



Laboration – Hantera text

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:

- Lär dig hur du omdirigerar och kopplar standardkanalerna för inmatning, utmatning och felmeddelanden.
- Använd reguljära uttryck för att filtrera utdata från kommandon eller innehåll i filer.
- Visa stora filer eller resultat från kommandon med program för sidvisning, och titta på utvalda delar.

Command Line Pipes och Redirection

Normalt när du kör ett kommando visas utdata i terminalfönstret. Denna utdata (som också kallas en kanal) kallas *standardutdata*, och betecknas med termen **stdout**. Filbeskrivaren för denna kanal har numret **1**.

Standardfel (**stderr**) uppstår när ett fel inträffar under körningen av ett kommando; den har filbeskrivaren **2**. Felmeddelanden skickas också till terminalfönstret som standard.

I denna labb kommer du att använda tecken som omdirigerar utdata från standardutdata (**stdout**) och standardfel (**stderr**) till en fil eller till ett annat kommando istället för terminalskärmen.

Standardinmatning, **stdin**, brukar ges till ett kommando genom att du skriver på tangentbordet; den har filbeskrivaren **0**. Genom att omdirigera standardinmatning kan dock även filer användas som **stdin**.

Använd omdirigeringsymbolen **>** tillsammans med **echo**-kommandot för att omdirigera utdata från den normala **stdout** (terminalen) till en fil. **cat**-kommandot kan användas för att visa filens innehåll och kommer att användas i detta exempel för att verifiera omdirigerad utdata till filen. Skriv följande:

```
echo "Hello World"
echo "Hello World" > mymessage
cat mymessage
```

Din utdata ska likna följande:

```
sysadmin@localhost:~$ echo "Hello World"
Hello World
sysadmin@localhost:~$ echo "Hello World" > mymessage
sysadmin@localhost:~$ cat mymessage
Hello World
```



Det första kommandot skriver ut meddelandet (**stdout**) till terminalen.

Det andra kommandot omdirigerar utdata; istället för att skicka utdata till terminalen, skickas den till en fil som heter **mymessage**.

Det sista kommandot visar innehållet i filen **mymessage**.

När du använder symbolen **>** för att omdirigera **stdout**, förstörs innehållet i filen först. Skriv följande kommandon för att se ett exempel:

```
echo "Greetings" > mymessage
cat mymessage
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ echo "Greetings" > mymessage
sysadmin@localhost:~$ cat mymessage
Greetings
sysadmin@localhost:~$
```

Observera att om du använder en omdirigeringsymbol så skrivs en befintlig fil över. Detta kallas att "klobba" (clobbering) en fil.

Du kan undvika att skriva över en fil ("klobba") genom att använda **>>** istället för **>**. Med **>>** lägger du till ("appenderar") till en fil. Kör följande kommandon för att se ett exempel på detta:

```
cat mymessage
echo "How are you?" >> mymessage
cat mymessage
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ cat mymessage
Greetings
sysadmin@localhost:~$ echo "How are you?" >> mymessage
sysadmin@localhost:~$ cat mymessage
Greetings
How are you?
```

Observera att när du använder **>>** så bevaras allt befintligt innehåll och den nya texten läggs till i slutet av filen.

Kommandoverktyget **find** är ett bra exempel för att visa hur felutmatning (**stderr**) fungerar. Det här mycket flexibla kommandot låter dig söka med en mängd olika alternativ, som filnamn, storlek, datum, typ och behörigheter.

Kommandoverktyget **find** startar sökningen i den angivna katalogen och söker rekursivt i alla underkataloger. Till exempel, för att söka efter filer i din hemkatalog som innehåller namnet "**bash**":

```
find ~ -name "*bash"
```

Din utdata bör likna följande:



```
sysadmin@localhost:~$ find ~ -name "*bash*"
/home/sysadmin/.bash_logout
/home/sysadmin/.bashrc
sysadmin@localhost:~$
```

Kom ihåg att `~` används för att representera din hemkatalog.

Kör nu följande kommando och observera resultatet:

```
find /etc -name hosts
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ find /etc -name hosts
/etc/hosts
find: '/etc/ssl/private': Permission denied
sysadmin@localhost:~$
```

Observera felmeddelandet som visar att du inte har behörighet att komma åt vissa filer eller kataloger. Detta beror på att du som vanlig användare inte har rätt att "titta in" i vissa kataloger. Den här typen av felmeddelanden skickas till felutmatningen (`stderr`), inte till standardutmatningen (`stdout`).

