

Linux forelesningsnotater

Hårek Haugerud
TKD
OsloMet

Contents

1 Forelesning 16/1-24(2 timer). Linux-shell, Linux-filsystem	2
1.1 Forelesningsvideoer	2
1.2 Hva er Linux?	2
1.3 Linux	3
1.4 Linux-fordeler	3
1.5 Hvor brukes Linux?	3
1.6 Hva er et shell?	3
1.7 Hvorfor shell/kommandolinje?	4
1.8 Innlogging	4
1.9 Linux filsystem	5
1.10 Hvordan man flytter seg i et Linux-filtre	5
1.11 Å lage et shell-script	5
1.12 Filbehandling (Viktig!)	6
1.13 Spesielle mapper	7
2 Forelesning 23/1-24(2 timer). Linux-filsystem, Linux-kommandoer	7
2.1 Sist	7
2.2 Forelesningsvideoer	8
2.3 Manualsider og apropos	8
2.4 Tidsbesparende triks i et Linux-shell	9
2.5 Linux-shellscript	10
2.6 Absolutt og relativ path	11
2.7 Mer filbehandling	12
2.8 Sletting av filer og mapper	12
2.9 Enda mer filbehandling	13
2.10 Lovlige filnavn	13
2.11 Prosesser	14
2.12 Linux-maskiner	14

2.13 Orientering: Hvem, hva, hvor	14
2.14 Symbolske linker til filer (symbolic links)	14
2.15 Symbolske linker til mapper	15
2.16 Filrettigheter	15
2.17 Hvordan forstå filrettigheter	16
2.18 Endre filrettigheter	17
2.19 umask	18
3 Forelesning 30/1-24(2 timer). Variabler, omdirigering og pipes	19
3.1 Forelesningsvideoer	19
3.2 Sist	19
3.3 Dagens faktum: UNIX' opprinnelse	20
3.4 Shell-variabler	20
3.5 Globale shell-variabler	21
3.6 Hvor ligger alle kommandoene egentlig? Svar: PATH	22
3.7 Prosesser	22
3.8 Apostrofer	22
3.9 Tilordne output fra kommando til en variabel	24
3.10 Omdirigering (viktig!)	24
3.10.1 Omdirigering til og fra filer	24
3.11 Omdirigering til og fra kommandoer; pipes	26
3.12 Piping standard error	26
3.13 Sub-shell	26
3.14 source	28
3.15 Kommandoer brukt under forelesningen	29
4 Forelesning 6/2-24(2 timer). Bash-scripting	29
4.1 Forelesningsvideoer	29
4.2 Sist	30
4.3 Shell-programmering	30

4.4	if-test	31
4.5	if-eksempel; fil-testing	31
4.6	Flere filtester og sammenligning	32
4.7	Logiske operatorer	32
4.8	for-løkke	32
4.9	Break og Continue	34
4.10	Numerikk	34
4.11	Script og argumenter	35
4.12	Argumenter i for-løkke og exit-verdier.	35
4.13	while	36
4.14	/proc - et vindu til kjernen	36
5	Forelesning 13/2-24(2 timer). Bash-scripting	37
5.1	Forelesningsvideoer	37
5.2	Passord-kryptering	38
5.2.1	Hashing-algoritmer	39
5.2.2	Passord-cracking	39
5.3	find	40
5.4	sed	40
5.5	sort	41
5.5.1	Sortert alfabetisk	41
5.5.2	Sortert alfabetisk etter andre kolonne	41
5.5.3	Sortert numerisk etter tredje kolonne	41
5.6	head og tail	42
5.7	cut	42
5.8	Input fra bruker	43
5.9	Lese filer og output med while og read	43
5.10	Arrays	44
5.11	Et vanlig problem med pipe til while og read	45

5.12	Assosiative array	46
5.13	funksjoner	46
5.14	funksjoner og parametre	47
5.15	Signaler og trap	48
5.16	Oversikt over shell-typer	48
5.17	Oppstartsfiler	49
6	Forelesning 27/2-24(2 timer). Linux-VMer, Screen, ssh-keys, cron	50
6.1	Sist	50
6.2	Forelesningsvideoer	50
6.3	root og sudo	51
6.4	Brukere	51
6.5	Grupper	52
6.6	Rettighetsprinsipper i Linux/UNIX miljøer	53
6.7	ssh-copy-id	53
6.8	Root aksess til server med sudo-rettigheter	54
6.9	Bakgrunnsjobber og screen	55
6.9.1	Å dele en screen med samme bruker	56
6.9.2	Bakgrunnsjobber og screen	57
6.10	scp	57
6.11	Backup med rsync og cron-tab	58
6.12	wget	59
6.13	gzip,bzip2, tar og zip	59
6.14	awk (ward)	60
7	Forelesning 5/3-24(2 timer). Containere og Docker	61
7.1	Forelesningsvideoer	61
8	Forelesning 12/3-24(2 timer). Docker og Dockerfile	62
8.1	Forelesningsvideoer	62

