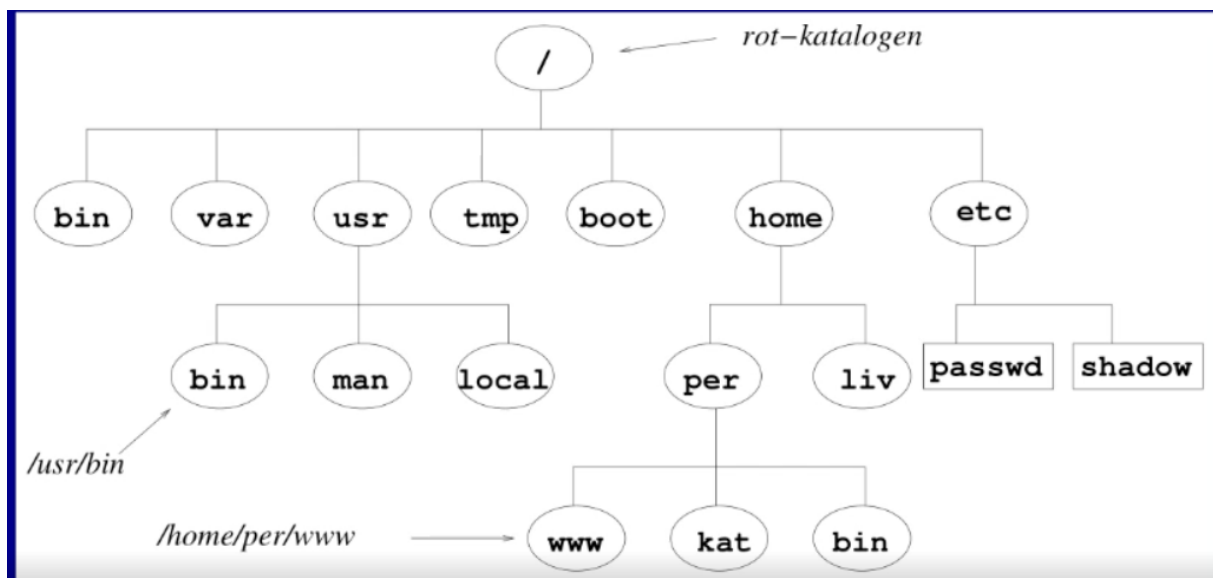


Linux lifeguide

- bin – her ligger det kjørbare programmer
 - var – her ligger det loggfiler
 - usr – lokale filer
-
- /usr/bin – de aller mest lokale filene
 - tmp – midlertidig filsystem som slettes automatisk når systemet genstarter eller når en spesifikk prosess er ferdig. Denne er tilgjengelig for alle brukere.
 - etc – konfigurasjonsfiler og installerte programmer. Sentral katalog av linux-filsystem som har viktig info om systemkonfigurasjon, instillinger for nettverk, passord og autorisasjonsregler.
 - "/etc/passwd" (brukerkontoinformasjon)
 - "/etc/fstab" (informasjon om monterte filsystemer)
 - "/etc/ssh/sshd_config" (konfigurasjonsfil for SSH-tjenesten)
 - "/etc/sudoers" (definisjoner av brukeres tillatelser til å utføre administrative oppgaver)



Linux-kommando	Virkning
\$ pwd	gir katalogen man står i
\$ cd home	change directory til "home" (kun fra /)
\$ cd /etc	flytter til /etc
\$ cd ..	flytter en katalog opp
\$ cd ../..	flytter to kataloger opp
\$ cd	går til hjemmekatalogen
\$ ls -l	viser alt som finnes i katalogen

Hvordan lage et shell script?

```
#!/bin/bash
```

Her betyr #! at dette scriptet skal tolkes av programmet som kommer etterpå, her /bin/bash som er selve shellet. Det er et binært program som ligger i mappen bin, og programmet heter bash. Og det er programmet som tolker alle kommandoene som kommer nedover for eksempel:

```
#!/bin/bash
```

```
pwd
```

```
ls
```

```
ls -l
```

Man må spesifisere hvor scriptet man vil kjøre ligger. Da må man skrive full path til mappen scriptet ligger i. Hvis jeg skriver ./script.sh (. er mappen jeg står i) får jeg fortsatt ikke kjørt det fordi den har ikke execute rettigheter.