Kommandoverktyget `find` är mer avancerat än vad som täcks i den här kursen. Syftet med att använda kommandot är att visa skillnaden mellan standardutmatning (`stdout`) och felutmatning (`stderr`).

För att omdirigera felutmatning (felmeddelanden) till en fil, kör följande kommando:

```
find /etc -name hosts 2> err.txt
cat err.txt
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ find /etc -name hosts 2> err.txt
/etc/hosts
sysadmin@localhost:~$ cat err.txt
find: `/etc/ssl/private': Permission denied
```

Kom ihåg att filbeskrivaren för felutmatning är nummer `2`, så den används tillsammans med `>`-symbolen för att omdirigera felutmatning till en fil som heter `err.txt`. Notera att `1>` är samma som `>`.

Det föregående exemplet visar varför det är viktigt att kunna omdirigera utmatning. Om du vill "ignorera" de fel som kommandot `find` visar kan du omdirigera dessa meddelanden till en fil och titta på dem senare, vilket gör det lättare att fokusera på resten av utdata från kommandot.

Du kan också omdirigera `stdout` och `stderr` till två separata filer.

```
find /etc -name hosts > std.out 2> std.err
cat std.err
cat std.out
```



Din utdata bör likna följande:

```
sysadmin@localhost:~$ find /etc -name hosts > std.out 2> std.err
sysadmin@localhost:~$ cat std.err
find: ` /etc/ssl/private': Permission denied
sysadmin@localhost:~$ cat std.out
/etc/hosts
```

Observera att ett mellanslag är tillåtet, men inte nödvändigt, efter `>`-symbolen för omdirigering.

För att omdirigera både standardutmatning (`stdout`) och standardfel (`stderr`) till en och samma fil, omdirigerar du först `stdout` till filen och sedan `stderr` till samma fil genom att använda notation `2>&1`.

```
find /etc -name hosts > find.out 2>&1
cat find.out
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ find /etc -name hosts > find.out 2>&1
sysadmin@localhost:~$ cat find.out
/etc/hosts
find: '/etc/ssl/private': Permission denied
sysadmin@localhost:~$
```

Delen `2>&1` i kommandot betyder "skicka `stderr` (kanal 2) till samma plats där `stdout` (kanal 1) hamnar".

Standard input (`stdin`) kan också omdirigeras. Normalt kommer `stdin` från tangentbordet, men ibland vill du att den ska komma från en fil istället. Till exempel accepterar kommandot `tr` endast data från `stdin`, aldrig från ett filnamn som anges som argument. Det här är smidigt när du vill göra något som att skriva om data till versaler direkt från tangentbordet (Obs: Tryck **Control + d** för att signalera till `tr`-kommandot att sluta behandla standard input):

```
tr a-z A-Z
this is interesting
how do I stop this?
^D
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ tr a-z A-Z
this is interesting
THIS IS INTERESTING
how do I stop this?
HOW DO I STOP THIS?
sysadmin@localhost:~$
```

Obs : `^D` symboliserar **Control + d**



Kommandot `tr` tar emot tangentbordsinmatning (`stdin`), översätter tecknen och skickar sedan utdata till `stdout`. För att skapa en fil med enbart små bokstäver, kör följande:

```
tr A-Z a-z > myfile
Wow, I SEE NOW
This WORKS!
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ tr A-Z a-z > myfile
Wow, I SEE NOW
This WORKS!
sysadmin@localhost:~$
```

Tryck på **Enter** så att markören hamnar på raden under "Detta fungerar!", använd sedan **Control + d** för att avsluta inmatningen. För att kontrollera att du har skapat filen, kör följande kommando:

```
cat myfile
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ cat myfile
wow, i see now
this works!
sysadmin@localhost:~$
```

Kör följande kommandon för att använda kommandot `tr` genom att omdirigera `stdin` från en fil:

```
cat myfile
tr a-z A-Z < myfile
```

Din utdata bör likna följande:

```
sysadmin@localhost:~$ cat myfile
wow, i see now
this works!
sysadmin@localhost:~$ tr a-z A-Z < myfile
WOW, I SEE NOW
THIS WORKS!
sysadmin@localhost:~$
```

