



# Laboration – Terminalen

OBS det kan förekomma små skillnader vad gäller sökvägar samt befintliga kommandon mellan olika distributioner. T.ex. mellan CentOS och Ubuntu,

**Material:** För att genomföra laborationen behöver man ha tillgång till en dator (vm) med Linux installerat. Windows Subsystem for Linux går att använda också. För vissa övningar kan lokal administratörsbehörighet behövas. I labben utgår det från att man är inloggad med en användare som heter **sysadmin**

**Mål:** I denna laboration kommer du att utföra följande uppgifter:

- Utforska Bash-funktioner
- Använda skalvariabler
- Kunna använda citattecken (quoting)

## Filer och mappar

I den här uppgiften kommer vi att få tillgång till Command Line Interface (CLI) för Linux att undersöka hur man utför grundläggande kommandon och vad som påverkar hur de kan utföras.

De flesta användare är förmodligen mer bekanta med hur kommandon körs med hjälp av ett grafiskt användargränssnitt (GUI). Därför kommer denna uppgift sannolikt att presentera några nya begrepp för dig om du inte tidigare har arbetat med en CLI. För att använda en CLI måste du skriva kommandot som du vill köra.

Fönstret där du skriver ditt kommando kallas en terminal emulatorapplikation. Inuti Terminalfönstret visar systemet en uppmaning, som för närvarande innehåller en *prompt* följt av en blinkande markör:

```
sysadmin@localhost:~$
```

Prompen berättar att du är **sysadmin**; värden eller datorn du använder: **localhost**; och katalogen där du är på: **~**, som representerar din hemkatalog.

När du skriver ett kommando visas det vid textmarkören. Du kan använda tangenter som HOME, END, backspace och piltangenterna för att redigera kommandot du skriver.

Lika viktigt är kommandoradssyntaxen, som är den ordning i vilken kommandot, alternativet och argumenten/argumenten måste föras in i prompten så att skalet känner igen hur man korrekt utför kommandot. Korrekt kommandoradssyntax ser ut som följande:

```
command [options] [arguments]
```

Skriv in kommandot **ls** vid prompten för att visa vilka filer och mappar som finns i din nuvarande arbetskatalog. I det här exemplet används inga alternativ eller argument.



När du har skrivit in kommandot korrekt, tryck på **Enter** för att köra det.

```
sysadmin@localhost:~$ls
Desktop Documents Downloads Music Pictures Public Templates
Videos
ls -l
```

Kommandot **ls** används för att visa information om mappar och filer, och som standard visar det information för den aktuella katalogen. Använd alternativet **-l** för att visa denna information i ett långt format, vilket ger ytterligare detaljer om filer som finns i den nuvarande arbetskatalogen:

Resultatet borde så ut så här:

Observera att kataloger betraktas som en typ av fil i Linux filsystem.

```
sysadmin@localhost:~$ls -l
total 32
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Desktop
drwxr-xr-x 4 sysadmin sysadmin 4096 Oct 31 19:52 Documents
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Downloads
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Music
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Pictures
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Public
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Templates
drwxr-xr-x 2 sysadmin sysadmin 4096 Oct 31 19:52 Videos
```

Du kan också lägga till argument till kommandon. Om du anger sökvägen till en specifik katalog tillsammans med kommandot **ls** visas information om just den katalogen. Använd argumentet **/home** för att visa detaljerad information om filer i katalogen **/home**.

```
ls -l /home
```

Din utdata ska likna följande:

```
sysadmin@localhost:~$ls -l /home
total 4
drwxr-xr-x 1 sysadmin sysadmin 4096 Aug 8 17:46 sysadmin
```

Genom att använda alternativet **-l** och argumentet **/home** ser vi nu att katalogen **/home** innehåller en mapp som heter **sysadmin**

Följande kommando visar samma information som du ser i den första delen av prompten. Se till att du först har valt (klickat på) **Terminal**-fönstret och skriv sedan in följande kommando, följt av att du trycker på **Enter**:

```
whoami
```

Din utdata ska likna följande:

```
sysadmin@localhost:~$ whoami
sysadmin
sysadmin@localhost:~$
```



Utmatningen av kommandot `whoami`, alltså `sysadmin`, visar användarnamnet för den aktuella användaren. Även om ditt användarnamn redan syns i prompten i det här fallet, kan det här kommandot användas för att få fram informationen om prompten inte visar det.

