Modul 6

Filrättigheter och miljön

Sebastian Bengtegård, Johan Holmberg

22 Januari 2025

Innehållsförteckning

1 Förord

I denna modul kommer vi att introduceras till sju nya kommandon:

- `chmod` (*change file mode bits*)
- `sudo` (*execute a command as another user*)
- `chown` (*change file owner*)
- `chgrp` (*change group ownership*)
- `type` (*display information about a command*)
- `printenv` (*print all or part of environment*)
- `echo` (*display a line of text*)

Genom dessa nya kommandon kommer vi att kunna arbeta med filrättigheter, det vill säga bestämma vem och vilken åtkomst någon ska ha till en fil. Vi kommer även att lära oss lite mer om hur vår arbetsmiljö fungerar (vad är egentligen ett kommando?).

Modulen avslutas sedan med en övning, som inte inkluderar en förberedd övningsmapp.

Denna modul är mer informationstät och inte lika praktisk som tidigare moduler, däremot minst lika viktig. Vissa kommandon presenteras endast för vår kännedom och kommer därför inte inkluderas i den avslutande övningen.

2 Filrättigheter

Det är inte ovanligt att en dator har fler än en användare och på grund av detta vill vi kunna bestämma vem som kan komma åt våra filer. Vi kan till exempel ange att en användare endast kan öppna och läsa av innehållet i en fil, men samtidigt inte kunna redigera och spara filen.

Detta görs genom att ange vilka rättigheter en fil ska ha. Vi har tre typer av rättigheter, som även kan kombineras hur vi vill.

1. Läsa (*read*)
2. Skriva (*write*)
3. Exekvera (*execute*)

Dessa brukar förkortas till bokstäverna **r** (*read*), **w** (*write*) och **x** (*execute*).

Läsrättigheter innebär att vi kan öppna och läsa av filen, t.ex. genom programmet `less`. Skrivrättigheter innebär att vi kan redigera och spara våra ändringar, t.ex. genom programmet `nano`. Den sista typen bestämmer om en fil är exekverbar eller inte, vilket

är detsamma som om vi kan "köra" eller använda oss av den, t.ex. är kommandot `ls` exekverbart (och även en fil, men mer om det i ett senare avsnitt).

Alla filer och mappar har någon kombination av filrättigheter och för att se vilka dessa är kan vi använda oss av kommandot `ls -l` (*long format*).

```
sebastian@mindator:~$ ls -l
-rw-r--r--  1 sebastian sebastian    3526 Nov 10 22:16 example.txt
drwxr-xr-x  6 sebastian sebastian    4096 May  5 09:45 Desktop
drwxr-xr-x  3 sebastian sebastian    4096 Mar 27 18:06 Documents
drwxr-xr-x  4 sebastian sebastian    4096 May  4 10:36 Downloads
-rw-r--r--  1 sebastian sebastian     252 Feb  3 14:28 essay.pdf
...
```

Den första kolumnen (t.ex. `-rw-r--r--` eller `drwxr-xr-x`) består av filtypen följt av alla rättigheter. Det första tecknet anger filtypen, ett `-` representerar en vanlig fil och `d` en mapp (*directory*).

De resterande tecknen skrivs som `rwx rwx rwx` (tre kolumner). Varje kolumn representerar, i ordning, rättigheterna för:

1. den som äger filen.
2. den grupp som har tillgång till filen.
3. övriga.

Ett bindestreck bland dessa innebär att rättigheten (`r`, `w` eller `x`) är avstängd.

Om vi tar de två översta filerna från vår utskrift ovan kan vi avläsa följande:

1. `-rw-r--r--`: det första tecknet visar att detta är en vanlig fil. Därefter kan vi avläsa att ägaren av filen har läs- och skrivrättigheter (`rw-`) och alla andra har endast läsrättigheter (`r--`).
2. `drwxr-xr-x`: det första tecknet visar att detta är en mapp. Därefter kan vi avläsa att ägaren av mappen har alla rättigheter (`rwx`) och alla andra har rättighet till att läsa och exekvera (`r-x`).

