp` vid prompten. Då startar ett särskilt program (som jag skrivit) som ändrar promptens utseende till endast ett dollartecken. Jag tycker att det är renare med en mindre prompt. Vi kan också trycka ctrl och + som förstorar texten i kommandotolken vilket gör det mer lättläst. Efter dessa manövrar har vi följande utseende:



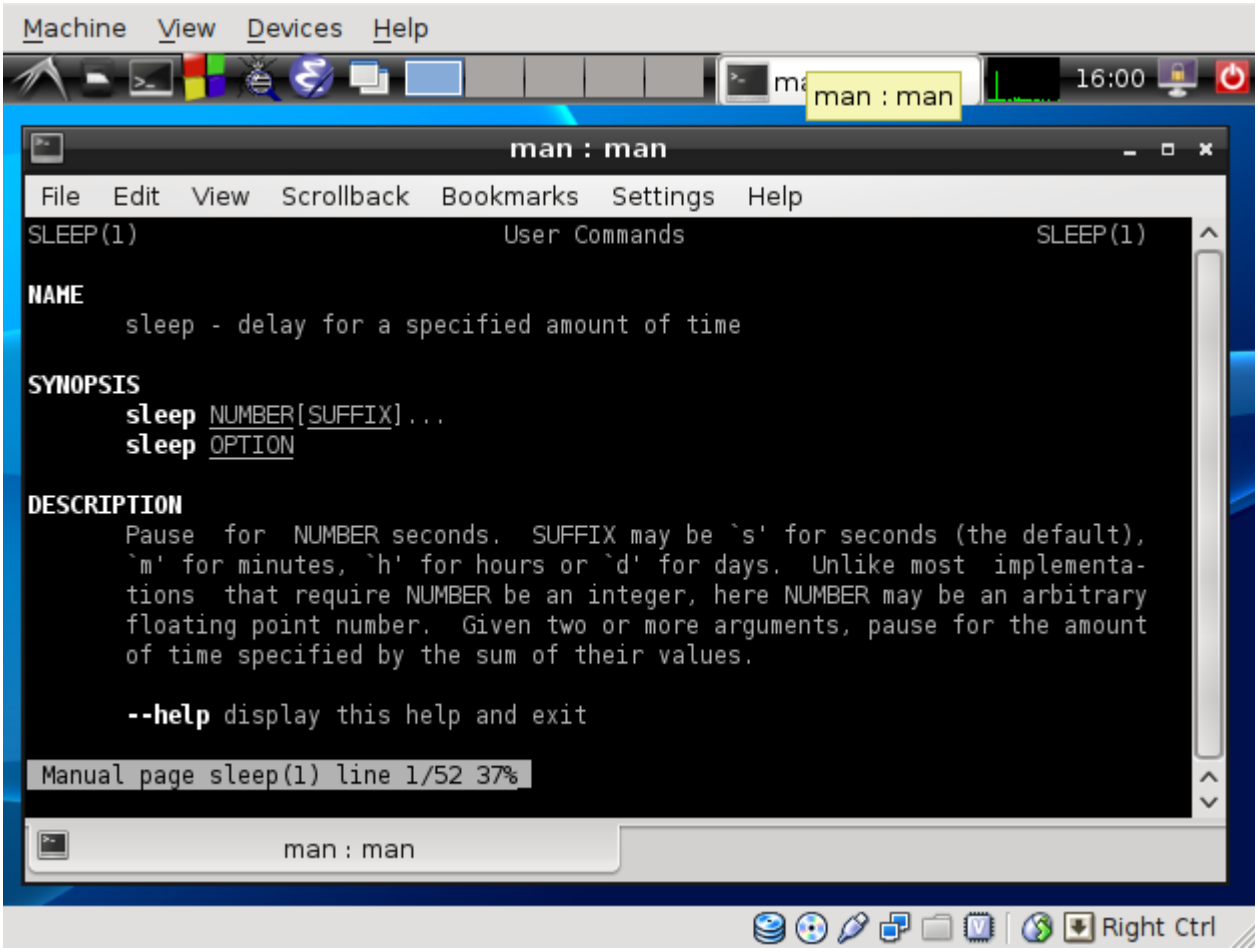
Vi har nu återigen också gett kommandona `sleep 10` följt av `whoami` för att vi ska kunna jämföra med det tidigare utseendet. (Det är förstås en smaksak, men jag föredrar en mindre prompt.)