Nästa kommando visar information om det aktuella systemet. För att kunna se namnet på den kärna du använder, skriv in följande kommando i terminalen:

```
uname
```

Din utdata bör likna det här:

```
sysadmin@localhost:~$ uname
Linux
```

Många kommandon som körs ger textutmatning som denna. Du kan ändra vilken utmatning ett kommando ger genom att använda *alternativ* efter kommandonamnet.

Alternativ för ett kommando kan anges på flera sätt. Traditionellt i UNIX uttrycktes alternativ med ett bindestreck följt av en annan bokstav, till exempel: `-n`.

I Linux kan alternativ ibland också anges med två bindestreck följt av ett ord eller ett sammansatt ord, till exempel: `--nodename`.

Kör `uname`-kommandot igen två gånger i terminalen, en gång med alternativet `-n` och en gång med alternativet `--nodename`. Detta visar nätverksnodens värddamn, vilket också syns i prompten.

```
uname -n
uname --nodename
```

Din utdata bör likna det här:

```
sysadmin@localhost:~$ uname -n
localhost
sysadmin@localhost:~$ uname -nodename
localhost
```

Kommandot `pwd` används för att visa din nuvarande "plats" eller nuvarande "arbetande" katalog. Skriv in följande kommando för att visa arbetskatalogen:

```
pwd
```

Din utdata bör likna det här:

```
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```

Den aktuella katalogen i exemplet ovan är `/home/sysadmin`. Detta kallas också din *hemkatalog*, en speciell plats där du har kontroll över filer och andra användare normalt inte har tillgång. Som standard heter katalogen samma som ditt användarnamn och ligger under katalogen `/home`.

Som du kan se från kommandots utdata, `/home/sysadmin`, använder Linux snedstreck `/` för att separera kataloger och skapa det som kallas en *sökväg*. Det första snedstreckat representerar den översta katalogen, som kallas rotkatalogen. Mer information om filer, kataloger och sökvägar kommer att presenteras i senare labbar.



Tilde-tecknet `~` som du ser i prompten visar också vilken aktuell katalog du befinner dig i. Det här tecknet är ett "genvägssätt" att representera din hemkatalog.

### Fundera på detta

`pwd` står för "print working directory". Det "printar" egentligen inte i moderna versioner, men äldre UNIX-maskiner hade inga skärmar så kommandoutdata skickades till en skrivare, därav det något missvisande namnet `pwd`.

## Kommandohistorik

Bash-skalet sparar en historik över de kommandon du skriver in. Tidigare kommandon kan enkelt nås på flera olika sätt via denna historik.

Det första och enklaste sättet att återkalla ett tidigare kommando är att använda **uppåt-pilen**. Varje gång du trycker på **uppåt-pilen** bläddrar du ett steg bakåt i kommandohistoriken. Om du råkar gå för långt tillbaka kan du använda **nedåt-pilen** för att gå framåt genom historiken.

När du hittat kommandot du vill köra kan du använda **vänsterpil** och **högerpil** för att placera markören och redigera. Andra användbara tangenter för redigering är **Home**, **End**, **Backspace** och **Delete**.

Ett annat sätt att använda din kommandohistorik är att köra kommandot `history` för att visa en *numrerad* lista med historik. Numret till vänster om kommandot kan användas för att köra kommandot igen. `History`-kommandot har också ett antal alternativ och argument som kan styra vilka kommandon som sparas eller visas.

Kör ett nytt kommando och kör sedan kommandot `history`:

```
echo Hej
history
```

### Kom ihåg

Kommandot `date` visar tid och datum på systemet. Kommandot `clear` rensar skärmen.

Din utdata bör likna följande:

```
sysadmin@localhost:~$ history
 1  ls
 2  ls -l
 3  ls -l /tmp
 4  whoami
 5  uname
 6  uname -n
 7  uname --nodename
 8  pwd
 9  echo Hej
10  history
sysadmin@localhost:~$
```

Dina kommando-nummer kan skilja sig från de som visas ovan. Det beror på att du kan ha kört ett annat antal kommandon sedan du öppnade den virtuella terminalen.



För att visa ett begränsat antal kommandon kan `history`-kommandot ta ett nummer som parameter och visa exakt så många senaste poster. Skriv följande kommando för att visa de fem senaste kommandona från din historik:

```
history 5
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ history 5
 7  uname --nodename
 8  pwd
 9  echo Hej
10  history
11  history 5
```

För att köra ett kommando igen, skriv utropstecken följt av numret i historiklistan. Till exempel, för att köra det nionde kommandot i din historiklista, gör du så här