Hvis jeg er i script.sh så kan jeg bruke kommandoene med tasten lengst nede til venstre (fn) og f10 tasten. Når man lager et script med jed vil det alltid lagres en sikkerhetskopi. Det eneste med nano er at man ser kommandoene nede og i jed er det å komme seg til dem. Med vim kan man få riktig innrykk osv (man går ved esc kolon w og q)

- uname – kommando som gir info om systemets navn
- uname -r – returnerer versjonsnummer for operativsystemet

- `uname -m` – returnerer informasjon om maskinvareplattformen
- `ls -a` – lister alle filer og mapper i en katalog inkludert de skjulte. A for all. Skjulte mapper og filer begynner vanligvis med punktum(.)
- `mkdir` – make directory
- `touch` – lager fil
 - `Touch dir2/fil.txt` – lager fil.txt under dir2
- `cat filnavn` - viser innholdet i filen filnavn
- `cp script.sh nyttScript.sh` – vil kopiere script.sh til nyttScript.sh
- `cp -i script.sh nyttScript.sh` gjør akkurat det samme, men den ber deg om en bekreftelse før den overskriver en eksisterende fil i målkatalogen. Da viser den en prompt som spør om du vil overskrive den eksisterende filen eller ikke, før den fortsetter med kopieringen. Dette gjør at du kan unngå uønskede overskrivninger av filer ved feil.
- `mv nyttScript.sh moved.sh` – vil ta nyttScript.sh og flytte til moved.sh som betyr at bare gir ett nytt navn
- `mv moved.sh nyMappe/` - vil flytte fra en mappe til en annen: her vil moved.sh til en allerede eksisterende mappe «nyMappe/»
- `ps aux` – alle prosesser som kjører
- `grep noe` – «global regular expression print» er et søkeverktøy som lar en søke etter tekststrenger i en fil eller gruppe av filer. Returnerer linjer som inneholder den angitte søketeksten
 - `grep -v` – inverterer meningen til søket, slik at det returnerer linjer som ikke matcher det spesifiserte mønsteret, i stedet for linjer som matcher mønsteret.
- `man kommando` – gir manualsiden til kommando. For å søke inni manualsiden til en kommando etter et ord kan man skrive en slash (/) når man er inni der, og så skrive ordet man vil søke etter. Da vil alle de bli markert og man kan taste seg til neste forekomst av ordet ved å trykke på n
- `history` – viser alle kommandoer gjort tidligere og man kan bla seg opp og ned
- `tree` – viser strukturert oppsett av filer og mapper

`cp ../script.sh .` → den setningen betyr kopier script.sh fra mappa over til hit (.)

En meget nyttig måte å teste ut bash-script på er å bruke -x parameteren. Kjør scriptet ditt, som heter f. eks. mittscript, slik:

```
$ bash -x mittscript
$ bash -x mittscript
+ '[' -f /etc/passwd ']'
+ echo Bra
Bra
og hver kommando som utføres blir skrevet til skjermen.
```

Absolute path og relative path er begreper som brukes til å beskrive hvordan en fil eller en mappe er adressert i et datamaskinnettverk.

Absolute path er en fullstendig adresse til en fil eller mappe, inkludert alle mapper som fører til den, fra rotmappen på harddisken. Eksempel: "/Users/username/Documents/file.txt"

Relative path derimot, beskriver filens plassering i forhold til gjeldende posisjon i filsystemet. Eksempel: "../Documents/file.txt" hvis du er i "/Users/username/Pictures" mappen.

- rmdir noe – kan bare slette en tom mappe
- rmdir -r noe – kan slette mappe med innhold
- rm -i – spør er du sikker på at du vil slette filen, selvom den er tom eller ikke
- locate fil.txt – vil lete etter fil.txt i hele systemet
- find dir2 -name fil.txt – vil finne fil.txt under dir2
 - find . -name «*il*» - betyr finn alle som har «il» inni seg
- grep 123 /etc/passwd – vil finne 123 sine alle forekomster i /etc/passwd
- wc -l /etc/passwd – word count gir antall linjer i /etc/passwd
 - Uten -l vil den også telle ord og tegn
- grep 123 /etc/passwd | wc -l - vil telle alle linjene i /etc/passwd med 123
- rm min\ fil – vil fjerne en fil som heter «min fil». Uten backslash vil den forsøke å fjerne to ulike filer hvor en heter min og den andre fil
- top - er en annen måte å vise alle prosesser på dynamisk
- hostname – navnet på maskinen man befinner seg på
- whoami - brukernavn
- type kommando – vil vise hvilken kommando som egentlig brukes
- /bin/pwd vil fortelle akkurat hvor du er i filsystemet

