



# Laboration – Processhantering och loggar

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:

- Undersök hur **/proc**-filsystemet används av kärnan
- Använd kommandot **ps** för att visa information om processer
- Lär dig hantera processer genom att starta, stoppa och återuppta dem
- Visa loggfiler
- Hantera möjligheten att ladda delade bibliotek

## Kärnan och /proc

I den här uppgiften ska du utforska katalogen **/proc** och kommandon som kommunicerar med Linux-kärnan. Katalogen **/proc** ser ut som en vanlig katalog, till exempel **/usr** eller **/etc**, men den är faktiskt inte det. Till skillnad från **/usr**- eller **/etc**-katalogerna, som vanligtvis skrivs till en hårddisk, är **/proc** en *pseudo-filstruktur* som hålls i datorns minne.

### /proc-katalogen

**/proc**-katalogen innehåller en underkatalog för varje körande process i systemet. Program som **ps** och **top** hämtar information om aktiva processer från dessa kataloger. **/proc** innehåller också information om operativsystemet och dess hårdvara i filer som **/proc/cpuinfo**, **/proc/meminfo** och **/proc/devices**.

Underkatalogen **/proc/sys** innehåller *pseudo-filer* som kan användas för att ändra inställningar för den aktiva kärnan. Eftersom dessa filer inte är "riktiga" filer bör man inte använda en textredigerare för att ändra dem; istället ska man använda kommandona **echo** eller **sysctl** för att skriva över innehållet i filerna. Av samma anledning ska du heller inte försöka visa filerna i en editor, utan använd istället **cat** eller **sysctl**.

För permanenta konfigurationsändringar använder kärnan filen **/etc/sysctl.conf**. Vanligtvis används den här filen av kärnan för att göra ändringar i **/proc**-filerna när systemet startar upp.

I den här uppgiften kommer du att undersöka några av filerna i katalogen **/proc**.

```
su
ls /proc
```

Din utskrift ska likna följande:



```
sysadmin@localhost:~$ su
Password:
root@localhost:~# ls /proc
1          cmdline   ioports   modules    sys
10         consoles  irq        mounts     sysrq-trigger
14         cpuinfo   kallsyms  mpt        sysvipc
16         crypto    kcore     mtrr       thread-self
26         devices  key-users net         timer_list
47         diskstats keys       pagetypeinfo tty
67         dma       kmsg      partitions uptime
68         driver    kpagecgroup pressure    version
7          dynamic_debug kpagecount schedstat  version_signature
79         execdomains kpageflags scsi       vmallocinfo
acpi      fb         loadavg   self       vmstat
bootconfig filesystems locks     slabinfo   zoneinfo
buddyinfo fs         mdstat    softirqs
bus       interrupts meminfo   stat
cgroups  iomem     misc      swaps
root@localhost:~#
```

Kom ihåg att katalogerna med siffror som namn representerar aktiva processer i systemet. Den första processen är alltid `/sbin/init`, så katalogen `/proc/1` kommer att innehålla filer med information om den körande `init`-processen.

Filen `cmdline` i processkatalogen (`/proc/1/cmdline`, till exempel) visar kommandot som kördes. I vilken ordning andra processer startas varierar mycket från system till system. Eftersom innehållet i denna fil inte innehåller någon radbrytning, används kommandot `echo` för att få prompten att hamna på en ny rad.

Använd `cat` och sedan `ps` för att visa information om processen `/sbin/init` (Process-ID, PID, är 1):

```
cat /proc/1/cmdline; echo
ps -p 1
```

Din utskrift ska likna följande:

```
root@localhost:~# cat /proc/1/cmdline; echo
/sbin/init
```

#### Obs

`echo`-kommandot i det här exemplet körs direkt efter `cat`-kommandot. Eftersom det inte har något argument används det bara för att sätta nästa kommandoprompt på en ny rad. Kör `cat`-kommandot separat för att se skillnaden.

```
root@localhost:~# ps -p 1
PID TTY          TIME CMD
  1 pts/0        00:00:00 init
```

De andra filerna i `/proc`-katalogen innehåller information om operativsystemet. Följande uppgifter används för att visa och ändra dessa filer.