8.2	Endring siden innspillingen ble laget	63
9	Forelesning 12/3-24(2 timer). Docker og Dockerhub, shell script ytelse	63
9.1	Sist	64
9.2	Dockerhub	64
9.3	Shell-programmering, oppsummering	65
9.4	Hastighet til programmer skrevet i bash, python, perl, Java og C	66
9.5	Regulære uttrykk (referansestoff, ikke pensum)	67
9.6	Regulære uttrykk med grep(referansestoff, ikke pensum)	69
9.7	Spesielle tegn i Regulære uttrykk(referansestoff, ikke pensum)	69
9.8	Kvantifisering i Regulære uttrykk II(referansestoff, ikke pensum)	70
9.9	Spesielle koder i Regulære uttrykk(referansestoff, ikke pensum)	70
9.10	Bytte ut med regexp(referansestoff, ikke pensum)	70
10	Forelesning 2/4-24(2 timer). Docker Compose	71
10.1	Docker Compose og docker-compose.yaml	71
10.2	Docker Compose hello-world	71
10.3	Docker Compose nginx	72
10.4	Tjenester med flere samtidige containere	74
10.5	docker-compose build	75
11	Forelesning 9/4-24(2 timer). Virtuelle maskiner	75
11.1	Forelesningsvideoer	75
11.2	Virtualisering	76
11.3	Hvorfor virtualisering?	76
11.4	Isolasjon	77
11.5	Ressurssparing	77
11.6	Fleksibilitet	77
11.7	Skalering av ressurser	78
11.8	Programvare-utvikling	78

11.9 Skytjenester	78
11.10Historie	79
11.11Krav til virtualisering	79
11.12Hardware støttet virtualisering	79
11.13Type 1 hypervisor	79
11.14Type 2 hypervisor	80
11.15Binær oversettelse	80
11.16Paravirtualisering	81
11.17Ytelse	81
11.18Disk ytelse	81
11.19RAM ytelse	81
11.20CPU ytelse	81
11.21Live migration	84
11.22Simulering av migration	84
11.23Migration av Ubuntu	85
11.24Migration av IncludeOS	85
11.25Nested virtualisering	85
11.26QEMU og KVM	85
11.27Test av QEMU og KVM ytelse	86
11.27.1 sum.c	86
11.27.2 getppid.c	86
11.27.3 gettimeofday.c	87
11.27.4 forkwait.c	87
11.27.5 Bare metal	88
11.27.6 qemu med -enable-kvm	88
11.27.7 qemu uten -enable-kvm	88
11.28Unikernels og IncludeOS	89
12 Forelesning 16/4-24(2 timer). Windows og PowerShell	90

12.1 Forelesningsvideoer	91
12.2 Kort Windows historikk	91
12.3 Windows brukere	92
12.4 Windows grupper	92
12.5 Local Security Policy	92
12.6 Windows filsystemer og rettigheter	93
12.7 Windows kommandolinje script	93
12.8 Windows Script Host	93
12.9 VBScript	94
12.10VBScript Hello World!	94
12.11VBScript-eksempler(referansestoff)	94
12.12Input og popup-menyer(referansestoff)	95
12.13Windows PowerShell	96
12.13.1Verdens korteste Hello World program	97
12.14To viktige kommandoer	97
12.15Likheter med bash	98
12.15.1Omdirigering	99
12.16Variabler	99
12.17Environmentvariabler	100
12.18Apostrofer	101
12.19Objekter og Get-Member	101
12.20Undersøke typen til et objekt	103
12.21ps	104
12.22Foreach	104
12.23Installasjon av programmer fra PowerShell	105
12.24Select-String, PowerShells svar på grep	105
12.25Logiske operatører	106
12.26Kommandoer brukt under forelesningen som ble gitt i 2018	107

13 Forelesning 23/4-24(2 timer). Powershell, Windows registry	107
13.1 Forelesningsvideoer	107
13.2 Dagens sitat:Jeffrey Snover - PowerShell Architect	107
13.3 Windows script editor	107
13.4 Funksjoner	108
13.5 Summere antall bytes i filer	108
13.6 Stoppe prosesser med et gitt navn: nkill.ps1	109
13.7 PowerShell oneliner	109
13.8 Lese og sette registry verdier	110
13.9 Sort-Object og Select-Object	110
13.10DateTime	110
13.11array og strenger	113
13.12Event view og PowerShell	114

1 Forelesning 16/1-24(2 timer). Linux-shell, Linux-filsystem

Forelesning slides¹.

1.1 Forelesningsvideoer

Denne forelesningen ble ikke holdt live, men opptak av forelesningene er samlet her. NB! Her omtales innlogging til studssh, se endringer på dette i starten av Uke 3-oppgavene. I år skal vi istedet bruke data2500.cs.oslomet.no.

Samlet opptak av forelesningen:

linux.mp4² (56:34) Samlet opptak av alle delene nedenfor.

Opptak av forelesningen inndelt etter temaer:

linux1del1.mp4³ (06:10) Slides: Innledning, hva er Linux?

linux1del2.mp4⁴ (13:09) Slides: Linux-fordeler og bruksområder, containere, shell

linux1del3.mp4⁵ (04:55) Demo: Installere putty og logge inn til studssh fra Windows (bruk data2500.cs.oslomet.no istedet for studssh.cs.hioa.no)

linux1del4.mp4⁶ (02:10) Demo: Logge inn til studssh fra Mac og Linux-terminal (bruk data2500.cs.oslomet.no istedet for studssh.cs.hioa.no)

linux1del5.mp4⁷ (05:08) Slides: Linux filsystem og hvordan man flytter seg i fil-treet

linux1del6.mp4⁸ (05:05) Demo: Hvordan man flytter seg i et Linux filsystem

linux1del7.mp4⁹ (07:27) Demo: Hvordan lage et shell-script

linux1del8.mp4¹⁰ (12:01) Demo: Filbehandling og spesielle mapper

linux1del9.mp4¹¹ (03:47) Demo: Hvordan dokumentere oppgaver og hint til top og psuser-oppgavene i uke 3