- `ln -s /usr/bin mbin` – lager en link som heter mbin. Ved `cd` til mbin vil det flyttes rett fra der du er til `/usr/sbin`
- `chmod a+w dir2/` - vil endre rettigheter slik at alle (a) vil kunne få skriverettigheter (w). Med a-w kan man fjerne alle de rettighetene
 - a er for alle
 - g er for group
 - u er for user
 - o er for den siste gruppen
- `umask` – bestemmer standardrettigheter og er vanligvis 0022
- `echo` – brukes til å vise output til terminalen og tar en streng tekst som et argument
- `set` – lister alle variabler som er definert i programmet

Shell variabler er noe man definerer og kan bruke senere i programmer.

Lage variabel:

```
$ hest=«Serina er en hest»  
$ echo $hest  
Serina er en hest  
$ esel=Serina  
$ echo $esel  
Serina
```

Fjerne variabel:

```
Unset hest  
Unset esel
```

Globale variabler

- `export navn` – vil gjøre navn variablene global som betyr at fra dette shellet når det startes ett nytt program vil den globale variabelen sendes med programmet. Det blir til en miljøvennlig variabel for andre prosesser, og er da synlig og tilgjengelig for andre kommandoer eller skript om kjører i samme terminaløkt.
- `export` – vil liste alle globale variabler
 - `export | more`

- `echo $PATH` – vil vise verdien til PATH-miljøvariabelen. \$PATH er en spesiell variabel som inneholder en liste over søkestier som shellen vil søke gjennom når den prøver å finne et program. Inneholder alle mappene det letes i når man utfører en kommando
- `echo $CLASSPATH` - CLASSPATH-variabelen inneholder en liste over kataloger og JAR-filer som er adskilt med kolon (:), og den brukes av Java Virtual Machine (JVM) for å søke etter Java-klasser når du kjører Java-programmer. Når du kjører et Java-program, vil JVM søke etter Java-klasser i katalogene og JAR-filene angitt i CLASSPATH, i den rekkefølgen de er angitt.

Eksempel: Hvis du har en JAR-fil med navnet "mylibrary.jar" i katalogen ~/lib, og du vil at JVM skal søke i denne katalogen når du kjører Java-programmer, kan du legge til følgende linje i din .bashrc eller .bash_profile fil:

```
export CLASSPATH="$CLASSPATH:$HOME/lib/mylibrary.jar"
```

- `env` – man kan finne alle globale variabler

Grep leter etter akkurat hva du skriver, og det er derfor veldig viktig å være obs med store og små bokstaver. Hvis man for eksempel vil søke for store og små bokstaver kan man enkelt ordne dette ved å skrive en -i foran slik:

```
env | grep -i path
```

Å sette PATH=«» (lik tom) vil gjøre slik at vi ikke kan utføre kommandoer som mv og ls. pwd finner man fordi det er en shell built inn.

Prosesser

Generelt lister vi prosesser med ps og uten argument lister vi de i det lokale shellen. Med aux viser vi alle som kjører i programmet. Hvis jeg vil drepe en prosess kan jeg ved hjelp av

```
ps aux | grep NAVN
```

få listet opp alle prosessene også kan jeg se prosessID og skrive kill etterfulgt av tallet.

Hvis vi vil vite akkurat hvor lang tid en prosess bruker så kan man skrive time ./run3.sh (hvis for eksempel shellen heter run3.sh).

Apostrofer

```
$ dir=mappe

$ echo 'ls $dir'      ' -> Gir eksakt tekststreng
ls $dir

$ echo "ls $dir"      " -> Variabler substitueres; verdien av $dir skrives ut.
ls mappe

$ echo `ls $dir`      ` -> utfører kommando!
fill fil2 fil.txt
```

Den mest robuste måten:

```
echo $(ls $dir)
```

```
haugerud@lap:~$ tall=$(seq 1 5)
haugerud@lap:~$ echo $tall
1 2 3 4 5
```

Seq kommando^

```
haugerud@lap:~$ for i in {1..4}
> do
> echo $i
> done
1
2
3
4
haugerud@lap:~$
```