Rättigheten att kunna "exekvera" en mapp innebär att vi kan gå ner i den eller att öppna den, t.ex. med kommandot `cd`*.*

Kom ihåg att kommandot `ls -l` även visar oss vem som äger filen samt vilken grupp som har tillgång till den.

3 `chmod`: Ändra en fils rättighetsinställningar

För att ändra rättigheterna på en eller flera filer använder vi oss av kommandot `chmod`.

```
sebastian@mindator:~$ chmod permissions file...
```

För att ange rättigheterna på en fil gör vi detta vanligtvis med notationen *octal* (dvs. med siffror från `0` till `7`). Till exempel `644`, där respektive siffra, i ordning, representerar rättigheterna för den som äger filen (`6`), gruppen som har tillgång till den (`4`) och övriga (`4`).

Vad betyder då siffrorna? För att förstå detta behöver vi komma ihåg att en dator arbetar bara med siffrorna `1` och `0`, så kallade *bits*. Flera *bits* tillsammans kan representera större siffror, det vi kallar för binära tal (*binary*). Till exempel är det binära talet `110` detsamma som decimaltalet `6`.

Rättigheter kan ses som en serie av tre *bits*, det vill säga ett binärt tal, där varje *bit* anger om en viss typ av rättighet ska vara på eller av. Skulle vi till exempel vilja ange att en fil endast ska ha läs- och skrivrättigheter, dvs. `rw-`, hade det varit det binära talet `110`, som samtidigt är decimaltalet `6`.

För att få en tydligare bild av detta kan du ta en titt på överblicken nedan.

```
rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx r-- --- = 111 100 000
rw- r-- r-- = 110 100 100

och så vidare...

rwx = 111 (binary) = 7 (decimal)
rw- = 110 (binary) = 6 (decimal)
r-x = 101 (binary) = 5 (decimal)
r-- = 100 (binary) = 4 (decimal)
```

Exempelsiffran `644` är då detsamma som `rw-r--r--`, det vill säga att ägaren av filen har läs- och skrivrättigheter och alla andra har bara läsrättigheter. Vilket skrivs som följande kommando:

```
sebastian@mindator:~$ chmod 644 example.txt
```

Kom ihåg att det finns fortfarande ett inledande tecken framför rättigheterna som visade

på om filen var en vanlig fil eller mapp.

Det är onekligen ett smidigt sätt att ange rättigheter, men likväl väldigt svårt att komma ihåg. Därför har vi sammanställt en liten tabell över några av de vanligare rättigheterna som brukar anges.

Table 1: Ett urval av vanliga rättighetsinställningar (filer)

Värde	Rättigheter	Betydelse
`777`	`rwxrwxrwx`	Alla har alla rättigheter. Något som vi **inte** rekommenderar.
`755`	`rwxr-xr-x`	Filägaren har alla rättigheter och alla andra kan läsa och exekvera filen. Vanlig inställning för program som används av många, t.ex. `less` och `ls`.
`700`	`rwx------`	Filägaren har alla rättigheter, ingen annan har åtkomst till filen. En bra inställning om en fil behöver vara privat.
`666`	`rw-rw-rw-`	Alla har läs- och skrivrättigheter.
`644`	`rw-r--r--`	Filägaren har läs- och skrivrättigheter, alla andra har endast läsrättigheter. Bra om endast en person ska kunna redigera filen.
`600`	`rw-------`	Filägaren har läs- och skrivrättigheter, ingen annan har åtkomst till filen. Också en bra inställing för privata filer.

4 Rättigheter och mappar

Vi använder även `chmod` när vi vill ändra rättigheter på en mapp. Däremot finns det några skillnader i vad rättigheterna `r`, `w` och `x` innebär.

- `r` Användaren har möjlighet att lista innehållet, t.ex. via `ls`, **om** även `x` är satt.
- `w` Användaren kan skapa, radera och döpa om filer, **om** även `x` är satt.
- `x` Användaren kan gå ner eller öppna mappen, t.ex. via `cd`.

Nedan finner du några få exempel.

Table 2: Ett urval av vanliga rättighetsinställningar (mappar)

Värde	Rättigheter	Betydelse
`777`	`rwxrwxrwx`	Alla kan lista, skapa, radera och redigera filer. Något som vi **inte** rekommenderar.