Överst ser vi en text som inte funnits förut, det är "`/home/me`". Det är någonting som hör till programmet som byter prompt och det blixtrar förbi när man byter prompt. Det är resultatet av ett annat kommando som också hjälper till med orienteringen och det kommandot heter `pwd`, som är en förkortning för *print working directory*. När man arbetar med en kommandoprompt så har man hela tiden ett speciellt fokus på en speciell plats. Denna plats defineras av en katalog och den katalog där man alltså fokuserar sitt arbete kallas *arbetskatalog*. Man kan säga att man *står* i en viss katalog och alla kommandon man ger tar denna katalog som utgångspunkt. När vi ger kommandot

`pwd` så anges det vilken katalog vi står i. Man kan byta arbetskatalog och så att säga gå till en annan katalog och då baserar sig de kommandon man ger på den nya katalogen. Vi ska se hur detta går till men innan vi tittar på mer allmänna kommandon ska titta på två mycket användbara kommandon som ni troligtvis kommer att använda under hela er yrkesverksamma tid som ingenjörer om ni kommer att utveckla teknik som ska kommunicera med eller finnas i ett *UNIX*-system. Dessa kommandon heter `man` och `info`.

Kommandot `man`

Ett *UNIX*-system har till stora delar inbyggd dokumentation i ett system av manualsidor och man kommer åt dessa manualsidor via kommandot man. Vi skriver till exempel `man sleep` vid en prompt och då får vi upp en manualsida som beskriver kommandot `sleep` i detalj. Det ser ut så här:



Kommandotolkens fönster tas över av manualprogrammet som visar en sida med dokumentation över det kommando man vill undersöka. Här ser vi hur dokumentationen presenteras för kommandot sleep. Man bläddrar med piltangenter och trycker **`q`** (*quit*) för att komma ur manualsidan. Det finns olika sektioner i en manualsida. NAME anger namnet på kommandot och en kort beskrivning av vad kommandot gör. “sleep – delay for a specified amount of time”. Nästa sektion heter SYNOPSIS och utgör en beskrivning för hur man anropar detta kommando. Här står det `sleep NUMBER[SUFFIX]` vilket betyder att man ska skriva ett tal på platsen `NUMBER`. Inom

hakparenteser kan man också ange ett suffix (alltså något som man skriver efteråt) för att ange om `sleep` ska sova ett antal minuter eller timmar istället för sekunder som är det som `sleep` sover om man inte anger ett suffix. Exempel:

```
sleep 10
```

sover 10 sekunder som vi sett, NUMBER har då värdet 10. Men

```
sleep 10m
```

sover 10 *minuter* eftersom vi anger suffixet `m` efter talet 10. NUMBER har då värdet 10, men betydelsen ändras från "sov 10 sekunder" till "sov 10 minuter".

Beskrivningen av dessa egenskaper anges längre ner på manualsidan i andra sektioner och för att veta detta behöver man läsa manualsidan. Ta för vana att läsa manualsidor till alla kommandon ni stöter på. Det kan upplevas tungt i början, ni kommer att tjäna mycket på det när ni kommer igång ordentligt med programmeringen. Då slipper man gå till en bok och fråga, "hur var det man anropadet det där kommandot nu igen?" många gånger räcker det med kommandot `man`.

Vi studerar manualsidan till `pwd`:

```
NAME
       pwd - print name of current/working directory

SYNOPSIS
       pwd [OPTION]...

DESCRIPTION
       Print the full filename of the current working directory.
```

Det här är bara ett utdrag, och var `OPTION` betyder kan ni läsa mer om i manualsidan. Ofta när man läser manualsidor stöter man på termer och begrepp man inte förstår och det är OK. Bry er inte om det, läs det ni förstår och pröva er fram. (Eftersom `OPTION` är inom hakparenteser behöver detta kommando ingen `OPTION` och det är så man vanligtvis använder det.)

Vi studerar ett utdrag ur manualsidan till `whoami`:

```
NAME
          whoami - print effective userid

SYNOPSIS
          whoami [OPTION]...

DESCRIPTION
          Print the user name associated with the current effective
          user ID.  Same as id -un.

          --help display this help and exit
```

Här ser vi ett exempel på att vi stöter på ett nytt begrepp, det står "`print effective userid`", vad är det? Det vet vi inte, men strunt i det, vi läser lite längre ner och då står det: "`Print the user name`", vi kan nöja oss med det på det här stadiet. Senare ska vi lära oss vad "effektive user ID" är för något, men just nu kan vi bortse från den exakta betydelsen.