Visa filen `/proc/cmdline` för att se vilka argument som skickades till kärnan vid uppstart:

```
cat /proc/cmdline
```



Utdata från kommandot bör se ut ungefär så här:

```
root@localhost:~# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-5.15.0-94-generic root=UUID=130ed84e-
7f39-4532-b2d4-c1e
6fc0b2111 ro noibrs noibpb nopti nospectre_v2 nospectre_v1
l1tf=off nospec_store
_bypass_disable no_stf_barrier mds=off mitigations=off
```

Den här utskriften innehåller all information, såsom kommandoradsparametrar, särskilda instruktioner med mera, som skickades till kärnan när den startades första gången.

## Hantera processer

Från terminalen, skriv följande kommando:

```
ping localhost > /dev/null
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# ping localhost > /dev/null
—
```

Utdata från `ping`-kommandot omdirigeras till filen `/dev/null` (som ofta kallas "bit bucket").

### Obs

Notera att terminalen verkar "hänga" vid det här kommandot. Detta beror på att kommandot körs i förgrunden. En process som körs i förgrunden gör att du inte kan använda skalet förrän processen är klar. Om processen istället körs i bakgrunden kan du fortsätta använda skalet och köra andra kommandon.

Systemet fortsätter att pinga tills processen avslutas eller pausas av användaren.

Avsluta förgrundsprocessen genom att trycka på **Ctrl + C**.

```
root@localhost:~# ping localhost > /dev/null

^C
root@localhost:~#
```

Nästa steg är att starta samma process i bakgrunden, skriv då:

```
ping localhost > /dev/null &
```

Din utskrift bör likna det här:

```
root@localhost:~# ping localhost > /dev/null &
[1] 107
```

Genom att lägga till ett `&`-tecken i slutet av kommandot startas processen i bakgrunden, vilket gör att du behåller kontrollen över terminalen.



### Tänk på detta

Ett smidigare sätt att skriva ovanstående kommando är att använda kommandots historik. Du kan trycka på **Uppåtpil** ↑ på tangentbordet, lägga till ett **mellanslag** och **&** i slutet av kommandot, och sedan trycka på **Enter** . Det sparar tid när du ska skriva liknande kommandon.

Observera att det tidigare kommandot returnerar följande information:

```
[1] 107
```

Det här betyder att processen har ett jobbnr 1 (vilket visas med utskriften **[1]**) och ett Process-ID (PID) på **107**. Varje terminal/shell har unika jobbnr. PID är systemomfattande; varje process har ett unikt ID-nummer.

Den här informationen är viktig när du ska göra vissa processhanteringar, till exempel stoppa processer eller ändra deras prioritet.

Ditt process-ID kommer troligen att vara annorlunda än det som visas i exemplet.

För att se vilka kommandon som körs i den aktuella terminalen, skriv följande kommando:

```
jobs
```

Din utskrift bör likna det här:

```
root@localhost:~# jobs
[1]+  Running                  ping localhost > /dev/null &
```

Nästa steg är att starta ytterligare ett **ping**-kommando i bakgrunden genom att skriva följande:

```
ping localhost > /dev/null &
```

Din utskrift bör likna det här:

```
root@localhost:~# ping localhost > /dev/null &
[2] 108
```

Notera att det här nya kommandot har ett annat jobbnr och process-ID.