Värde	Rättigheter	Betydelse
`755`	`rwxr-xr-x`	Mappägaren har alla rättigheter, alla andra kan lista innehållet **men inte** skapa eller radera filer. Vanligt för mappar du delar med andra.
`700`	`rwx------`	Mappägaren har alla rättigheter, ingen annan har åtkomst till mappen. En bra inställning om mappen behöver vara privat.

5 `sudo`: Kör ett kommando som en annan användare

Vissa saker vi gör på vår dator kräver att vi anger vårt lösenord, t.ex. om vi ska ändra någon systeminställning eller ibland när vi ska installera ett nytt program. Samma sak gäller när vi utföra vissa kommandon i en terminal. Däremot presenteras vi inte automatiskt med en ruta där vi kan fylla i vårt lösenord, vi behöver göra detta manuellt.

Kommandot `sudo` används för att tillfälligt bli den så kallade *superuser* (den som har administrationsrättigheter) och placeras framför det kommando du ska utföra.

```
sebastian@mindator:~$ sudo command
```

När ska vi använda detta då? Det beror på ditt operativsystem. Men, för att demonstrera detta kan vi försöka lista innehållet i mappen `/root`, som på operativsystemet Linux kräver att du är *superuser*.

```
sebastian@mindator:~$ ls /root
ls: cannot open directory '/root': permission denied
```

Meddelande `permission denied` berättar för oss att vi inte har rättigheterna att utföra kommandot, vilket är det tillfälle då vi kan använda oss av `sudo`, för att tillfälligt ge oss dessa rättigheter.

```
sebastian@mindator:~$ sudo ls /root
[sudo] password for sebastian:
```

Därefter fyller vi i vårt lösenord och kommandot kommer sedan att fungera som vanligt.

Observera dock att du ser varken lösenordet du fyller i eller hur många tecken du skrivit, det skrivs alltså i blindo (av säkerhetsskäl).

6 `chown`: Ändra en fils ägare

För att ändra vem som äger en fil använder vi kommandot `chown`, följt av den nya ägaren och vilken fil, eller filer, det gäller.

```
sebastian@mindator:~$ chown new_owner file...
```

Skulle jag vilja ändra ägaren av filen `names.txt` till `johan` hade vi skrivit följande:

```
sebastian@mindator:~$ sudo chown johan names.txt
```

Observera att detta kommando kräver att vi har administrationsrättigheter (*superuser privileges*), därför skriver vi `sudo` framför.

Kommandot fungerar på samma vis för mappar.

7 `chgrp`: Ändra grupptillhörighet för en fil

För att ändra vilken grupp som har tillgång till en fil använder vi kommandot `chgrp`, följt av den nya gruppen och vilken fil, eller filer, det gäller.

```
sebastian@mindator:~$ chgrp new_group file...
```

Detta kommando kräver inte administrationsrättigheter.

8 Filägare och grupper

Vi har genom modulerna stött på både termen filägare och att en grupp kan ha tillgång till en fil. Kort betyder detta att en fil alltid ägs av någon, t.ex. den som skapade filen, och att en grupp kan ha tillgång till en fil utöver ägaren, t.ex. systemadministratörer.

Vi nämner detta för din kännedom, det är sannolikt så att du kanske aldrig stöter på detta eller behöver använda dig av denna informationen. Men, skulle du hamna i en situation där du behöver göra något är det bra att känna till detta på denna nivå (terminalen).

9 Vad är ett “kommando”?

Fram till nu har vi använt oss av olika kommandon utan att kanske veta vad dessa egentligen är. Alla kommandon vi skriver i vår terminal är en av fyra olika typer:

1. **Ett exekverbart program** som `ls`, `less` eller `find`. Dessa program är antingen så kallade *compiled binaries* skrivna i programmeringsspråk som C och C++, eller så

kallade *scripts* skrivna i programmeringsspråk som Python, Ruby, JavaScript, etc.