En annan vanlig form av omdirigering är att ta utdata från ett kommando och skicka det som indata till ett annat kommando. Till exempel kan utdata från vissa kommandon vara enorm, vilket gör att det scrollar förbi på skärmen så snabbt att det är svårt att läsa. Kör följande kommando för att ta utdata från kommandot `ls` och skicka det vidare till kommandot `more`, som visar en sida data åt gången:

```
ls -l /etc | more
```

Din utdata bör likna följande:



Du måste trycka på **mellanslag** för att fortsätta, eller så kan du även trycka på **CTRL + c** för att avbryta listningen.

```
sysadmin@localhost:~$ ls -l /etc | more
total 372
-rw-r--r-- 1 root root 2981 Jan 28 2015 adduser.conf
-rw-r--r-- 1 root root 10 Jan 28 2015 adjtime
drwxr-xr-x 1 root root 900 Jan 29 2015 alternatives
drwxr-xr-x 1 root root 114 Jan 29 2015 apparmor.d
drwxr-xr-x 1 root root 168 Oct 1 2014 apt
-rw-r--r-- 1 root root 2076 Apr 3 2012 bash.bashrc
drwxr-xr-x 1 root root 72 Jan 28 2015 bash_completion.d
drwxr-sr-x 1 root bind 342 Jan 29 2015 bind
-rw-r--r-- 1 root root 356 Apr 19 2012 bindresvport.blacklist
-rw-r--r-- 1 root root 321 Mar 30 2012 blkid.conf
-rw-r--r-- 1 root root 2969 Mar 15 2012 debconf.conf
--More--
```

Kommandot **cut** är användbart för att extrahera fält från filer som antingen är avgränsade med ett tecken, som kolon **:** i **/etc/passwd**, eller som har en fast bredd. Det kommer att användas i de nästa exemplen eftersom det ofta ger mycket utdata som vi kan använda för att visa hur tecknet **|** fungerar.

I följande exempel kommer du att använda ett kommando som heter **cut** för att extrahera alla användarnamn från en databas som heter **/etc/passwd** (en fil som innehåller information om användarkonton). Börja med att köra **cut**-kommandot på egen hand:

```
cut -d: -f1 /etc/passwd
```

En del av kommandots utdata visas i grafiken nedan.

```
sysadmin@localhost:~$ cut -d: -f1 /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
```

Utdata i det föregående exemplet var osorterad och scrollade bort från skärmen. I nästa steg ska du ta utdata från **cut**-kommandot och skicka den vidare till **sort**-kommandot för att få lite ordning på utdata:

```
cut -d: -f1 /etc/passwd | sort
```

En del av kommandots utdata visas i grafiken nedan.



```
sysadmin@localhost:~$ cut -d: -f1 /etc/passwd | sort
backup
bin
bind
daemon
games
gnats
irc
libuuid
list
```

Nu är utdata sorterad, men den scroller fortfarande bort från skärmen. Skicka istället utdatan från `sort`-kommandot vidare till `more`-kommandot för att lösa detta problem:

```
cut -d: -f1 /etc/passwd | sort | more
```

```
sysadmin@localhost:~$ cut -d: -f1 /etc/passwd | sort | more
backup
bin
bind
daemon
games
gnats
irc
libuuid
list
lp
--More--
```

Läsa stora textfiler

Stora textfiler kan visserligen visas med kommandot `cat`, men det är krångligt att bläddra tillbaka mot början av filen. Dessutom går det inte att visa riktigt stora filer på det sättet, eftersom terminalfönstret bara lagrar ett visst antal rader av utdata i minnet.

Om du använder kommandona `more` eller `less` kan du visa data en "sida" eller rad i taget. Dessa så kallade pager-kommandon tillåter även andra sätt att navigera och söka, vilket kommer att visas i det här avsnittet.

Obs

Exempel visas med både `more` och `less`. För det mesta fungerar kommandona likadant, men `less` är mer avancerat och har fler funktioner. `more` är ändå viktigt att känna till eftersom vissa Linux-distributioner saknar `less`, medan alla har `more`.

Om du inte är intresserad av att visa hela filen eller all utdata från ett kommando finns det flera kommandon för att filtrera innehållet. I den här delen får du lära dig hur `head` och `tail` används för att plocka ut information från början eller slutet av utdata från ett kommando eller filinnehåll.