1.2 Hva er Linux?

- Linux er et operativsystem = et stort og komplisert program som styrer en datamaskin
- Linux-kjernen laget av Linus Torvalds i 1991
- GNU/Linux er et mer korrekt navn
- Mest brukt som server OS
- Linux er et Unix-OS, andre er Mac OS X, FreeBSD, BSD, Solaris, AIX
- Unix ble laget av Ken Thompson og Dennis Ritchie i 1969

¹<https://www.cs.oslomet.no/haugerud/os/intro.pdf>

²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1a.mp4>

³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del1.mp4>

⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del2.mp4>

⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del3.mp4>

⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del4.mp4>

⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del5.mp4>

⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del6.mp4>

⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del7.mp4>

¹⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del8.mp4>

¹¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux1del9.mp4>

- Viktig del av Unix-filosofien: Sette sammen mange små programmer på mange måter

1.3 Linux

- Åpen kildekode, Linux-kjernen er GPL
- Det finnes mange distribusjoner av Linux, i alle størrelser.
- Små: i IP-kameraer, Mobiltelefoner(Android), Routere, switcher
- Store: Ubuntu/Debian, Red Hat/Fedora/Centos, SUSE/openSUSE
- GUI med vinduer og pek-og-klikk (for de som trenger det)

1.4 Linux-fordeler

- Gratis og åpen kildekode
- Naturlig del av åpen kildekode-prosjekter
- Sikkerhet
- Stabilitet

1.5 Hvor brukes Linux?

- Desktop/laptop: 1.5%
- Web servere: 70%
- Public Cloud: Amazon EC2 92% (Totalt: AWS 41%, Microsoft Azure 29%)
- Smartphone/nettbrett: Android 70%, iOS 24% (Unix basert)
- Supercomputere: 100% av de 500 største

1.6 Hva er et shell?

- kommandobasert verktøy
- tar imot kommandoer fra tastatur
- *Grensesnitt mot Linux-kjernen*

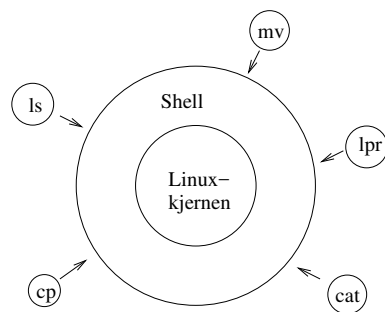


Figure 1: Linux-kommandoene sendes til shellen som er et skall rundt Linux-kjernen. Shelllet sørger for at oppdraget det får blir utført ved å gjøre et sett av systemkall til kjernen.

1.7 Hvorfor shell/kommandolinje?

Tidligere gikk all kommunikasjon med et Linux-system gjennom et shell. ¹²

- Stor frihetsgrad; ”Alt” er mulig å gjøre
- **Kompliserte oppgaver kan løses effektivt, ved å sette sammen mange små Linux-program; sort, grep, sed, cp, mv**
- *et programmeringsspråk: shell-script som kombinerer Linux-kommandoer; systemprogrammering*
- Vanskelig å automatisere og replikere en lang sekvens av pek og klikk
- Mye brukt i Linux automatisering, Cloud, Docker, Kubernetes, Git, osv

1.8 Innlogging

Hver bruker på et Linux-system har

- entydig brukernavn
- passord

Oversikt over alle brukere på systemet ligger i filen

- `/etc/passwd`

og de krypterte passordene ligger i filen

- `/etc/shadow`

¹²Spørsmål: Tilsvaret et shell-script i Linux et DOS batch-program (Som f. eks. autoexec.bat)? Svar: Ja, men shell-script er et mye kraftigere verktøy; flere muligheter

Kan ikke leses av vanlige brukere, kun av *root* (superuser)

Passordet settes/endres på OsloMet via web.

1.9 Linux filsystem

Filer er et helt sentralt Linux-begrep. *Alle data lagres som filer og strømmer av data fra tastatur og andre devicer blir behandlet som om de var filer.*

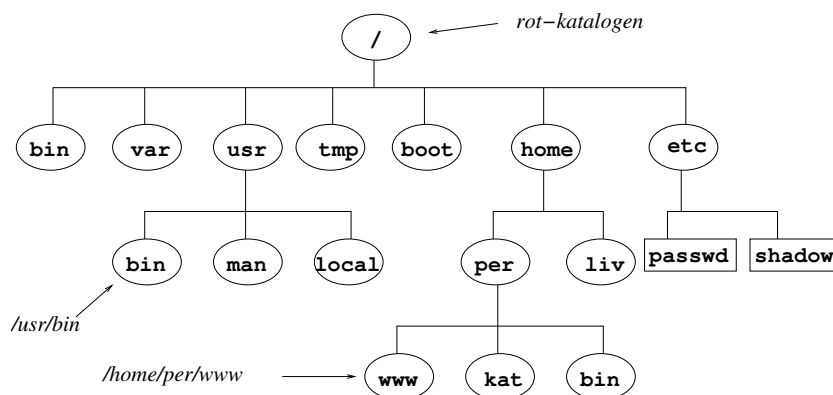


Figure 2: Et typisk Linux-filte

1.10 Hvordan man flytter seg i et Linux-filte

Linux-kommando	Virkning
\$ pwd	gir mappen/katalogen man står i (Print Working Directory)
\$ cd home	change directory til "home" (kun fra /)
\$ cd /etc	flytter til /etc
\$ cd ..	flytter en mappe opp
\$ cd ../../	flytter to mapper opp
\$ cd	går til hjemmemappen
\$ ls -l	viser alt som finnes i mappen