2. **Ett inbyggt kommando i vårt skal.** Dessa kallar vi för *shell builtins* och Bash kommer med sina egna, till exempel kommandot `cd`.
3. **En skalfunktion.** En *shell function* är något som kan utföra flera kommandon och andra funktioner efter varandra. Däremot är detta något vi skapar på egen hand, något som vi inte kommer att gå igenom men är bra att känna till.
4. **Ett alias** från ett kommando till ett annat, där vi kan inkludera tillval m.m. Till exempel kunde vi hittat på kommandot `ll` som skulle vara ett *alias* till `ls -l`.

10 `type`: Få information om ett kommando

För att ta reda på vilken typ ett kommando är så använder vi oss av `type` följt av namnet på kommandot. Till exempel, låt oss ta reda på vad kommandot `ls` är.

```
sebastian@mindator:~$ type ls
ls is /usr/bin/ls
```

Beroende på hur din dator är inställd kan du få en annan utskrift än `ls is /usr/bin/ls`

Kommandot `ls` är alltså detsamma som filen, i detta exempel, `/usr/bin/ls`. Men vad är detta för en fil? Låt oss ta reda på det.

```
sebastian@mindator:~$ file /usr/bin/ls
/usr/bin/ls: ELF 64-bit LSB pie executable, ...
```

Det ger oss en hel den information, men kort och gott betyder det att kommandot `ls` är en så kallad *compiled binary* (en fil som består av en massa 1:or och 0:or), vilket är anledningen till att vi får en sökväg som svar från kommandot `type`.

Låt oss ta ännu ett exempel, denna gången med kommandot `cd`.

```
sebastian@mindator:~$ type cd
cd is a shell builtin
```

Kommandot `cd` är alltså ett inbyggt kommando i Bash. Nu har du möjlighet att på egen hand se vad olika kommandon du använder dig av är för någonting.

Den exakta utskriften från kommandot type kan skilja sig en aning mellan operativsystem.

11 Bonuskommando: `alias`

Att kunna skapa sina egna *alias* är väldigt användbart och på sistonde har vi använt kommandot `ls -l` flera gånger, ett ypperligt tillfälle att skapa ett alias, förslagsvis under namnet `ll`.

Kommandot `alias` är ett inbyggt kommando som ger oss möjlighet att skapa ett *alias* från ett kommando till ett annat och skrivs på följande vis:

```
sebastian@mindator:~$ alias ll="ls -l"
```

Observera att vi skriver det i formatet `nya_namnet="ditt kommando"`.

Därefter kan vi direkt börja använda oss av det.

```
sebastian@mindator:~$ ll
-rw-r--r--  1 sebastian sebastian    3526 Nov 10 22:16 example.txt
drwxr-xr-x  6 sebastian sebastian    4096 May  5 09:45 Desktop
drwxr-xr-x  3 sebastian sebastian    4096 Mar 27 18:06 Documents
...
```

Men vad är nu kommandot `ll`?

```
sebastian@mindator:~$ type ll
ll is alias to 'ls -l'
```

Precis som vi tidigare skrev, ett alias till `ls -l`.

11.1 Spara dina alias

En stor nackdel med våra alias är att dom inte sparas permanent, så när vi stänger ner vår terminal så finns dom inte kvar. Däremot kan dessa sparas i en speciell fil som Bash alltid läser in när skalet startas, för att vi ska kunna spara våra alias (och mycket annat)

Filen i fråga har vanligtvis namnet `.bashrc` och i denna kan vi enkelt lägga till våra alias (t.ex. i slutet av filen), dessa skrivs precis som kommandot tidigare.

Detta är däremot utanför omfånget för detta material, så detta lämnar vi som en övning åt dig (läs mer här).

Det finns en liten chans att du redan har denna filen i din hemmapp, om den finns kan du pröva att lägga till ett alias i slutet av filen och sedan starta om terminalen.

12 Miljön

Nu när vi vet vad ett kommando är kommer den intressanta följdfrågan, hur hittar datorn våra kommando? Hur kan den veta att kommandot `ls` befinner sig på platsen `/usr/bin/ls`?

Detta har att göra med hur vår miljö (*environment*) är inställd. Vår arbetsmiljö består bland annat av en mängd olika miljövariabler (*environment variables*), som styr allt från vilket skal vi använder oss av till vilka mappar datorn ska leta i efter kommandon som `ls`. Vår dator förhåller sig alltså till dessa variabler när den ska utföra olika uppgifter.