For løkke i terminal

```
haugerud@lap:~$ for i in $(seq 1 4); do echo $i; done
1
2
3
4
haugerud@lap:~$
```

Annen måte å lage for løkke på

Omdirigering (viktig)

I Datateknikk refererer «stdin», «stdout», og «stderr» til standard input, standard output og standard error. De tre standard kanaler for å sende og motta data mellom et program og operativsystemet eller andre programmer:

- «stdin» (standard input) er den kanalen som et program bruker for å motta data, ofte fra tastaturet eller en fil
- «stdout» (standard output) er den kanalen som et program bruker for å sende data, ofte til skjermen eller en fil
- «stderr» (standard error) er en separat kanal for feilmeldinger, som ikke er en del av programmets normale output til «stdout»

Når vi skriver «| more» i linux sender vi utskriften fra forrige kommando til en «more»-prosess. «More» er en tekstvisningsverktøy som viser teksten side om side, slik at vi kan se en side om gangen. Dette gjør det enklere å se store mengder tekst uten å bli overveldet. Vi kan da bla gjennom teksten ved hjelp av piltastene, og trykke på «q» for å avslutte.

Når vi skriver echo hei så går det til standard out:

```
$ echo hei  
hei
```

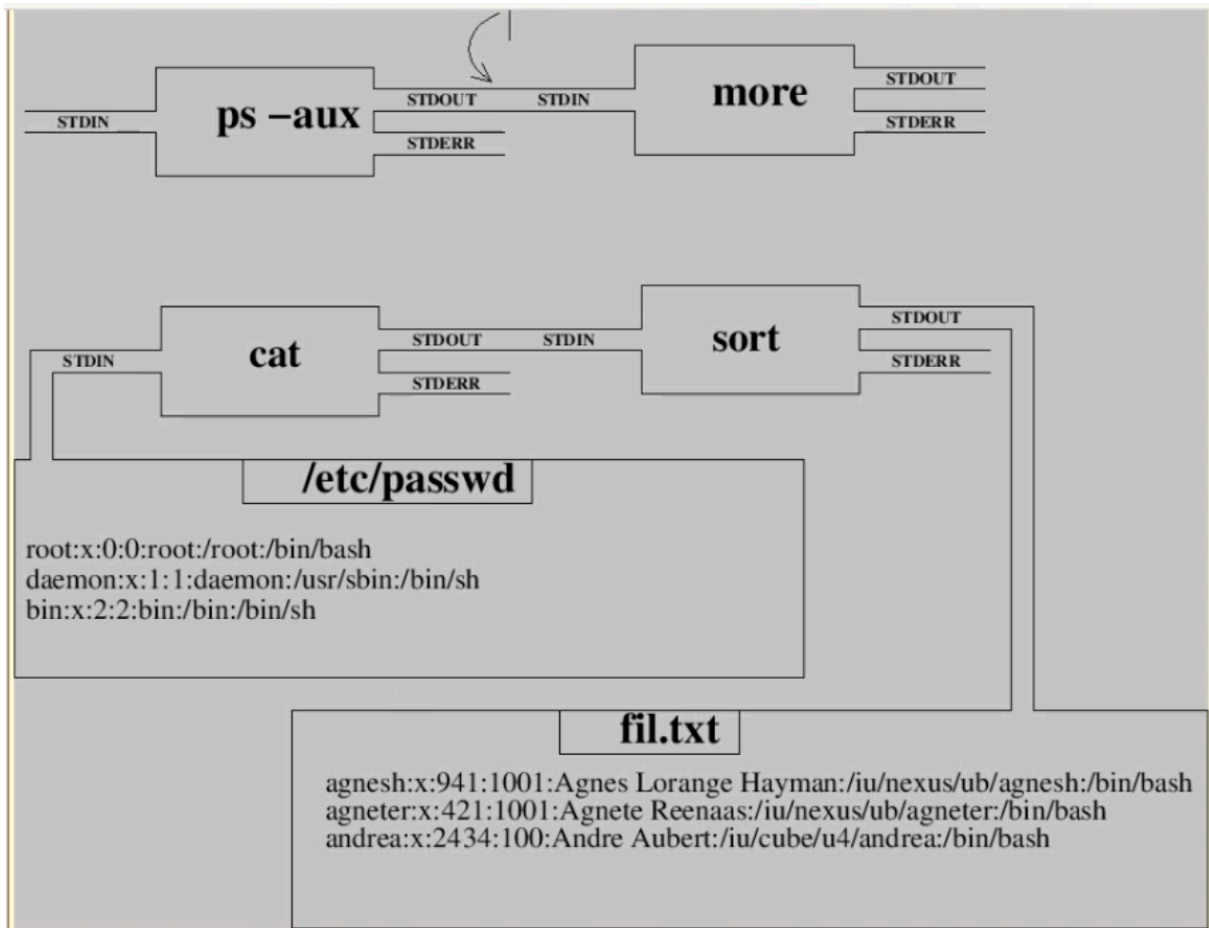
I stedet for å sende dette til standardout kan det omdirigeres til en fil:

```
$ echo hei > fil.txt  
$ cat fil.txt  
hei
```