Som sagt manualsystemet är en dokumentation av stora delar av *UNIX*-systemet och faktiskt så finns det en manualsida till `man`-kommandot själv. Om vi skriver `man man` vid en prompt så får vi följande text:

```
NAME
     man - format and display the on-line manual pages

SYNOPSIS
     man  [-acdfFhkKtwW] [--path] [-m system] [-p string]
     [-C config_file] [-M pathlist] [-P pager] [-B browser]
     [-H htmlpager] [-S section_list] [section] name ...


DESCRIPTION
     man formats and displays the on-line manual pages.  If you
     specify section, man only looks in that section of the manual.

     name  is normally  the  name  of  the manual page, which is
     typically the name of a command, function, or file.  However,
     if name contains a slash (/) then man interprets it as a file
     specification, so that you can do man ./foo.5 or even man
     /cd/foo/bar.1.gz.
```

Här ser vi alltså hur man hanterar manualkommandot. Vi kan se det är en massa saker man inte behöver ange (inom hakparenteser) men `name` måste man ange som är det kommando man vill ha upplysningar om.

Manualsystemet i *UNIX* uppfattas ofta av nybörjare som ganska svårt att förstå. Det är naturligt, eftersom de inte alls är pedagogiskt. Men vi rekommenderar ändå att ni tar för vana att studera manualsidor löpande. Ni kommer då att bygga upp en vana att hantera manualsidor och på sikt så kommer ni att uppskatta den precision och saklighet som manualsystemet ändå ger. Hädanefter behöver ni alltså läsa manualsidorna till samtliga kommandon vi stöter på.

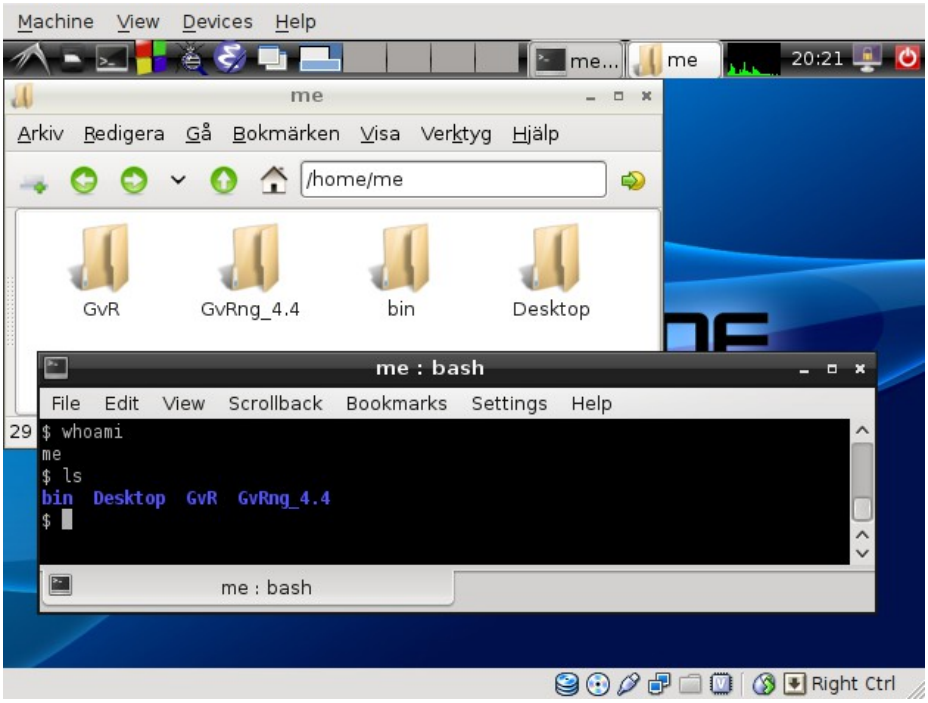
För att underlätta för er vill jag tipsa om kommandot `info` som finns i *GNU*-versioner av *UNIX*-system. Det fungerar väsentligen på samma sätt som kommandot `man`, med kommandot `info` skriver vi till exempel `info sleep` och får då en beskrivning av kommandot `sleep`. Några uppfattar `info` som mer lättförståelig. Kommandona `man` och `info` kompletterar ibland varandra, man kan ibland se hänvisningar i manualsidor till infosidor.

Man kan som sagt ovan skriva `man man`, men man kan också skriva `info info` och för delen även `info man` och `man info`.

Som sagt, använd `man` och `info` som ett stöd.

Kommandot `ls`

Nu kommer vi till ett mycket centralt kommando: `ls`. Det är med hjälp av `ls` som man listar innehållet i arbetskatalogen. Vi ger `ls` då vi står i `/home/me` och vi får då följande resultat:



Vi ser att resultatet av kommandot `ls` är en utskrift av

```
bin  Desktop  Gvr  GvRng_4.4
```

det visas i blått i bilden. Vi har också startat *File manager* i bakgrunden som också visar innehållet i katalogen `/home/me`. Vi ser här att de fyra sakerna, `bin`, `Desktop`, `Gvr` och `GvRng_4.4` är *kataloger* (engelska: *directories*).