12.1 `printenv`: Skriv ut miljövariabler

För att se vilka miljövariabler vårt system har använder vi oss av kommandot `printenv`. Detta skriver ut respektive variabel i formatet `NAMN=värde` (namnen skrivs vanligtvis i versaler).

```
sebastian@mindator:~$ printenv
USER=sebastian
HOME=/home/sebastian
PATH=/usr/local/bin:/usr/bin:/usr
...
```

Som du säkert upptäckt finns det en hel del variabler, vi har därför valt ut några få i exemplet ovan som kan vara av intresse.

- `USER` Innehåller namnet på vår användare.
- `HOME` Innehåller sökvägen till vår hemmapp.
- `PATH` Innehåller flera sökvägar separerade med ett kolon. Det är i dessa mappar som vår dator letar efter kommandon.

En intressant observation kan vara att miljövariabeln `PATH` innehåller mappen `/usr/bin`, vilket är den plats där vi tidigare fick reda på att kommandot `ls` återfanns.

12.2 `echo`: Skriv ut innehållet i en variabel

I många fall vill vi endast kontrollera vad en av våra miljövariabler innehåller, då har vi kommandot `echo` till hands. Detta kommando skriver bara ut vad det som kommer efter kommandot, vilket kan låta lite underligt.

```
sebastian@mindator:~$ echo Hello World!
Hello World!
```

Men, om vi kommer ihåg att vi kan omdirigera saker kan vi faktiskt enkelt skriva ut en bit text till en fil med hjälp av detta kommandot.

```
sebastian@mindator:~$ echo Hello World! > test.txt
sebastian@mindator:~$ cat test.txt
Hello World!
```

Vi ville däremot skriva ut innehållet i en miljövariabel, och för att göra det måste vi veta hur vi kan komma åt dem. Detta görs genom att skriva ett dollartecken framför namnet på den miljövariabel vi vill komma åt, till exempel `$USER` eller `$HOME`.

```
sebastian@mindator:~$ echo $USER
sebastian
sebastian@mindator:~$ echo $HOME
/home/sebastian
```

Dessa kan användas nästan hur vi vill, låt oss säga att vi vill spara texten "Hello World!" i en fil med samma namn som vår användare, dvs. `sebastian.txt`.

```
sebastian@mindator:~$ echo Hello World! > $USER.txt
```

Eller säg att vi skulle vilja lista alla filerna i vår hemmapp.

```
sebastian@mindator:~$ ls -l $HOME
```

Det kan många gånger vara användbart att skapa egna miljövariabler, även om det bara är för stunden. Till exempel om vi har en lång sökväg vi kommer behöva upprepa mellan olika kommandon.

För att spara en ny miljövariabel skriver vi det i formatet `NAMN="värde"`. Till exempel:

```
sebastian@mindator:~$ WORKDIR="/my/long/path"
sebastian@mindator:~$ echo $WORKDIR
/my/long/path
```

Säg att vi i ett senare skede skulle vilja skriva ut alla mappar som finns i `WORKDIR`.

```
sebastian@mindator:~$ find $WORKDIR -type d
```

Ganska så smidigt, eller hur?

13 Övningar

I denna övning finns det ingen färdig övningsmapp att utgå från. Skapa en egen mapp någonstans i ditt filsystem och navigera sedan till den, så att vi har en plats att utföra några experiment på.

Det finns risk för att frågorna A1 och A2 inte går att genomföra på Windows pga. säkerhetsskäl, hoppa över dem isåfall.

- Skapa filen `a1.txt` (t.ex. med hjälp av `touch`) och ge den sedan rättigheterna `rw-r-xr-x`. Anteckna ner det kommando du skrev som **A1**.
- Om du försöker byta filägaren av `a1.txt` till "demo", vad får du då för ett meddelande? Anteckna ner meddelandet som **A2**.
- Vilken mapp befinner sig kommandot `cat` i? Anteckna ner detta som **A3**.
- Vad innehåller miljövariabeln `SHELL`? Anteckna ner vilket kommando du skrev samt vilken utskrift du fick som **A4**.