`/etc/passwd` är troligen för stor för att visas på skärmen utan att du måste scrollera. För att se ett exempel, använd `cat`-kommandot för att visa hela innehållet i `/etc/passwd`:

```
cat /etc/passwd
```



Din utdata bör likna detta:

```
sysadmin@localhost:~$ cat /etc/passwd
```

```
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:103:./home/syslog:/bin/false
bind:x:102:105:./var/cache/bind:/bin/false
sshd:x:103:65534:./var/run/sshd:/usr/sbin/nologin
operator:x:1000:37:./root:/bin/sh
sysadmin:x:1001:1001:System
Administrator,,,:/home/sysadmin:/bin/bash
svsadmin@localhost:~$
```

Använd kommandot `more` för att visa hela innehållet i `/etc/passwd`-filen:

```
more /etc/passwd
```

Din utdata bör likna detta:

```
sysadmin@localhost:~$ more /etc/passwd
```

```
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101:./var/lib/libuuid:/bin/sh
syslog:x:101:103:./home/syslog:/bin/false
bind:x:102:105:./var/cache/bind:/bin/false
sshd:x:103:65534:./var/run/sshd:/usr/sbin/nologin
operator:x:1000:37:./root:/bin/sh
--More-- (92%)
```

Obs

`--More-- (92%)` betyder att du använder `more`-kommandot och att du är 92% igenom den aktuella datan

När du är inne i kommandot `more` kan du visa hjälpskärmen genom att trycka på tangenten `h`:

```
h
```

Din utdata bör likna detta:



```
Most commands optionally preceded by integer argument k. Defaults in brackets
Star (*) indicates argument becomes new default.
```

```
-----
<space>          Display next k lines of text [current screen size]
z                Display next k lines of text [current screen size]*
<return>        Display next k lines of text [1]*
d or ctrl-D     Scroll k lines [current scroll size, initially 11]*
q or Q or <interrupt> Exit from more
s                Skip forward k lines of text [1]
f                Skip forward k screenfuls of text [1]
b or ctrl-B     Skip backwards k screenfuls of text [1]
'                Go to place where previous search started
=                Display current line number
/<regular expression> Search for kth occurrence of regular expression [1]
n                Search for kth occurrence of last r.e [1]
!<cmd> or :!<cmd> Execute <cmd> in a subshell
v                Start up /usr/bin/vi at current line
ctrl-L          Redraw screen
:n              Go to kth next file [1]
:p              Go to kth previous file [1]
:f              Display current file name and line number
.              Repeat previous command
-----
```

```
--More-- (92%)
```

Tryck på **mellanslagstangenten** för att visa resten av dokumentet:

```
<SPACE>
```

I nästa exempel får du lära dig hur du söker i ett dokument med antingen **more**- eller **less**-kommandot.

För att söka efter ett mönster med både **more** och **less**, skriver du snedstreck **/** följt av mönstret du vill hitta. Om en träff finns så bläddrar skärmen till den första matchningen. För att gå vidare till nästa träff, tryck på **n**. Med **less**-kommandot kan du också gå bakåt till tidigare träffar genom att trycka på **N** (stort N).

Använd **less**-kommandot för att visa hela innehållet i **/etc/passwd**-filen. Sök därefter efter ordet **bin**, använd **n** för att gå framåt och **N** för att gå bakåt. Avsluta slutligen **less** genom att skriva **q**:

```
less /etc/passwd
```

```
/bin
```

```
nnnNNNq
```

Viktigt

Till skillnad från **more**-kommandot som automatiskt avslutas när du når slutet av filen, måste du trycka på en avslutningstangent som **q** för att avsluta **less**-programmet.

Du kan använda **head**-kommandot för att visa den första delen av en fil. Som standard visar **head** de första tio raderna:

```
head /etc/passwd
```



Din utdata bör likna detta:

```
sysadmin@localhost:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
sysadmin@localhost:~$
```

Använd `tail`-kommandot för att visa de sista tio raderna i `/etc/passwd`-filen:

```
tail /etc/passwd
```

Din utdata bör likna detta:

```
sysadmin@localhost:~$ tail /etc/passwd
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/system
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd/resolve
syslog:x:103:106::/home/syslog:/usr/sbin/nologin
messagebus:x:104:107::/nonexistent:/usr/sbin/nologin
bind:x:105:110::/var/cache/bind:/usr/sbin/nologin
sshd:x:106:65534::/run/sshd:/usr/sbin/nologin
operator:x:1000:37::/root:/bin/sh
sysadmin:x:1001:1001:System Administrator,,,:/home/sysadmin:/bin/b
sysadmin@localhost:~$
```