Hvis jeg igjen omdirigerer en hei til samme fil vil den bare overskrives. Hvis jeg vil legge den inn og ikke erstatte det så bruker jeg 2 >>. Generelt kan det være nyttig å omdirigere feilmeldinger.

- /dev/null – er en type svart hull, alt du sender dit blir borte, men det betyr egentlig bare glem det og ikke lagre det
- Echo hei 2> err.txt – send kanal to til err.txt
- Echo hei &> err.txt – send både kanal en og to til err.txt.
- Ls finnesikke > fil.txt 2>&1 – hvor finnesikke er en fil som ikke finnes, fil.txt finnes og den siste kodedelen betyr: error skal sendes til kanal 2 og 1, hvor &1 er adressen til 1.

- Find / -name «*passwd*»
 - Find / -name «*passwd*» 2>&! | grep -v Permission – vil fjerne alt som har permission. 2>&1 → før standard 1 sendes til | så sendes alt som skal til standard 1 til pipen.
- Cat /etc/passwd | sort > fil.txt – til venstre for grep tegnet vil innholdet i filen /etc/passwd leses (inneholde informasjon om systemets brukere). Pipe vil sende utdata fra «cat» til input, til neste kommando i kommandolinjen. Sort vil sortere linjene i utdataen alfabetisk. > symbolet vil omdirigere den sorterte utdataen til en ny fil med navnet fil.txt, og overskrive eventuelt eksisterende innhold i filen. Så, kommandoen vil lese innholdet i filen «/etc/passwd», sortere linjene alfabetisk, og skrive den sorterte utdataen til filen «fil.txt»



Man kan se det fungerer som et rørsystem.

- Ps aux | awk '{print \$1}' | sort | uniq | wc -l → vil telle antall kjørende prosesser på brukerne. Uniq vil eliminere flere like forekomster, og vise en. Awk er et

tekstbehandlingsverktøy som brukes til å manipulere og hente ut data fra tekstfiler. det virker ved å lese inn inputfiler, linje for linje, og utføre handlinger basert på betingelser som er spesifisert i et «program».

- Awk '{print \$1}' fil.txt vil skrive ut den første kolonnen I hver linje i oppgitt fil, \$1 refererer i det første feltet i hver linje (brukernavn)

Jeg kan skrive og lage et nytt script direkte i terminal ved å skrive `cat > navnPåScriptet`. Etter å ha skrevet inn alt jeg vil ha der kan jeg gå ut med kontroll d.

Hvis man noensinne skrive «source navnPåScript» så kjører den direkte alle kommandoene fra skriptet ute.