Nu ska det finnas två ping-kommandon som körs i bakgrunden. För att kontrollera detta, skriv kommandot **jobs** igen:

```
jobs
```

Din utskrift bör likna det här:

```
root@localhost:~# jobs
[1]-  Running                  ping localhost > /dev/null &
[2]+  Running                  ping localhost > /dev/null &
```

När du har kontrollerat att två **ping**-kommandon körs, ta fram det första kommandot till förgrunden genom att skriva följande:

```
fg %1
```

Din utskrift bör likna det här:



```
root@localhost:~# fg %1
ping localhost > /dev/null
```

Observera att `ping`-kommandot återigen har tagit kontroll över terminalen. För att pausa processen och få tillbaka kontrollen över terminalen, tryck **Ctrl - Z** :

```
root@localhost:~# fg %1
ping localhost > /dev/null
^Z
[1]+  Stopped                  ping localhost > /dev/null
```

För att låta processen fortsätta att köra i bakgrunden, kör följande kommando:

```
bg %1
```

Din utskrift bör likna det här:

```
root@localhost:~# bg %1
[1]+ ping localhost > /dev/null &
```

Kör kommandot `jobs` igen för att kontrollera att två processer körs:

```
jobs
```

Din utskrift bör likna det här:

```
root@localhost:~# jobs
[1]-  Running                  ping localhost > /dev/null &
[2]+  Running                  ping localhost > /dev/null &
```

Nästa steg är att starta ytterligare ett `ping`-kommando genom att skriva följande:

```
ping localhost > /dev/null &
```

Din utskrift bör likna det här:

```
root@localhost:~# ping localhost > /dev/null &
[3] 110
```

Kör kommandot `jobs` igen för att kontrollera att tre processer körs:

```
jobs
```

```
root@localhost:~# jobs
[1]  Running                  ping localhost > /dev/null &
[2]- Running                  ping localhost > /dev/null &
[3]+ Running                  ping localhost > /dev/null &
```

Använd jobbnumret för att stoppa det senaste ping-kommandot med kommandot `kill` och kontrollera sedan att det stoppats genom att köra `jobs`-kommandot:

```
kill %3
```

```
jobs
```

Din utskrift bör likna det här:



```
root@localhost:~# kill %3
root@localhost:~# jobs
[1]  Running                ping localhost > /dev/null &
[2]-  Running                ping localhost > /dev/null &
[3]+  Terminated           ping localhost > /dev/null
```

Slutligen kan du stoppa alla `ping`-kommandon med kommandot `killall`. När du har kört `killall`-kommandot, vänta några ögonblick och kör sedan `jobs`-kommandot för att kontrollera att alla processer har stoppats:

```
killall ping
jobs
```

Din utskrift bör likna det här:

```
root@localhost:~# killall ping
root@localhost:~# jobs
[1]-  Terminated           ping localhost > /dev/null
[2]+  Terminated           ping localhost > /dev/null
```

## Använda `top` för att visa processer

I den här uppgiften ska du använda kommandot `top` för att jobba med processer. Som standard sorterar `top`-programmet processerna i fallande ordning efter CPU-användning i procent, så de program som använder mest CPU hamnar högst upp i listan.

Från terminalfönstret, skriv in följande kommandon:

```
ping localhost > /dev/null &
ping localhost > /dev/null &
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# ping localhost > /dev/null &
[1] 122
root@localhost:~# ping localhost > /dev/null &
[2] 123
```

Lägg märke till de PID:er som skrivs ut av ovanstående kommandon! De kommer att vara annorlunda än i de exempel vi ger här. Du kommer att använda PID:erna i kommande steg.

Nästa steg är att starta kommandot `top` genom att skriva följande i terminalen:

```
top
```

```
root@localhost:~# top
```