Använd kommandot `head` för att visa de två första raderna i `/etc/passwd`-filen:

```
head -2 /etc/passwd
```

Din utdata bör likna detta:

```
sysadmin@localhost:~$ head -2 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
sysadmin@localhost:~$
```

Kör följande kommandorad för att skicka utdata från kommandot `ls` vidare till kommandot `tail`, så att de fem sista filnamnen i katalogen `/etc` visas:

```
ls /etc | tail -5
```

Din utdata bör likna detta:



```
sysadmin@localhost:~$ ls /etc | tail -5
updatedb.conf
vim
vtrgb
wgetrc
xdg
sysadmin@localhost:~$
```

Som du har sett skriver både `head`- och `tail`-kommandona ut tio rader som standard. Du kan också använda flaggan `-#` (eller du kan använda flaggan `-n #`, där `#` är antalet rader som ska visas). Båda kommandona kan användas för att läsa standardinput från en pipe som tar emot utdata från ett kommando.

Du har också sett var `head`- och `tail`-kommandona skiljer sig åt: `head` börjar räkna rader från början av datan, medan `tail` räknar rader från slutet av datan. Det finns ytterligare skillnader mellan dessa två kommandon, vilket visas i de kommande övningarna.

Ett annat sätt att ange hur många rader som ska visas med `head`-kommandot är att använda flaggan `-n -#`, där `#` är antalet rader som räknas från slutet av utdata för att exkluderas. Lägg märke till minustecknet `-` framför siffran. Om till exempel `/etc/passwd` innehåller 27 rader, så kommer följande kommando att visa raderna 1–7, alltså de första sju raderna, och hoppa över de sista tjugo:

```
head -n -20 /etc/passwd
```

```
sysadmin@localhost:~$ head -n -20 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
sysadmin@localhost:~$
```

Söka i texter med reguljära uttryck

I den här uppgiften ska du använda kommandon från `grep`-familjen tillsammans med reguljära uttryck för att söka efter en specifik teckensträng i en datamängd (till exempel en textfil).

Kommandot `grep` använder *grundläggande reguljära uttryck*, speciella tecken som jokertecken för att matcha mönster i data. `grep`-kommandot returnerar hela raden som innehåller det matchande mönstret.

Flaggan `-E` till `grep`-kommandot kan användas för att göra sökningar med *utökade reguljära uttryck*, alltså mer avancerade reguljära uttryck. Ett annat sätt att använda utökade reguljära uttryck är att använda kommandot `egrep`.

Kommandot `fgrep` används för att matcha bokstavliga tecken och ignorerar den speciella betydelsen av reguljära uttryckstecken.

Det enklaste sättet att använda `grep` är att söka efter en given teckensträng, till exempel `sshd` i filen `/etc/passwd`. `grep`-kommandot skriver ut hela raden som innehåller träffen:



```
cd /etc
grep sshd passwd
```

Din utdata bör likna detta:

```
sysadmin@localhost:~$ cd /etc
sysadmin@localhost:/etc$ grep sshd passwd
sshd:x:106:65534:./run/sshd:/usr/sbin/nologin
sysadmin@localhost:/etc$
```

Reguljära uttryck är "giriga" i den bemärkelsen att de matchar varje enskilt förekomst av det angivna mönstret:

```
grep root passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ grep root passwd
root:x:0:0:root:/root:/bin/bash
operator:x:1000:37:./root:/bin/sh
sysadmin@localhost:/etc$
```

Observera att de röda markeringarna visar exakt vad som matchades. Du kan också se att alla förekomster av `root` matchades på varje rad.

För att begränsa utdata kan du använda reguljära uttryck för att ange ett mer precist mönster. Till exempel kan cirkumflextecknet `^` användas för att matcha ett mönster i början av en rad; så när du kör följande kommandorad kommer bara de rader som börjar med `root` att matchas och visas:

```
grep '^root' passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ grep '^root' passwd
root:x:0:0:root:/root:/bin/bash
sysadmin@localhost:/etc$
```

Observera att det finns två ytterligare förekomster av ordet `root`, men endast den som står i början av raden matchas (visas i rött).

Bästa praxis

Använd enkla citattecken (inte dubbla) runt reguljära uttryck för att förhindra att skalprogrammet försöker tolka dem.