1.11 Å lage et shell-script

```
$ jed script.sh
```

- `#!` --> nå kommer et script
- `/bin/bash` --> skal tolkes av `/bin/bash`
- Rettigheter må settes slik at filen er kjørbare (x)

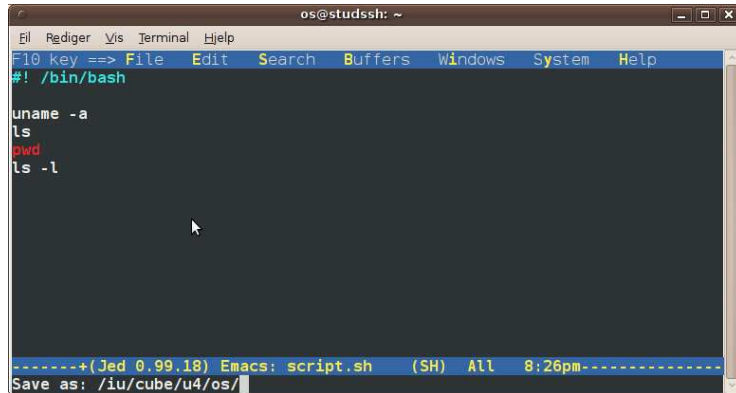


Figure 3: script.sh i jed

```
[os]studssh:~$ script.sh
-bash: ./script.sh: Permission denied

[os]studssh:~$ ls -l script.sh
-rw-r--r-- 1 os student 37 2010-01-06 20:23 script.sh

[os]studssh:~$ chmod 700 script.sh

[os]studssh:~$ ls -l script.sh
-rwx----- 1 os student 37 2010-01-06 20:23 script.sh

[os]studssh:~$ script.sh
Linux studssh 2.6.24-26-generic #1 SMP Tue Dec 1 18:37:31 UTC 2009 i686 GNU/Linux
tmp
/iu/cube/u4/os/mappe
total 4
drwxr-xr-x 2 os student 4096 2010-01-04 12:11 tmp

[os]studssh:~$
```

1.12 Filbehandling (Viktig!)

"Alt" i Linux er filer; *vanligvis ASCII-filer*.

Linux-kommando	resultat
\$ ls	lister filer/mapper i mappen der du står
\$ ls -l	ekstra info
\$ ls -a	lister “skjulte” filer (.fil)
\$ ls /etc	lister alt i /etc
\$ mkdir mappe	lager en mappe
\$ cat fil1	skriv innhold til skjermen
\$ touch fil2	lag en tom fil med navn “fil2”/oppdaterer tidsstempel hvis den fins
\$ jed fil3.txt	editer en fil med navn fil3.txt. Rask og effektiv editor som også kan brukes fra putty.
\$ emacs fil4.txt	editer en fil med navn fil4.txt. Mer omfattende GUI-editor.
\$ cp fil1 fil2	kopierer fil1 til fil2
\$ cp -i fil1 fil2	Spørr om fil2 skal overskrives
\$ mv fil1 fil2	Endrer navn fra fil1 til fil2
\$ mv fil2 /tmp	Flytter fil2 til /tmp

1.13 Spesielle mapper

betegnelse	Mappe
.	den du står i
..	den over
../..	den over den igjen
~	Din hjemmemappe

Bruk av ~:

```
$ echo ~
/iu/nexus/ud/haugerud
$ cat ~/.bashrc (skriver din .bashrc til skjermen.)
$ echo ~haugerud
/iu/nexus/ud/haugerud
$ cd ~/www      {# gå til din hjemmesidemappe.}
```

2 Forelesning 23/1-24(2 timer). Linux-filsystem, Linux-kommandoer

2.1 Sist

- Jobb med oppgaver!
- Shell
- Shell-script

2.2 Forelesningsvideoer

Denne forelesningen ble ikke holdt live, men opptak av forelesningene er samlet her. Opptak av forelesningen inndelt etter temaer:

linux2del1.mp4¹³ (02:42) Demo: Manualsider
linux2del2.mp4¹⁴ (04:02) Demo: Tidsbesparende triks i et Linux-shell
linux2del3.mp4¹⁵ (03:08) Slides: Oppsummering om Linux-shellsript
linux2del4.mp4¹⁶ (03:06) Demo: Absolutt og relativ path
linux2del5.mp4¹⁷ (06:42) Demo: Flytte og kopiere mapper og undermapper
linux2del6.mp4¹⁸ (04:57) Demo: Hvordan slette mapper og filer. Mer filbehandling.
linux2del7.mp4¹⁹ (07:05) Demo: Filnavn, prosesser, Linux-hosts, Orientering
linux2del8.mp4²⁰ (03:35) Demo: Symbolske linker til mapper og filer
linux2del9.mp4²¹ (10:29) Demo: Informasjon fra `ls -l` og filrettigheter
linux2del10.mp4²² (04:03) Demo: Alternativ bruk av `chmod`. Umask og default rettigheter.

2.3 Manualsider og apropos

For å finne ut hvordan en viss kommando (f. eks. `date`) virker, kan man slå opp i manualsiden med

```
$ man date
```

Manualsiden er vanligvis for lang til å vises på en side. Du kan manøvrere deg nedover en side av gangen ved å taste `space`. Man søker ved å taste `/søkeord` og så taste `"n"` fortløpende for flere forekomster. Tast `"q"` for å avslutte.