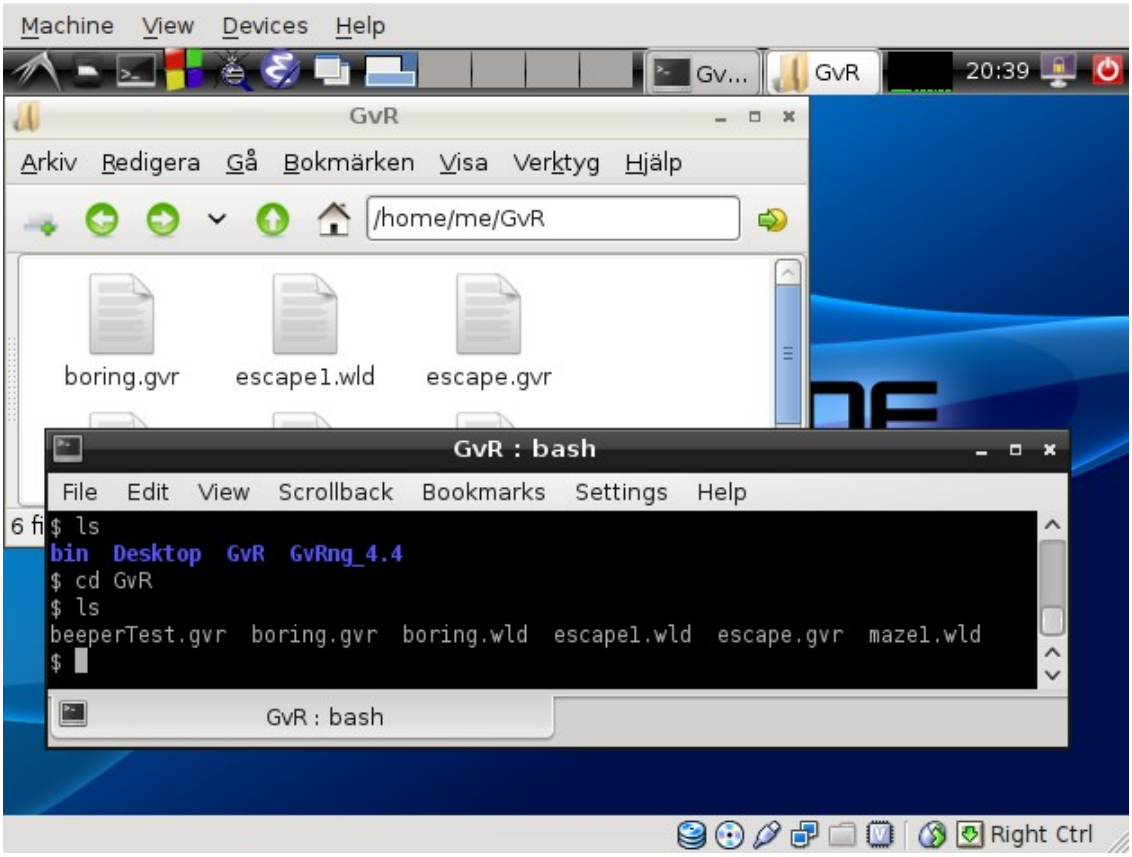
Vi studerar ett utdrag ur manualsidan till `ls`:

```
NAME
     ls - list directory contents

SYNOPSIS
     ls [OPTION]... [FILE]...

DESCRIPTION
     List information about the FILEs (the current directory by
     default).  Sort entries alphabetically if none
     of -cftuvSUX nor –sort.
```

Vi ser här ett exempel på precisionen och sakligheten i manualsystemet, under `DESCRIPTION` så står det "`List information about the FILEs (the current directory by default)`" och det betyder alltså att vi listar information om filerna i arbetskatalogen. Men det står också implicit att `ls` kan fungera på andra sätt, "`information`" är ett ganska vitt begrepp.

Kommandot `cd`

Vi ska nu byta arbetskatalog, det gör man med kommandot `cd`, förkortning för *change directory*, vi skriver `cd GvR` vid prompten för att byta arbetskatalog till `GvR`. Vi passar också på att i *File manager* gå ner i katalogen `GvR`. Efter `cd GvR` gör vi också `ls` för att se vilka filer som finns i `GvR`. Vi ser på resultatet:

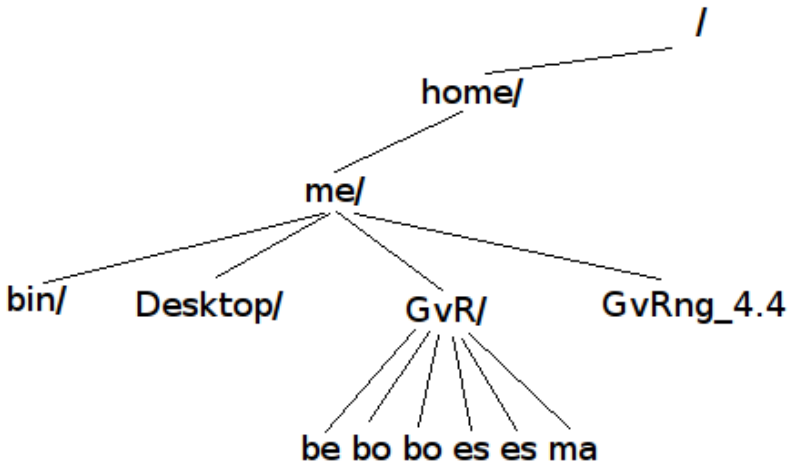


Resultatet är en lista på filer, `beeperTest.gvr`, `boring.gvr`, `boring.wld`, `escape1.wld`, `escape.gvr` och `maze1.wld`. Det är alltså 6 filer i denna katalog och vi kan också se dem grafiskt (även om de skyms lite grann av kommandotolken) med *File manager*.

Vi lägger också märke till här att katalogen i *File manager* nu anges som `/home/me/GvR/`, det vill säga samma som förut, `/home/me/`, fast med `GvR/` påhängt. Det är så man uttrycker att en katalog ligger inuti en annan, katalogen `GvR/` ligger inuti katalogen `/home/me/` och det anges genom att katalogen `GvR/` egentligen heter `/home/me/GvR/`. Det kallas för den *absoluta sökvägen*. Vi ser därmed också att hemkatalogen, `/home/me/`, i själva verket ligger i en annan katalog som heter `/home/`. Faktiskt räknas även `/` som en katalog som katalogen `home/` ligger i och vi har följande struktur:

Filerna `beeperTest.gvr`, `boring.gvr`, `boring.wld`, `escape1.wld`, `escape.gvr` och `maze1.wld` **ligger i** katalogen `GvR/` som **ligger i** katalogen `me/`, som **ligger i** katalogen `home/` som **ligger i** katalogen `/`.

Vi kan rita detta på följande sätt:



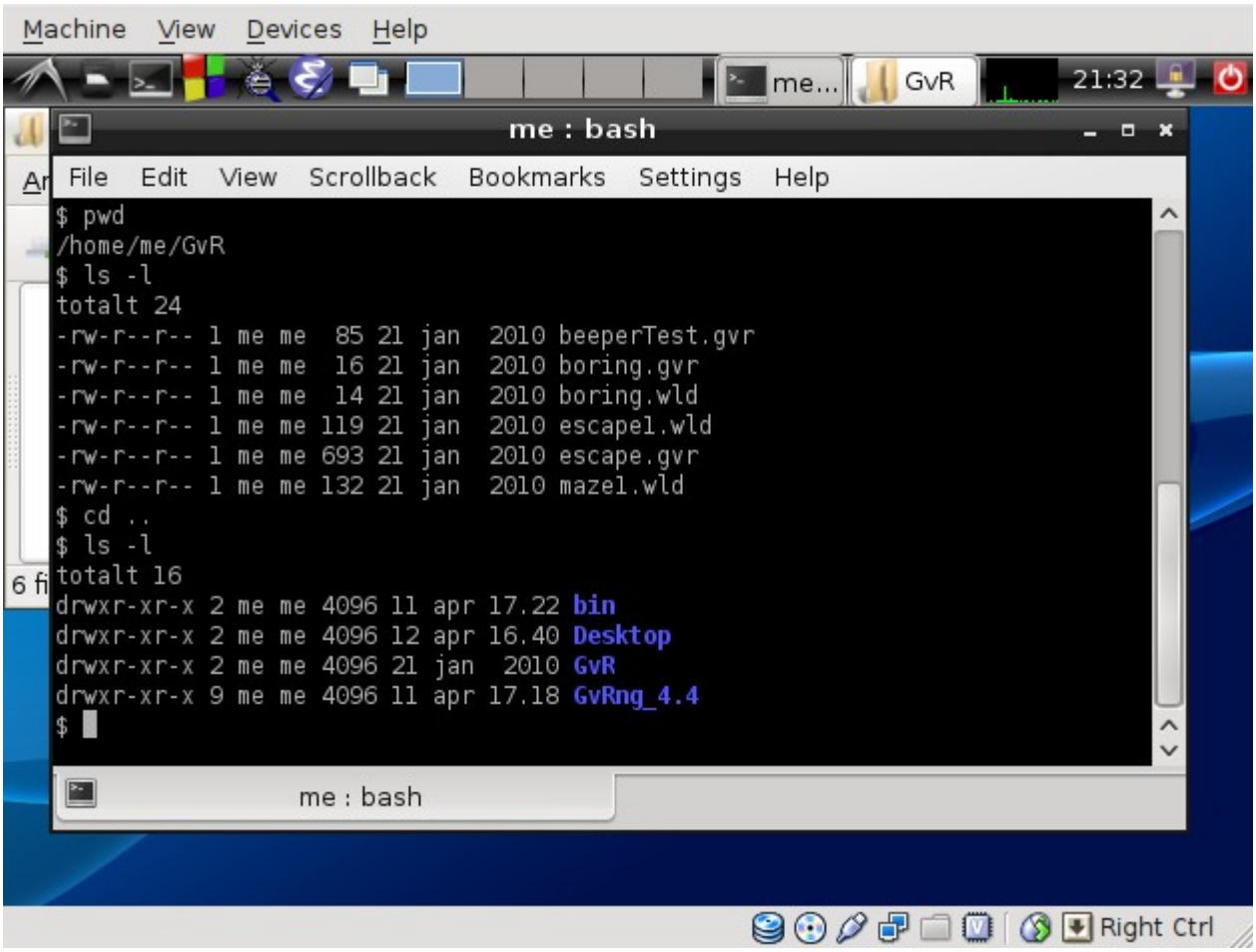
(Vi har av layoutskäl förkortat filnamnen på `beeperTest.gvr`, `boring.gvr`, `boring.wld`, `escape1.wld`, `escape.gvr` och `maze1.wld` till `be bo bo es es ma`.) Lägg märke till hur kataloger och filer bildar en hierarki som liknar ett uppochnervänt träd. Liknelsen med träd gör att man faktiskt kallar katalogen vars namn bara består av ett `/` (den som är överst) för *roten*. Varje katalog är också innehållen i *exakt* en annan katalog, utom just `/` som är filhierarkins högsta punkt. Vi gjorde kommandot `cd GvR` och kom ner i katalogen `/home/me/GvR/`. Om vi vill gå upp igen så kan vi skriva kommandot "`cd ..`", här står då `..` för den katalog som arbetskatalogen är innehållen i. Vi kan skriva `cd /home/me/` då vi står i `/home/me/GvR` för att komma upp ett steg i hierarkin med det blir alltså samma resultat om vi skriver "`cd ..`".

Alla filer och kataloger är inte utritade i figuren ovan, i själva verket har en filhierarki i ett *UNIX*-system tusentals filer och kataloger.

Dokumentation av kommandot `cd`

Faktiskt finns det ingen manualsida för `cd`. Anledningen är att `cd` inte är ett kommando i strikt mening, `cd` är någonting som hör till själva kommandotolken (ett så kallat *inbyggt kommando*, det är inget externt program som körs då vi ger kommandot `cd` till en kommandotolk). Att ha en arbetskatalog är något som varje process och varje process kan byta innevarande katalog, men en process gör detta via ett så kallat systemanrop som heter `chdir`, inte `cd`. Anledningen till varför det är så här ska vi inte gå in närmare på än det som nämnts ovan, vi ska inte ens gå in på betydelsen av termen *process* i den här kursen. Vi ska bara lära oss några basala kommandon och få en liten känsla för hur man hanterar ett *UNIX*-system med en kommandotolk. Vi kan bara acceptera att det inte finns en manualsida för `cd` och lära oss `cd` genom experiment istället. Vissa system (tex *Gentoo*) dokumenterar `cd` via `info` istället för `man`.

Vi kan nu successivt använda `cd`, `ls` och `pwd` för att bilda oss en uppfattning om filhierarkins innehåll. Innan vi gör det kan vi också introducera en annan form av kommandot `ls`, vi kan skriva `ls -l` för att få en "lång" eller mer detaljerad listning av det innehåll som listas. Vi studerar vad som händer om vi gör kommandot `ls -l` då vi står i `/home/me/GvR/` följt av "`cd ..`", alltså att vi går upp ett steg i hierarkin, följt av ytterligare ett `ls -l`. Då får vi följande resultat.