```
!9
```

```
sysadmin@localhost:~$ !9
echo Hej
Hej
```

Prova nu att bläddra i din historik med hjälp av **uppåtpil** och **nedåtpil** på tangentbordet. Fortsätt att trycka på **uppåtpil** tills du hittar ett kommando du vill köra. Om du behöver, använd andra tangenter för att redigera kommandot och tryck sedan på **Enter** för att köra kommandot.

## Shell variabler

Shellvariabler används för att lagra data i Linux. Denna data används både av själva skalet, program och användare.

Syftet med detta avsnitt är att lära sig hur man visar värdena för shellvariabler.

Kommandot `echo` kan användas för att skriva ut text och värdet av en variabel, samt visa hur skalets miljö expanderar *metatecken* (mer om metatecken senare i denna labb). Skriv följande kommando för att få den att skriva ut bokstavlig text:

```
echo Hello Student
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo Hello Student
Hello Student
```

Miljövariabler är tillgängliga för hela systemet. Systemet återskapar automatiskt miljövariabler varje gång ett nytt skal öppnas. Exempel på sådana variabler är `PATH`, `HOME` och `HISTSIZE`. `HISTSIZE`-variabeln bestämmer hur många tidigare kommandon som sparas i historiklistan. I exemplet nedan kommer kommandot att visa värdet på `HISTSIZE`-variabeln:



```
sysadmin@localhost:~$ echo $HISTSIZE
1000
```

Skriv följande kommando för att visa värdet på PATH-variabeln:

```
echo $PATH
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/b
in:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

**PATH**-variabeln visas genom att sätta ett **\$**-tecken framför variabelnamnet.

Denna variabel används för att hitta platsen för kommandon. Varje katalog som listas ovan söks igenom när du kör ett kommando. Till exempel, om du försöker köra kommandot **date**, kommer skalet först att leta efter kommandot i katalogen **/home/sysadmin/bin** och därefter i katalogen **/usr/local/sbin** och så vidare. När kommandot **date** har hittats, "kör" skalet det.

Använd kommandot **which** för att ta reda på om det finns en körbar fil, i detta fall vid namn **date**, som ligger i någon av katalogerna som anges i **PATH**-variabeln:

```
which date
```

Din utdata bör se ut ungefär så här:

```
sysadmin@localhost:~$ which date
/bin/date
```

Utdata från kommandot **which** visar att när du kör kommandot **date** kommer systemet att köra kommandot **/bin/date**. Kommandot **which** använder **PATH**-variabeln för att avgöra var **date**-kommandot finns.

## Kommandotyper

I det här avsnittet kommer vi att gå igenom de fyra typerna av kommandon som används i Linux. Genom att förstå var dessa kommandon kommer ifrån och hur de skiljer sig åt kan en administratör hantera systemet mer effektivt.

Ett sätt att lära sig mer om ett kommando är att ta reda på var det kommer ifrån. Kommandot **type** kan användas för att få information om kommandotypen.

```
type kommando
```

Det finns flera olika källor till kommandon i skalet på din CLI:

*Interna kommandon* är inbyggda i själva skalet. Ett bra exempel är kommandot **cd** (byta katalog), eftersom det är en del av Bash-skalet. När en användare skriver **cd** är Bash-skalet redan igång och vet hur det ska tolka kommandot, vilket innebär att inga extra program behöver startas.

Kommandot **type** identifierar **cd** som ett internt kommando:

```
sysadmin@localhost:~$ type cd
cd is a shell builtin
```



*Externa kommandon* är binära program som finns i kataloger som skalet söker igenom. Om en användare skriver kommandot `ls`, söker skalet igenom de kataloger som finns listade i `PATH`-variabeln för att hitta en fil som heter `ls` och som kan köras. Du kan använda kommandot `which` för att visa den fullständiga sökvägen till `ls`-kommandot.

```
which ls
```

```
sysadmin@localhost:~$ which ls
/bin/ls
```

För externa kommandon visar kommandot `type` platsen där kommandot finns:

```
sysadmin@localhost:~$ type cp
cp is /bin/cp
```

I vissa fall kan utdata från kommandot `type` skilja sig avsevärt från utdata från kommandot `which`:

```
sysadmin@localhost:~$ which cp
/bin/cp
```

Om du använder `-a`-flaggan med kommandot `type` visas alla platser där kommandot finns:

```
sysadmin@localhost:~$ type -a ls
ls is aliased to 'ls --color=auto'
ls is /bin/ls
```

*Alias* kan användas för att koppla längre kommandon till kortare tangentsekvenser. När skalet upptäcker att ett alias används, ersätter det den längre kommandosträngen innan det tolkar kommandona vidare.