Apropos: finner kommandoer relatert til et emne.

```
haugerud@studssh:~$ apropos compare
[ (1)                - check file types and compare values
bcmp (3)             - compare byte sequences
bzipcmp (1)          - compare bzip2 compressed files
bzdiff (1)           - compare bzip2 compressed files
cg_diff (1)          - compares two Ccache output files
cmp (1)              - compare two files byte by byte
comm (1)             - compare two sorted files line by line
diff (1)             - compare files line by line
```

```
haugerud@studssh:~$ man diff
```

¹³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del1.mp4>
¹⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del2.mp4>
¹⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del3.mp4>
¹⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del4.mp4>
¹⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del5.mp4>
¹⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del6.mp4>
¹⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del7.mp4>
²⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del8.mp4>
²¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del9.mp4>
²²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux2del10.mp4>

NAME

`diff` - compare files line by line

SYNOPSIS

`diff [OPTION]... FILES`

DESCRIPTION

Compare FILES line by line.

Mandatory arguments to long options are mandatory for short options too.

`--normal`

output a normal diff (the default)

`-i, --ignore-case`

ignore case differences in file contents

Alt som er listet i [] er opsjoner som kan men ikke må tas med. Tast "q" for å avslutte.

```
studssh$ diff -i fil1 fil2
```

Og et google-søk med 'linux command line' eller 'linux bash' og det du ønsker å finne gir ofte raskt det du leter etter.

2.4 Tidsbesparende triks i et Linux-shell

Det finnes mange smarte knep som gjør inntasting av shell-kommandoer enklere. Bla tilbake i tidligere kommandoer og rediger dem med piltastene og trykk TAB-tasten for å finne mulige forlengelser av det du har skrevet inn.

- Tidligere kommandoer kan gjentas og endres med piltaster og vanlig editering
- ←,→ editere en tidligere kommando, og utføre den på nytt med Enter
- \$ `exit` og CTRL-d avslutter et shell.
- Bruk Ctrl-C Hvis du skal stoppe et program som går i et shell

Man kan søke bakover i tidligere kommandoer ved å taste CTRL-r og så fortløpende det man søker etter, "mitt" i eksempelet under:

```
(reverse-i-search)'' :
(reverse-i-search)'mitt': mv mitt.sh new
```

Generelt kan man få skallet til å komplettere filnavn og kommandoer ved å trykke på TAB-tasten. Skal man f.eks. flytte til mappen kjempelangtnavn:

```
$ cd kj [TAB]
$ cd kjempelangtnavn
```

Hvis du også har en mappe der du står som heter kjiipt, piper det etter TAB

```
$ cd kj [TAB] [TAB]
kjempelangtnavn/ kjiipt/
$ cd kje [TAB]
$ cd kjempelangtnavn
$ loc [TAB]
local      locale      localedef  locate      lockfile
```

Du kan også bruke TAB-tasten til å komplettere kommandoer du skal utføre. Skriver du

```
$ net
```

og så trykker TAB-tasten to ganger vil du få alle kommandoer som begynner på "net".

```
$ net
net      netcat      netkit-ftp  netstat
```

tast en "s" og TAB igjen og "netstat" fullføres av shellet.

Se mer om dette og andre tips under 'Nyttige tips om bruk av Linux' under Linux-help ikonet på kurs-siden.

Hvis du kjører Linux med grafisk brukergrensesnitt vil i noen tilfeller kopiering og lignende være litt anderledes enn i andre systemer:

- Copy = Merk område med venstre musetast
- Paste = høyre musetast (putty) eller midtre musetast(linux)
- Windows varianten: cut=CTRL-x copy=CTRL-c og paste=CTRL-v virker i de fleste nyere GUI-applikasjoner

2.5 Linux-shellscript

- Samling av kommandolinjer
- program som utføres linje for linje
- Kompileres ikke

+ Raskt å lage.

+ Fint for små oppgaver.

- Langsamt i forhold til kompilert kode.
- Mangler avanserte datastrukturer.
- Kryptisk syntaks

- Vanskelig å feilsøke/debugge

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.
```

2.6 Absolutt og relativ path

En path (bane) til en mappe eller en fil angis alltid absolutt eller relativt til posisjonen man er i filtreet. Absolutt path begynner alltid med / som er rot-mappen som alle de andre henger på.