Din utskrift bör se ut ungefär så här

```
top - 19:32:10 up 15 days, 11:09, 1 user, load average: 7.52, 4.62, 3.01
Tasks: 12 total, 1 running, 11 sleeping, 0 stopped, 0 zombie
%Cpu(s): 7.6 us, 10.0 sy, 0.0 ni, 81.6 id, 0.0 wa, 0.0 hi, 0.8 si, 0.0 st
KiB Mem : 13201464+total, 50793780 free, 55954760 used, 25266100 buff/cache
KiB Swap: 13419622+total, 13401251+free, 183704 used. 75423536 avail Mem

  PID USER      PR    NI  VIRT  RES  SHR  S  %CPU  %MEM     TIME+  COMMAND
    1 root        20     0   18376 3092 2816 S   0.0   0.0   0:00.13 init
    9 syslog     20     0 191328 3748 3244 S   0.0   0.0   0:00.03 rsyslogd
   13 root        20     0  28356 2648 2420 S   0.0   0.0   0:00.00 cron
   15 root        20     0  72296 3096 2352 S   0.0   0.0   0:00.00 sshd
   31 bind        20     0 878108 38180 6888 S   0.0   0.0   0:00.56 named
   51 root        20     0  78636 3676 3120 S   0.0   0.0   0:00.00 login
   64 sysadmin   20     0  18508 3416 2960 S   0.0   0.0   0:00.00 bash
   90 root        20     0  60084 3288 2832 S   0.0   0.0   0:00.01 su
   91 root        20     0  18608 3528 3040 S   0.0   0.0   0:00.02 bash
```

### Obs

Utskriften från kommandot **top** ändras varannan sekund.

Kommandot **top** är ett interaktivt program, vilket innebär att du kan ge kommandon inuti programmet. Du kommer att använda kommandot **top** för att döda ping-processerna. Börja med att trycka på bokstaven **k**. Lagg märke till att en prompt nu har dykt upp:

```
top - 22:18:59 up 15 days, 13:56, 1 user, load average: 1.90, 1.52, 1.16
Tasks: 12 total, 1 running, 11 sleeping, 0 stopped, 0 zombie
%Cpu(s): 6.5 us, 3.0 sy, 0.0 ni, 90.2 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
KiB Mem : 13201464+total, 75271912 free, 49921140 used, 6821588 buff/cache
KiB Swap: 13419622+total, 13401260+free, 183608 used. 81460888 avail Mem
PID to signal/kill: [default pid = 1]

  PID USER      PR    NI  VIRT  RES  SHR  S  %CPU  %MEM     TIME+  COMMAND
    1 root        20     0   18376 3092 2816 S   0.0   0.0   0:00.13 init
    9 syslog     20     0 191328 3748 3244 S   0.0   0.0   0:00.04 rsyslogd
   13 root        20     0  28356 2648 2420 S   0.0   0.0   0:00.00 cron
   15 root        20     0  72296 3096 2352 S   0.0   0.0   0:00.00 sshd
   31 bind        20     0 1271584 38440 6888 S   0.0   0.0   0:02.58 named
   51 root        20     0  78636 3676 3120 S   0.0   0.0   0:00.00 login
   64 sysadmin   20     0  18508 3416 2960 S   0.0   0.0   0:00.00 bash
   90 root        20     0  60084 3288 2832 S   0.0   0.0   0:00.01 su
   91 root        20     0  18608 3528 3040 S   0.0   0.0   0:00.02 bash
```

Vid prompten "At the PID to kill:", skriv in PID-numret för den första aktiva **ping**-processen och tryck sedan på **Enter** . Lagg märke till att prompten nu ändras enligt nedan:

```
top - 02:03:27 up 9:24, 1 user, load average: 0.53, 0.27, 0.23
Tasks: 10 total, 1 running, 9 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.0%us, 1.8%sy, 0.0%ni, 95.9%id, 0.0%wa, 0.0%hi, 0.3%si, 0.0%st
Mem: 65969788k total, 7445364k used, 58524424k free, 288612k buffers
Swap: 2097148k total, 0k used, 2097148k free, 1096724k cached
PID to kill: 112

  PID USER      PR    NI  VIRT  RES  SHR  S  %CPU  %MEM     TIME+  COMMAND
  114 sysadmin   20     0 17212 2192 1960 R    1   0.0   0:00.10 top
    1 root        20     0 17868 2872 2624 S    0   0.0   0:00.14 init
```