För att ta reda på vilka alias som är inställda i det aktuella skalet, använd kommandot `alias`:

```
sysadmin@localhost:~$ alias
alias alert='notify-send --urgency=low -i "[ $? = 0 ] && echo
terminal || echo
error"' "$(history|tail -n1|sed -e '\s/\s*[0-
9]\+\s*//;s/[;&]\s*alert$//\s*\s*')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -aLF'
alias ls='ls --color=auto'
```

Den sista typen av kommando är *körbart program*. Dessa kommandon startar program som är installerade på systemet och utför specifika uppgifter. När en användare skriver kommandot `vi` letar skalet upp programmet via sökvägarna i `PATH`-variabeln och kör det. Program som vi finns på nästan alla Linux-distributioner, medan andra program, som `vlc` (en öppen källkodsmediaspelare som ofta används på Linux-skrivbord), installeras av användare eller administratörer för ett särskilt syfte och kommer inte att finnas i `PATH` om de inte installerats separat.

```
type vi
```



```
cd /bin
type vlc
cd
```

```
sysadmin@localhost:~$ type vi
vi is /usr/bin/vi
sysadmin@localhost:~$ cd /bin
sysadmin@localhost:/bin$ type vlc
-bash: type: vlc: not found
sysadmin@localhost:/bin$ cd
sysadmin@localhost:~$
```

## Citattecken

Det finns tre typer av citattecken som används i Bash-skalet: enkla citattecken (`'`), dubbla citattecken (`"`) och bakåtvända citattecken (```). Dessa tecken har särskilda funktioner i Bash-skalet som beskrivs nedan.

För att förstå skillnaden mellan enkla och dubbla citattecken, tänk på att det ibland finns tillfällen när du inte vill att skalet ska behandla vissa tecken som *specialtecken*. Till exempel används tecknet `*` som en *jokertecken*. Men vad gör du om du vill att `*` bara ska betyda en bokstavlig asterisk?

- Enkla citattecken `'` förhindrar skalet från att "tolka" eller expandera alla specialtecken. Ofta används enkla citattecken för att skydda en sträng (en teckensekvens) från att ändras av skalet, så att strängen kan användas som ett argument till ett kommando och därmed påverka hur kommandot körs.
- Dubbla citattecken `"` stoppar expansionen av glob-tecken som asterisk (`*`), frågetecken (`?`) och hakparenteser (`[]`). Dubbla citattecken *tillåter* dock både variabelexpansion och kommandosubstitution (se bakåtvända citattecken).
- Bakåtvända citattecken ``` orsakar *kommandosubstitution* vilket gör det möjligt att köra ett kommando inom raden för ett annat kommando.

När du använder citattecken måste de anges i par, annars kommer skalet inte att anse att kommandot är komplett.

Även om enkla citattecken är användbara för att blockera skalet från att tolka ett eller flera tecken, finns det också ett sätt att blockera tolkningen av bara ett enda tecken, nämligen att "escape:a" tecknet. För att *escape:a* den speciella betydelsen av ett skalmetatecken används backslash `\` som ett prefix framför det tecknet.

Kör följande kommando som använder bakåtvända citattecken ``` (finns under `~` på vissa tangentbord) för att köra kommandot `date` inom raden för kommandot `echo`:

```
echo Today is `date`
```

Din utdata bör se ut ungefär så här:

```
sysadmin@localhost:~$ echo Today is `date`
Today is Mon Feb 28 20:47:22 UTC 2026
```



Du kan också placera \$( före kommandot och ) efter kommandot för att åstadkomma kommandosubstitution:

```
echo Today is $(date)
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo Today is $(date)
Today is Mon Feb 28 20:47:22 UTC 2026
```

Varför finns det två olika metoder som gör samma sak? Bakåtvända citattecken liknar enkla citattecken, vilket gör det svårare att "se" vad ett kommando ska göra. Ursprungligen använde skal bakåtvända citattecken; formatet \$( kommando ) lades till i en senare version av Bash-skalet för att göra raden mer visuellt tydlig.