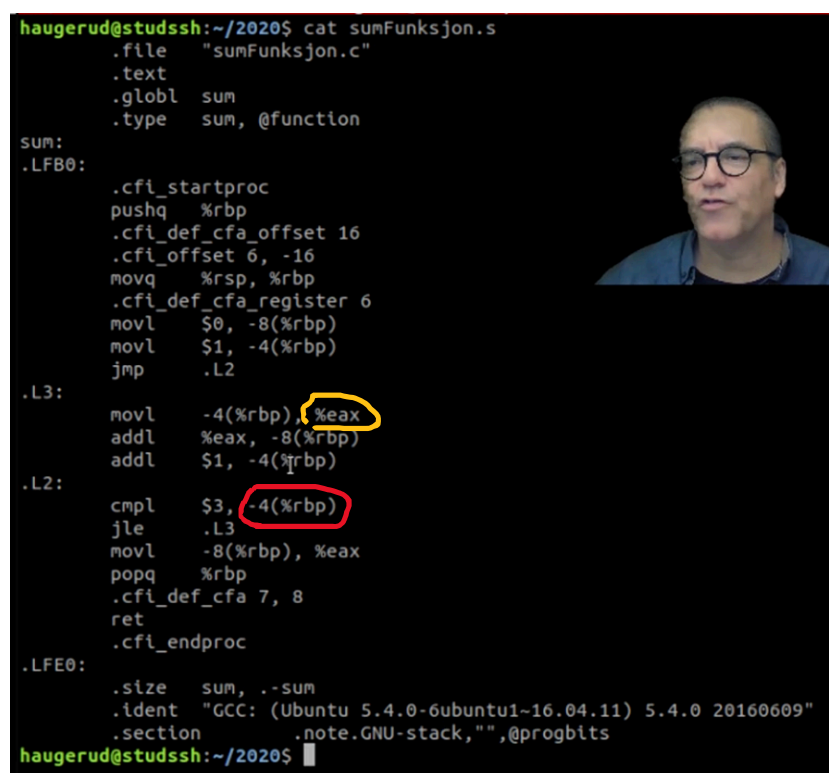
- Xxd a.out → vil hente ut innholdet fra den binære fila a.out
 - Libc.sc → er et bibliotek som vi bruker for å skrive ut printf for eksempel
- Gcc -c → bare kompiler, ikke load systembiblioteker som trengs for å få en ferdig kjørbar kode, bare kompiler akkurat det som er her
 - (gcc) -c sier lag maskinkode av det som er oppgitt etter
Man kan også be gcc kompilatoren om å lage assembly kode ved:
 - (gcc) -S funksjon.c

Litt assembly-kode:

-4(%rbp) hvor rbp er et register. Dette er en referanse til ram hvor rbp registeret ligger. Hvis det inni rbp ligger tallet 2, så er det en referanse til byte 2 i ram. -4 betyr at det er egentlig ikke byte nummer 2, men -4, så hvis rbp for eksempel er 8 så blir adressen til egt byte nr 4. (8 - 4).

%eax, hvor eax er et register. E betyr at det er et 32 bits register.

Movel betyr at vi skal flytte en long (som er en 32 bit). Samme med addl som også vil addere en long (32 bit). Vi skal bruke rax som er litt utvidet, en 64 bits registre.



```
haugerud@studssh:~/2020$ cat sumFunksjon.s
.file "sumFunksjon.c"
.text
.globl sum
.type sum, @function

sum:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $0, -8(%rbp)
movl $1, -4(%rbp)
jmp .L2

.L3:
movl -4(%rbp), %eax
addl %eax, -8(%rbp)
addl $1, -4(%rbp)

.L2:
cmpl $3, -4(%rbp)
jle .L3
movl -8(%rbp), %eax
popq %rbp
.cfi_def_cfa 7, 8
ret
.cfi_endproc

.LFE0:
.size sum, .-sum
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.11) 5.4.0 20160609"
.section .note.GNU-stack,"",@progbits
haugerud@studssh:~/2020$
```

Bash-scripting eller shell-programmering

Debugging: bash -x mittscript viser hva som skjer.

Demons er prosesser som alltid går i bakgrunne og utfører tjenester. \$mittprog& vil fortsette å kjøre etter loguot.

Det er veldig nyttig å legge inn if-tester i shell script. Tester skrives inni i [].

- `Echo $? →` hvor `$?` er en returverdi og vil returnere programmet som nettopp kjørte sin returverdi

Med kommando "escape + g» kan man skrive inn ett tall i en teksredigerer, som jed eller nano, og den lar deg hoppe til linja du skriver inn.

Script og argumenter:

variabel	innhold	verdi i eksempelet
<code>\$*</code>	hele argumentstrengen	fil1 fil2 fil3
<code>\$#</code>	antall argumenter	3
<code>\$1</code>	arg nr 1	fil1
<code>\$2</code>	arg nr 2	fil2
<code>\$3</code>	arg nr 3	fil3

- `Ls /proc →` inneholder mange mapper og filer. Den inneholder også masse info om systemet, prosesser og alt som foregår inni datamaskinen
 - `cat /proc/cpuinfo | more →` vil gi info om cpu-en som kjører på maskinen og masse mer
 - `cat /proc/meminfo | more →` forteller om hvor mye minne og rekke andre minneparametere

`ls -l /proc/ →` er det en masse tall, og hver av de har en prosessID.