Vid prompten "**Kill PID with signal [15]:**", skriv in signalen du vill skicka till processen. I det här fallet kan du bara trycka på **Enter** för att använda standardsignalen. Lägg märke till att det första **ping**-kommandot tas bort och att endast ett **ping**-kommando finns kvar i listan (du kan behöva vänta några sekunder när **top**-kommandot uppdateras):

```
top - 02:03:27 up 9:24, 1 user, load average: 0.53, 0.27, 0.23
Tasks: 10 total, 1 running, 9 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.0%us, 1.8%sy, 0.0%ni, 95.9%id, 0.0%wa, 0.0%hi, 0.3%si, 0.0%st
Mem: 65969788k total, 7445364k used, 58524424k free, 288612k buffers
Swap: 2097148k total, 0k used, 2097148k free, 1096724k cached
Kill PID 112 with signal [15]:
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
114	sysadmin	20	0	17212	2192	1960	R	1	0.0	0:00.10	top
1	root	20	0	17868	2872	2624	S	0	0.0	0:00.14	init

### Tänk på detta

Det finns flera olika numeriska värden som kan skickas till en process. Dessa är fördefinierade värden, och varje har en egen betydelse. Om du vill veta mer om dessa värden, skriv **man kill** i en terminal.

Prompten visar att standardsignalen är terminera-signalen, som kallas **SIGTERM** eller nummer **15**.

Nästa steg är att döda den återstående **ping**-processen på samma sätt som tidigare, men den här gången anger du värdet **9** på signalprompten **Kill PID with signal [15]:** istället för att acceptera standardvärdet **15**. Tryck på **Enter** för att bekräfta.

```
top - 02:04:12 up 9:25, 1 user, load average: 0.31, 0.24, 0.22
Tasks: 9 total, 1 running, 8 sleeping, 0 stopped, 0 zombie
Cpu(s): 2.9%us, 2.6%sy, 0.0%ni, 94.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 65969788k total, 7466640k used, 58503148k free, 288844k buffers
Swap: 2097148k total, 0k used, 2097148k free, 1097344k cached
PID to kill: 113
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	17868	2872	2624	S	0	0.0	0:00.14	init
19	syslog	20	0	171m	2796	2420	S	0	0.0	0:00.07	rsyslogd
23	root	20	0	19120	2020	1824	S	0	0.0	0:00.00	cron

### Tänk på detta

Signalsiffran **9**, eller **SIGKILL**, är en "tvångsavslutande" signal som inte kan ignoreras, till skillnad från standardvärdet **15**. Lägg märke till att alla referenser till **ping**-kommandot nu är borttagna från **top**.

Tryck på **q** för att avsluta **top**-kommandot. Skärmen visar nu att båda **ping**-kommandona har avslutats.

```
[1]- Terminated          ping localhost > /dev/null
[2]+ Killed               ping localhost > /dev/null
```

## Använd **pkill** och **kill** för att terminera processer

I den här uppgiften fortsätter vi att arbeta med processer. Du kommer att använda **pkill** och **kill** för att avsluta processer.

Börja med att skriva följande kommandon i terminalen:



```
sleep 888888 &
sleep 888888 &
```

Din utskrift bör likna det här:

```
root@localhost:~# sleep 888888 &
[1] 134
root@localhost:~# sleep 888888 &
[2] 135
```

Kommandot `sleep` används vanligtvis för att pausa ett program (till exempel ett shellskript) under en viss tid. I det här fallet används det bara för att ge ett kommando som tar lång tid att köra.

Se till att notera PID-numren för `sleep`-processerna i din virtuella maskin inför de kommande stegen! Dina PID-nummer kommer att skilja sig från de som visas i labben.