Vi ser då långa listningar med detaljerad information om varje enskild fil. Om vi börjar med den övre listan, som alltså är de 6 filerna i `/home/me/GvR/`, så ser vi tidpunkter, 21 januari 2010 är nämnt som indikerar den tidpunkt då filen i fråga skapades. Kolumnen innan är filens storlek, den innan är vilken grupptillhörighet filen har och den innan är vilken användare som äger filen. Vi går inte in på betydelsen av siffran 1 (som är nästa element) men det kan vara intressant att nämna något om de 10 tecken som står längst till vänster. För alla filer i är dessa tecken `-rw-r--r--`. Det vänstraste av dessa är – och anger filens typ. Eftersom tecknet är ett streck anges att filen är en vanlig fil. (Nedan ser vi att det vänstraste tecknet är ett `d` vilket indikerar att det är kataloger det är fråga om.) Följden av 9 tecken till höger om det vänstraste tecknet (som var ett –) brukar benämnas "rättighetsbitarna" för filerna i fråga. Vi har samma följd av rättighetsbitar för alla sex filer. Rättighetsbitarna delas in i tre gruppen, `rw-` och `r--` och `r--`. Den vänstraste gruppen anger rättigherna som ägaren till filen har, `rw-` betyder då att ägaren kan läsa och skriva men inte exekvera. Det är inte meningsfullt att exekvera en vanlig textfil. Nästa grupp av rättighetsbitar avser rättigheterna hos de som ingår i gruppen som är associerad med filen. Dessa bitar är `r--` vilket anger att medlemmar gruppen får läsa men inget mer, inte skriva till filen eller köra den. De sista rättighetsbitarna anger vad alla andra har för rättigheter och de är tydligen samma som för gruppen. Vi behöver inte fördjupa oss i grupper just nu. Det räcker med att vi vet rättigheter för ägaren till filen. Om vi nu går vidare till listningen under så ser vi att alla dessa filer i själva verket är kataloger eftersom de har ett `d` längst till vänster. Vi får den nya listningen efter vi gått upp ett steg med "`cd ..`". Rättighetsbitarna för kataloger är något annorlunda, exekveringsrättigheter betyder att man har rätt att gå in i katalogen (med `cd`), läsrättigheter betyder att man får göra `ls` i katalogen och skrivrättigheter betyder att man får lägga dit nya filer i katalogen.

Kommandona `mkdir` och `rmdir`

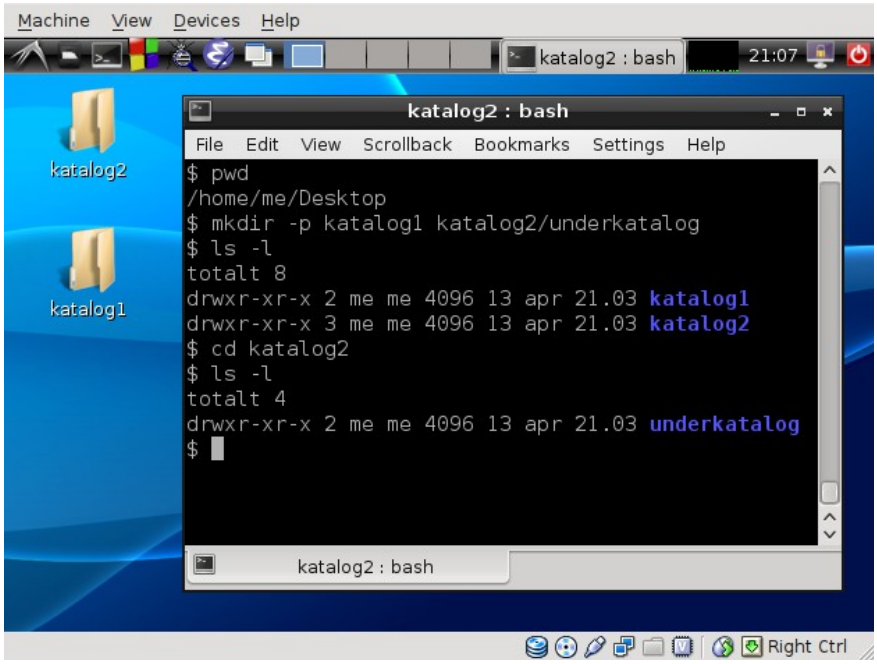
För att skapa nya kataloger och ta bort dem kan man använda sig av kommandona `mkdir` och `rmdir`. Namnen är förstås förkortningar för *make directory* och *remove directory*. Vi ser på utdrag från manualsidor hörande till dessa kommandon:

```
NAME
      mkdir - make directories

SYNOPSIS
      mkdir [OPTION]... DIRECTORY...

DESCRIPTION
      Create the DIRECTORY(ies), if they do not already exist.

      Mandatory arguments to long options are mandatory for short
      options too.

      -m, --mode=MODE
            set file mode (as in chmod), not a=rwx - umask

      -p, --parents
            no error if existing, make parent directories as needed
```