Matcha mönstret `sync` var som helst på en rad:

```
grep 'sync' passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ grep 'sync' passwd
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:/etc$
```



Använd symbolen `$` för att matcha mönstret `sync` i slutet av en rad:

```
grep 'sync$' passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ grep 'sync$' passwd
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:/etc$
```

Kommandot i föregående steg matchade varje förekomst; det andra matchar bara förekomsten i slutet av raden.

Använd punkten `.` för att matcha vilken enskild tecken som helst. Till exempel, kör följande kommando för att matcha vilket tecken som helst följt av ett `y`:

```
grep '.y' passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ grep '.y' passwd
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/systemd
systemd-resolve:x:102:103:systemd
sysadmin@localhost:/etc$
```

Pipe-tecknet, `|`, eller "alternativoperatör", fungerar som en "eller"-operator. Till exempel, kör följande kommando för att försöka matcha antingen `sshd`, `root` eller `operator`:

```
grep 'sshd|root|operator' passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ grep 'sshd|root|operator' passwd
sysadmin@localhost:/etc$
```

Observera att kommandot `grep` inte känner igen pipe-tecknet som en alternativoperator som standard. Grep-kommandot behandlar faktiskt pipe-tecknet som ett vanligt tecken i mönstret som ska matchas. Om du använder `grep -E` eller `egrep` kan du använda utökade reguljära uttryck, inklusive alternation.

Använd växeln `-E` för att låta `grep` arbeta i utökat läge så att den kan känna igen alternativoperatör:

```
grep -E 'sshd|root|operator' passwd
```

Din utdata bör likna följande:



```
sysadmin@localhost:/etc$ grep -E 'sshd|root|operator' passwd
root:x:0:0:root:/root:/bin/bash
sshd:x:103:65534:/:/var/run/ssh:/usr/sbin/nologin
operator:x:1000:37:/:/root:/bin/sh
sysadmin@localhost:/etc$
```

Använd ett annat utökat reguljärt uttryck, den här gången med `egrep` och alternation inom en grupp för att matcha ett mönster. Strängarna `nob` och `non` kommer att matchas:

```
egrep 'no(b|n)' passwd
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ egrep 'no(b|n)' passwd
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:/:/nonexistent:/usr/sbin/nologin
messagebus:x:104:107:/:/nonexistent:/usr/sbin/nologin
sysadmin@localhost:/etc$
```

Observera

Parenteserna, `()`, användes för att begränsa "räckvidden" för `|`-tecknet. Utan dem hade ett mönster som `nob|n` betytt "matcha antingen `nob` eller `n`".

Tecknen `[]` kan också användas för att matcha ett enskilt tecken. Till skillnad från punkttecknet `.`, så används `[]` för att specificera exakt vilket tecken du vill matcha. Om du till exempel vill matcha en siffra kan du ange `[0-9]`. Kör följande kommando för att se hur det fungerar:

```
head passwd | grep '[0-9]'
```

Din utdata bör likna följande:

```
sysadmin@localhost:/etc$ head passwd | grep '[0-9]'
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
sysadmin@localhost:/etc$
```

Observera

Kommandot `head` användes för att begränsa utdata från `grep`-kommandot.

Anta att du vill söka efter ett mönster som innehåller en sekvens av tre siffror. Du kan använda `{ }`-tecknen tillsammans med ett tal för att ange att du vill upprepa ett mönster ett visst antal gånger; till exempel: `{3}`. Användningen av den numeriska kvalificeraren kräver att `grep` körs i utökat läge:

```
grep -E '[0-9]{3}' passwd
```

Din utdata bör likna följande:



```
sysadmin@localhost:/etc$ grep -E '[0-9]{3}' passwd
sync:x:4:65534:sync:/bin:/bin/sync
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534:./nonexistent:/usr/sbin/nologin
systemd-network:x:101:102:systemd Network Management,,,:/run/system
systemd-resolve:x:102:103:systemd Resolver,,,:/run/systemd/resolve
syslog:x:103:106:./home/syslog:/usr/sbin/nologin
messagebus:x:104:107:./nonexistent:/usr/sbin/nologin
bind:x:105:110:./var/cache/bind:/usr/sbin/nologin
sshd:x:106:65534:./run/sshd:/usr/sbin/nologin
operator:x:1000:37:./root:/bin/sh
sysadmin:x:1001:1001:System Administrator,,,:/home/sysadmin:/bin
sysadmin@localhost:/etc$
```