Om du inte vill att bakåtvända citattecken ska användas för att köra ett kommando, sätt enkla citattecken runt dem. Kör följande:

```
echo Detta är kommandot `date`
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo This is the command `date`
This is the command `date`
```

Observera att du också kan placera ett omvänt snedstreck \ framför varje bakåtvänd citattecken. Kör följande:

```
echo Detta är kommandot `date`
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo This is the command `date`
This is the command `date`
```

Dubbelcitattecken " påverkar inte bakåtvända citattecken. Skalet kommer fortfarande att använda dem för kommandosubstitution. Kör följande för att se ett exempel:

```
echo Detta är kommandot "`date`"
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo This is the command "`date`"
This is the command Mon Feb 12 20:51:10 UTC 2026
```

Dubbelcitattecken påverkar jokertecken och gör att deras speciella betydelse försvinner. Kör följande:

```
echo D*
echo "D*"
```

Din utdata bör likna följande:



```
sysadmin@localhost:~$ echo D*
Desktop Documents Downloads
sysadmin@localhost:~$ echo "D*"
D*
sysadmin@localhost:~$
```

### Viktigt

Att citera kan verka banalt och knepigt just nu, men ju mer erfarenhet du får av att arbeta i kommandoskalet, desto mer kommer du inse att det är avgörande att förstå hur olika citattecken fungerar för att kunna använda skalet effektivt.

## Control statements

Vanligtvis skriver du ett enda kommando och kör det när du trycker på **Enter**. Bash-skalet erbjuder tre olika satser som du kan använda för att separera flera kommandon som skrivs ihop.

Den enklaste separatoren är semikolon (;). Om du använder semikolon mellan flera kommandon körs de efter varandra, i ordning från vänster till höger.

Tecknen && skapar ett logiskt "och"-villkor. Kommandon som separeras med && körs villkorligt. Om kommandot till vänster om && lyckas, kommer kommandot till höger också att köras. Om kommandot till vänster misslyckas, körs inte kommandot till höger.

Tecknen || skapar ett logiskt "eller"-villkor och innebär också villkorlig körning. När kommandon separeras med ||, körs kommandot till höger endast om kommandot till vänster misslyckas. Om kommandot till vänster lyckas, körs inte kommandot till höger.

För att se hur dessa styrsatser fungerar kommer du att använda två speciella program: `true` och `false`. Programmet `true` lyckas alltid när det körs, medan `false` alltid misslyckas. Även om det inte ger realistiska exempel på hur && och || fungerar, ger det ändå ett sätt att visa hur de fungerar utan att behöva introducera nya kommandon.

Kör de följande tre kommandona tillsammans, separerade med semikolon:

```
echo Hello; echo Linux; echo Student
```

Som du ser visar utdata att alla tre kommandon körs i följd:

```
sysadmin@localhost:~$ echo Hello; echo Linux; echo Student
Hello
Linux
Student
sysadmin@localhost:~$
```

Nu, skriv ihop tre kommandon separerade med semikolon, där det första kommandot körs och ger ett felresultat:

```
false; echo Not; echo Conditional
```

Ditt utdata bör likna det här:



```
sysadmin@localhost:~$ false; echo Not; echo Conditional
Not
Conditional
sysadmin@localhost:~$
```

Notera att i det tidigare exemplet kördes ändå alla tre kommandon, även om det första misslyckades. Det syns inte i utdata från `false`-kommandot, men det kördes faktiskt. När kommandon separeras med tecknet `;` är de helt oberoende av varandra.

Nästa steg: använd logiskt "och" för att separera kommandona:

```
echo Start && echo Going && echo Gone
```

Ditt utdata bör likna följande:

```
sysadmin@localhost:~$ echo Start && echo Going && echo Gone
Start
Going
Gone
sysadmin@localhost:~$
```

Eftersom varje `echo`-kommandot körs utan problem returneras ett lyckat resultat, vilket gör att nästa kommando också körs.

Använd logiskt "och" med ett kommando som misslyckas enligt exemplet nedan:

```
echo Success && false && echo Bye
```

Ditt utdata bör likna det här:

```
sysadmin@localhost:~$ echo Success && false && echo Bye
Success
sysadmin@localhost:~$
```

Det första `echo`-kommandot lyckas och vi ser dess utdata. `false`-kommandot körs och misslyckas, så det sista `echo`-kommandot körs inte.

Tecknen "eller" som separerar kommandona nedan visar hur ett misslyckande före "eller"-uttrycket gör att kommandot efteråt körs; däremot gör ett lyckat första kommando att det efterföljande inte körs.

```
false || echo Fail Or
true || echo Nothing to see here
```

Ditt utdata bör likna det här:

```
sysadmin@localhost:~$ false || echo Fail Or
Fail Or
sysadmin@localhost:~$ true || echo Nothing to see here
sysadmin@localhost:~$
```