Vi ser här av dokumentationen att ett exempel på ett anrop skulle kunna vara `mkdir minkatalog` och då skapas katalogen `minkatalog`, förutsatt att man har skrivrättigheter i arbetskatalogen. Vi kan också skapa kataloger som ligger i varandra med argumentet `-p`, ett anrop för att skapa två kataloger i svep skulle då kunna vara

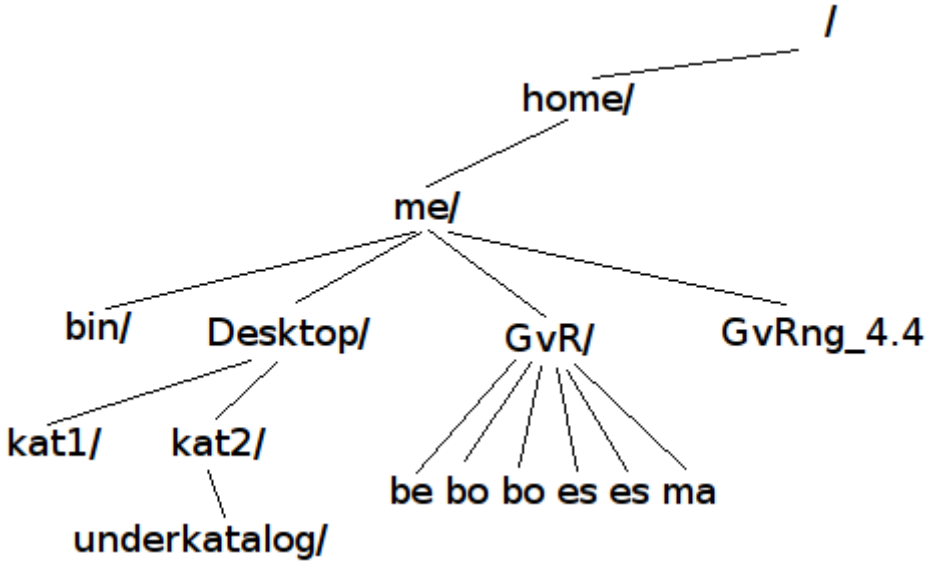
```
mkdir -p minkatalog/underkatalog
```

där då katalogen underkatalog skapas liggandes i katalogen minkatalog. Vi tittar på ett till exempel.



I det här exemplet står vi i katalogen `/home/me/Desktop/` som indikeras av `pwd`. Denna katalog är det som syns grafiskt så förändringar som vi göra här avspeglas ganska omgående i det grafiska användargränssnittet. Alltså när `mkdir`-kommandot körs (som skapar de tre katalogerna) så presenteras detta grafiskt genom att man ser en förändring straxt efteråt. Det behöver inte vara så här, det är bara katalogen `Desktop/` som har en grafisk representation. Men jag valde att göra förändringarna i `Desktop/` för att vi ska kunna jämföra. Efter vi gjort `mkdir` gör vi `ls -l` för att se textrepresentationen av vårt kommando, vi ser då listningen som visar oss de båda katalogerna. Här ser vi inte katalogen `underkatalog`, vi går därför ner i katalogen `katalog2`, med `cd`, och gör `ls -l` där och där kan vi mycket riktigt observera katalogen som heter `underkatalog`.

Vi kan representera det vi skapat med följande trädstruktur:



Vi har inte ritat ut allting här, men vi har indikerat de nya katalogerna under `Desktop/`. (Vi har också förkortat namnen igen av layoutskäl.)

Kommandot `rmdir` fungerar precis som `mkdir`, fast tvärtom, `rmdir` tar bort de kataloger man anger, de existerar inte efter `rmdir` har lyckats. Dock måste de kataloger man anger till `rmdir` vara tomma. Det betyder att om vi står i katalogen `Desktop/` (alltså om det är vår arbetskatalog) och ger kommandot `rmdir katalog1 katalog2` så kommer rmdir att lyckas ta bort `katalog1` eftersom den är tom. Kommandot `rmdir` kommer dock att ge ett felmeddelande och säga att det inte lyckades med att ta bort `katalog2` eftersom den inte är tom, den innehåller ju katalogen som heter `underkatalog`. Om man vill ta bort `katalog2` med `rmdir` måste man först gå ner i `katalog2` med `cd`, ta bort `underkatalog` med `rmdir` (det går bra eftersom den är tom), sedan gå upp ett steg, med "`cd ..`", så att vi återigen har arbetskatalogen `Desktop/`, och nu, på nytt ge kommandot `rmdir katalog2`. Då kommer det att lyckas.

Övning till detta material hittar ni i filen `Ovningar.pdf`. Dessa är mycket viktiga och förbereder er för att kunna arbeta vidare med kursen.