Nästa steg är att ta reda på vilka jobb som körs just nu genom att skriva:

```
jobs
```

Din utskrift bör se ungefär ut så här:

```
root@localhost:~# jobs
[1]-  Running                  sleep 888888 &
[2]+  Running                  sleep 888888 &
```

Nu ska du använda kommandot `kill` för att stoppa den första instansen av `sleep`-kommandot genom att skriva följande (ersätt `PID` med process-ID:t för din första sleep-process). Kör även `jobs` för att kontrollera att processen har avslutats:

```
ps
kill PID
jobs
```

Din utskrift bör likna det här:

```
root@localhost:~# ps
  PID TTY          TIME CMD
    1 ?            00:00:00 init
   51 ?            00:00:00 login
   90 ?            00:00:00 su
   91 ?            00:00:00 bash
  134 ?            00:00:00 sleep
  135 ?            00:00:00 sleep
  136 ?            00:00:00 ps
root@localhost:~# kill 134
root@localhost:~# jobs
[1]-  Terminated                sleep 888888
[2]+  Running                    sleep 888888 &
```



### Tips

Om du inte kommer ihåg PID för den första processen kan du bara skriva `ps`-kommandot, som visas ovan.

Nästa steg är att använda kommandot `pkill` för att avsluta den återstående `sleep`-processen, genom att använda programmets namn istället för PID:

```
pkill -15 sleep
```

Din utskrift bör se ungefär ut så här:

```
root@localhost:~# pkill -15 sleep
[1]-  Terminated          sleep 888888
[2]+  Terminated          sleep 888888
```

### Använda ps för att visa processer

Du kan använda kommandot `ps` för att visa processer. Som standard visar `ps`-kommandot bara de processer som körs i den aktuella skalet.

Starta en bakgrundsprocess med `ping` och visa aktuella processer med kommandot `ps`:

```
ping localhost > /dev/null &
ps
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# ping localhost > /dev/null &
[1] 138
root@localhost:~# ps
  PID TTY          TIME CMD
    1 ?            00:00:00 init
   51 ?            00:00:00 login
   90 ?            00:00:00 su
   91 ?            00:00:00 bash
  138 ?            00:00:00 ping
  139 ?            00:00:00 ps
```

Notera PID för `ping`-processen, du kommer att använda PID i ett senare steg.

Kör kommandot `ps` med alternativet `-e`, så visas alla processer.

```
ps -e
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# ps -e
  PID TTY          TIME CMD
    1 ?            00:00:00 init
    9 ?            00:00:00 rsyslogd
   13 ?            00:00:00 cron
   15 ?            00:00:00 sshd
   31 ?            00:00:02 named
   51 ?            00:00:00 login
```



Eftersom den här miljön är ett virtualiserat operativsystem finns det betydligt färre processer än vad man normalt skulle se om Linux kördes direkt på hårdvara.

Använd `ps`-kommandot med `-o`-alternativet för att ange vilka kolumner som ska visas.

```
ps -o pid, tty, time, %cpu, cmd
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# ps -o pid, tty, time, %cpu, cmd
PID TT          TIME %CPU CMD
  1 ?           00:00:00  0.0 /bin/bash /init
 51 ?           00:00:00  0.0 /bin/login -f
 90 ?           00:00:00  0.0 su
 91 ?           00:00:00  0.0 bash
138 ?           00:00:00  0.0 ping localhost
141 ?           00:00:00  0.0 ps -o pid, tty, time, %cpu, cmd
```

Använd alternativet `--sort` för att ange vilka kolumner som ska sorteras efter. Som standard sorteras en angiven kolumn i stigande ordning, detta kan tvingas genom att placera ett plus `+` framför kolumnnamnet. För att sortera i fallande ordning, använd ett minus `-` framför kolumnnamnet.

Sortera utskriften från `ps` efter `%mem`:

```
ps -o pid, tty, time, %mem, cmd --sort %mem
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# ps -o pid, tty, time, %mem, cmd --sort %mem
PID TT          TIME %MEM CMD
142 ?           00:00:00  0.0 ps -o pid, tty, time, %mem, cmd --sort %m
138 ?           00:00:00  0.0 ping localhost
  1 ?           00:00:00  0.0 /bin/bash /init
 90 ?           00:00:00  0.0 su
 91 ?           00:00:00  0.0 bash
 51 ?           00:00:00  0.0 /bin/login -f
```

Även om `ps`-kommandot kan visa hur stor andel av minnet en process använder, så visar `free`-kommandot den totala minnesanvändningen i systemet:

```
free
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# free
              total        used         free   shared  buff/cache   available
Mem:    132014640  50276284   4632368    10864    7105988   81105444
Swap:   134196220    183608  134012612
```

Stoppa `ping`-kommandot med följande `kill`-kommando och kontrollera sedan med `jobs`-kommandot:



```
kill PID
```

```
jobs
```

Din utskrift bör se ut ungefär så här:

```
root@localhost:~# kill 138
root@localhost:~# jobs
[1]+  Terminated                  ping localhost > /dev/null
```

## Granska systemloggar

Systemloggar är avgörande för många uppgifter, till exempel att felsöka operativsystemet och se till att systemet är säkert. Att veta var loggfilerna lagras och hur man underhåller dem är viktigt för en systemadministratör.

Det finns två demoner som hanterar loggmeddelanden: `syslogd`-demonen och `klogd`-demonen. Vanligtvis behöver du inte bry dig om `klogd`; den hanterar endast kärnans loggmeddelanden och skickar sin information vidare till `syslogd`-demonen.

Meddelanden som genererats av kärnan vid uppstart sparas i filen `/var/log/dmesg`. Kommandot `dmesg` låter dig visa aktuella kärnmeddelanden och ger även kontroll över om dessa meddelanden ska synas i terminalfönstret.

Den huvudsakliga loggfilen som `syslogd` skriver till är `/var/log/messages`.

Förutom loggning som görs av `syslogd`, finns det många andra processer som har egen loggning. Exempel på sådana är Apache webserver (loggfilen finns i katalogen `/var/log/httpd`), Common Unix Printing System (`/var/log/cups`) och `auditd`-demonen (`/var/log/audit`).

### Obs

På CentOS-system kallas `syslogd` för `rsyslogd`. **OBS `systemd-journald` är standard i moderna linuxversioner, både för Ubuntu och CentOS. Visas med `journalct`.** `syslogd` används i praktiken inte längre.

Systemloggar sparas i katalogen `/var/log`. Lista filerna i denna katalog:

```
ls /var/log
```

```
root@localhost:~# ls /var/log
alternatives.log  bootstrap.log  dist-upgrade  faillog  syslog
apt               btmp          dmesg         journal  tallylog
auth.log          cron.log      dpkg.log      lastlog  wtmp
root@localhost:~#
```

Varje loggfil motsvarar en tjänst eller funktion. Till exempel visar `auth.log`-filen information om auktorisation eller autentisering, som användares inloggningsförsök. Ny data läggs till längst ner i filen. Kör följande kommandon för att se ett exempel:

```
ssh localhost
{At the first prompt, type yes}
{At the second prompt, type abc}
{At the third prompt, type abc}
```



{At the fourth prompt, type abc}

```
tail -5 /var/log/auth.log
```

OBS du Kanske behöver installera `open-ssh` server..

```
root@localhost:~# ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is 5f:e2:43:0f:f9:26:e5:d5:77:ba:9e:95:72:9e:ee:64.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
root@localhost's password:
Permission denied, please try again.
root@localhost's password:
Permission denied, please try again.
root@localhost's password:
Permission denied (publickey,password).
root@localhost:~# tail -5 /var/log/auth.log
Apr  8 20:25:13 localhost sshd[117]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=
Apr  8 20:25:16 localhost sshd[117]: Failed password for root from ::1 port 58940 ssh2
Apr  8 20:25:28  sshd[117]: last message repeated 2 times
Apr  8 20:25:28 localhost sshd[117]: Connection closed by ::1 [preauth]
Apr  8 20:25:28 localhost sshd[117]: PAM 2 more authentication failures; logname= uid=0 euid=0 tty=ssh
```

The `ssh` command was used to generated data in the `/var/log/auth.log` file. Note the failed login attempts are logged in the `/var/log/auth.log` file.