```
$ pwd
/
$ cd home
$ pwd
/home
$ cd etc          <----- Relativ path
bash: cd: etc: No such file or directory
$ cd /etc         <----- Absolutt path
```

Alt I: <i>Relativ path</i>	AltII: <i>Absolutt path</i>
Fra /	Fra hvor som helst
\$ cd usr	\$ cd /usr/bin
\$ cd bin	\$ pwd
\$ pwd	/usr/bin
/usr/bin	
Begynner <i>ikke</i> med /	Begynner med /

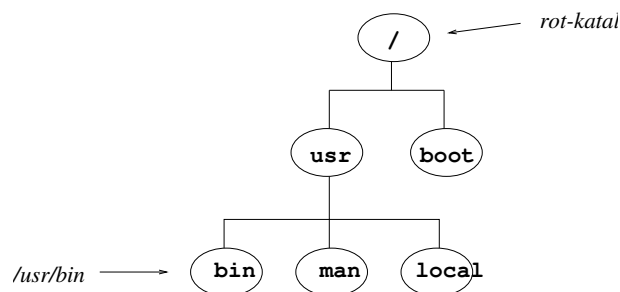


Figure 4: Mappen /usr/bin i filtreet.

2.7 Mer filbehandling

Linux-kommando	resultat
\$ tree	viser den underliggende mappe-strukturen
\$ sudo apt-get install tree	Installerer programmet tree om det ikke finnes fra før
\$ mv dir1 dir2	flytter mappen dir1 til mappen dir2
\$ cp /bin/c* /tmp	Kopier alt som begynner på c i /bin til /tmp
\$ cp -R dir1 dir2	Kopier dir1 med undermapper til dir2

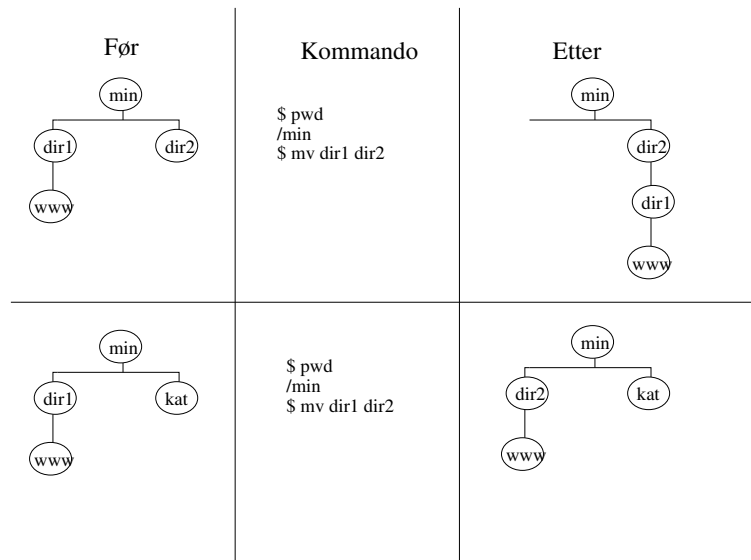


Figure 5: mv av hele mapper

2.8 Sletting av filer og mapper

NB! Fjernes for godt (ingen undelete)

Linux-kommando	resultat
\$ rm fil1	sletter fil1 (umulig å få tilbake)
\$ rmdir Mappe	kun tom mappe
\$ rm -R Mappe	sletter mappe og alle undermapper. Farlig
\$ rm -i fil2	ber om bekreftelse først

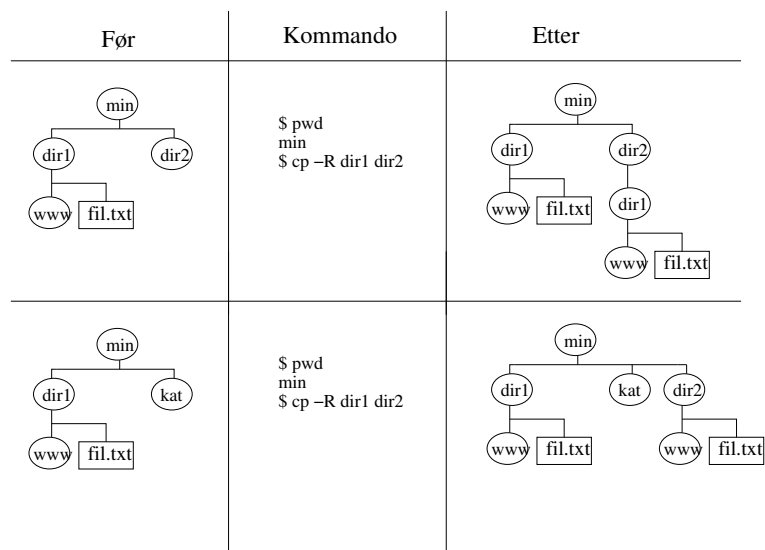


Figure 6: cp -R av hele mapper

2.9 Enda mer filbehandling

Linux-kommando	resultat
\$ locate noe	finner alle filer og mapper med navn som inneholder tekststrengen "noe"
\$ find tmp -name fil.txt	finner alt under tmp med navn som inneholder fil.txt
\$ find . -name "*fil*"	finner alt under mappen du står som inneholder strengen "fil" i navnet
\$ more fil1	skriv til skjerm; en side av gangen
\$ grep group /etc/passwd	skriv til skjerm alle linjer som inneholder strengen group
\$ wc -l /etc/passwd	tell antall linjer i /etc/passwd
\$ grep group /etc/passwd wc -l	tell antall linjer som inneholder strengen group

2.10 Lovlige filnavn

Alle tegn untatt / er lov// unngå spesielle tegn som æ,ø,å og mellomrom *for de må spesialbehandles* (som "en fin fil")

Bruk A-Z a-z 0-9 _ .

- fil1
- fil1.txt
- Index.html
- VeldigLangeFilnavn.help
- fil2.ver3.tekst

Linux er case-sensitiv. **fil1** er **IKKE** den samme filen som **FIL1**.

2.11 Prosesser

- `ps`
- `ps aux`
- `ps aux | grep root`
- `man ps`
- `top`

2.12 Linux-maskiner

- Hver Linux-maskin kalles en "host" (vert)
- Flere brukere kan logge inn og bruke samme host samtidig.
- Hver host har et navn: studssh (Linux, fil og www-server), rex (desktop), etc.
- Entydig adresse som kan nås fra hele verden krever domenenavn i tillegg: studssh.cs.oslomet.no, login.uio.no

Man kan logge inn fra hvorsomhelst i verden til en annen maskin:

```
ssh os@studssh.cs.oslomet.no
os@studssh.cs.oslomet.no's password:
[os]studssh:~$
```

Når man logger inn med ssh (Secure Shell) krypteres alt som sendes.

2.13 Orientering: Hvem, hva, hvor

Linux-kommando	Gir deg
<code>\$ whoami</code>	brukernavn
<code>\$ hostname</code>	maskin-navn
<code>\$ uname</code>	Operativsystem (Linux, Solaris,.....)
<code>\$ uname -a</code>	OS, versjon, etc.
<code>\$ who</code>	hvem som er logget inn
<code>\$ type chmod</code>	hviklet program kjøres med chmod?
<code>\$ history</code>	tidligere kommandoer

2.14 Symbolske linker til filer (symbolic links)

```
$ ln -s fil1 fil2
$ ls -l
-rw----- 1 haugerud drift          20 Sep  4 18:44 fil1
lrwxrwxrwx 1 haugerud drift          4 Sep  4 18:43 fil2 -> fil1
$ cat fil2
Denne teksten står i fil1
```

2.15 Symbolske linker til mapper

Ved å lage en link til en mappe kan man lage en snarvei i filtreet.

```
$ pwd
/home/min/dir1
$ ln -s /usr/bin mbin
$ ls -l
lrwxrwxrwx    1 haugerud drift          8 Sep  4 19:02 mbin -> /usr/bin
```

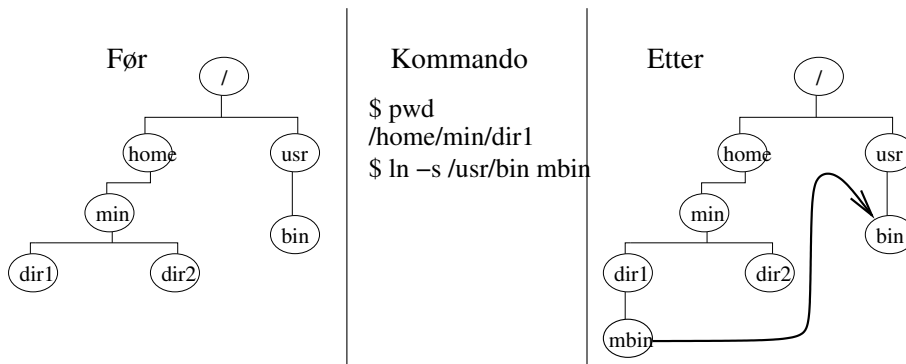


Figure 7: Symbolsk link fra mbin til /usr/bin

```
$ cd mbin
$ ls
822-date          giftopnm          awk              rcsmerge
Esetroot          giftrans          ncftp            rdist
.....etc.....   # alle filene i /usr/bin
$
$ pwd
/home/min/dir1/mbin    # /bin/pwd gir path relativt til linken
$ type pwd
pwd is a shell builtin # kommandoen pwd er bygd inn i bash
$ /bin/pwd
/usr/bin               # /bin/pwd gir absolutt path til der du er
$ cd ..
$ pwd
/home/min/dir1        # ikke /usr
```

2.16 Filrettigheter

Alle filer tilhører en bruker og en gruppe av brukere. For hver mappe og fil kan eieren sette rettighetene for

- brukeren filen tilhører (seg selv)
- gruppen filen tilhører

- alle andre

```
$ ls -l
-rwxrw-r-- 1 haugerud drift          7512 Aug 30 14:20 fil.exe
```

- fil (d = mappe, l = link)

rwx fileier sine rettigheter (r = read, w = write, x = executable)

rw- gruppens rettigheter (kan lese og skrive)

r-- alle andre sine rettigheter (kan bare lese filen)

1 antall hard links

haugerud eiers brukernavn

drift gruppen som filen tilhører (definert i /etc/group)

7512 antall byte

Aug 30 14:20 tidspunkt filen sist ble endret

fil.exe filnavn

For mapper betyr x at man har tilgang til mappen. Er ikke x satt, kan man ikke gå dit. Og da kan man heller ikke liste filer der eller kopiere noe dit.

2.17 Hvordan forstå filrettigheter

Rettighetene til eier, gruppe og andre er representert med kun tre bit. Dette begrenser hvor mange forskjellige rettigheter man kan ha, samtidig gjør det enkelt å representere rettighetene som tall og å regne ut rettigheter i hodet. Dette gjør man på følgende måte:

- kjørerrettigheter = 1
- skriverettigheter = 2
- leserettigheter = 4

Ved hjelp av disse tallene (og tallet 0) kan vi telle fra 0 til 7 ved å kombinere dem på forskjellige måter. F.eks:

- Kjørerrettigheter + leserettigheter = 5
- Kjøre + lese + skrive = 7 (maks)
- skrive og kjøre, men ikke lese = 3

2.18 Endre filrettigheter

Numerisk endring av filrettigheter:

```
$ chmod 644 fil.txt <- gir rw-r--r--  
$ chmod -R 755 dir <- gir rwxr-xr-x til dir og alle filer og mapper under dir
```

Tallene betyr:

rwx	octal	tekst
000	0	- - -
001	1	- - x
010	2	- w -
011	3	- w x
100	4	r - -
101	5	r - x
110	6	r w -
111	7	r w x

For mapper: Rettighet x (execute) gir tilgang til mappen.

Alternativt:

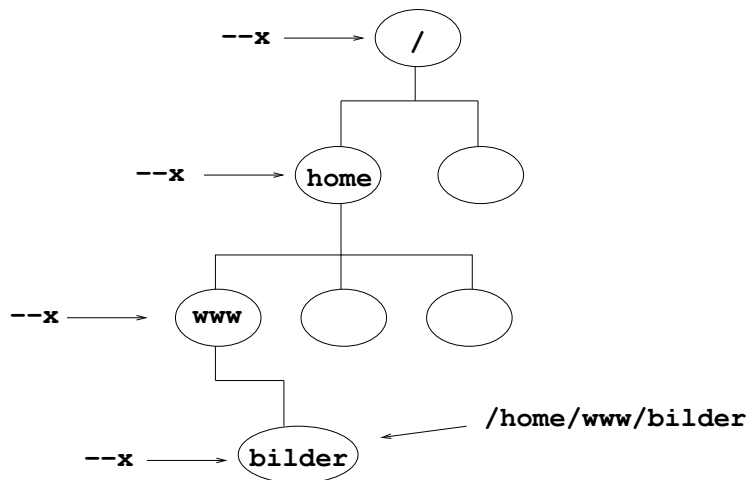


Figure 8: Man må ha tilgangsrettighet (x) til alle mappene i path for å kunne lese en fil (her: se et web-bilde i /home/www/bilder).

```
$ chmod a+xr fil.txt <- gir read og execute til alle (a)  
$ chmod g-w fil.exe <- fratar gruppen skriverrettigheter
```

u = user, g = group, o = other, a = all (både u, g og o)

2.19 umask

Kommandoen `umask` setter hva som skal være standard rettigheter for nye filer og mapper.

```
$ umask
0022
$ mkdir dir
$ touch fil
$ ls -l
total 1
drwxr-xr-x  2 haugerud drift 512 Aug 30 23:52 dir
-rw-r--r--  1 haugerud drift   0 Aug 30  2006 fil
```

`umask` er en 'maskering'. For hver brukergruppe settes rettighet '111 minus mask' for mapper og '110 minus mask' for filer. Bit som er satt i mask, settes alltid til null.

```
$ umask 027
$ mkdir dir2
$ touch fil2
$ ls -l
total 2
drwxr-x---  2 haugerud drift 512 Aug 30 23:53 dir2
-rw-r----- 1 haugerud drift   0 Aug 30  2006 fil2
```

Det er ikke så viktig å vite detaljene i hvordan `umask` virker, for man kan raskt teste ut hva resultatet blir om man har en viss forståelse. Slik beregnes rettighetene:

For mappe	7 = 111	7 = 111	7 = 111
minus mask	0 = 000	2 = 010	7 = 111
resultat	7 = 111	5 = 101	0 = 000
rettighet	7 = r w x	5 = r - x	0 = - - -

og for filer

For fil	6 = 110	6 = 110	6 = 110
minus mask	0 = 000	2 = 010	7 = 111
resultat	6 = 110	4 = 100	0 = 000
rettighet	6 = r w -	4 = r - -	0 = - - -

Spørsmål

Gir skriverettighet leserettighet? Har man automatisk leserettigheter hvis man har skriverettighet?

Svar: Nei, r, w og x settes uavhengig av hverandre. Men `-w-` (kun skriverettigheter) brukes i praksis aldri.

Kan man sette bare skriverettigheter på en fil? Ja. `chmod 200 fil` Da kan man skrive til filen, men ikke lese den.

Hvilke rettigheter trenger man for å kunne slette en fil? w, skriverettigheter.

Kan man slette en fil man selv ikke har skriverettigheter til? Ja, om man har skriverettigheter til mappen den ligger i, men man får en advarsel og blir spurt først. Det samme gjelder en fil med ingen rettigheter, 000.

Hva skjer med eierskapet om man kopierer en annens fil? Da blir man selv eier til den kopierte filen.

3 Forelesning 30/1-24(2 timer). Variabler, omdirigering og pipes

3.1 Forelesningsvideoer

Denne forelesningen ble ikke holdt live, men opptak av forelesningene er samlet her. Opptak av forelesningen inndelt etter temaer:

linux3del1.mp4²³ (03:40) Demo: Shell-variabler
linux3del2.mp4²⁴ (05:32) Demo: Globale shell-variabler
linux3del3.mp4²⁵ (02:12) Demo: Hvor ligger alle kommandoene egentlig? Svar: PATH
linux3del4.mp4²⁶ (05:11) Demo: Prosesser
linux3del5.mp4²⁷ (08:05) Demo: Apostrofer og å tilordne output fra kommando til en variabel
linux3del6.mp4²⁸ (04:22) Demo: Omdirigering
linux3del7.mp4²⁹ (10:37) Demo: Omdirigering til og fra filer
linux3del8.mp4³⁰ (06:02) Demo: Omdirigering til og fra kommandoer; pipes
linux3del9.mp4³¹ (1:07) Demo: Piping av standard error
linux3del10.mp4³² (04:03) Demo: Sub-shell
linux3del11.mp4³³ (04:02) Demo: source

3.2 Sist

- filbehandling
- Kommandoer: cp, mv, rm, grep
- Rettigheter

²³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del1.mp4>

²⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del2.mp4>

²⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del3.mp4>

²⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del4.mp4>

²⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del5.mp4>

²⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del6.mp4>

²⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del7.mp4>

³⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del8.mp4>

³¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del9.mp4>

³²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del10.mp4>

³³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux3del11.mp4>

- `chmod`

3.3 Dagens faktum: UNIX' opprinnelse

Linux er en åpen kildekode variant av operativsystemet UNIX som opprinnelig ble skrevet i PDP-7 assembler av Ken Thompson i 1969 ved AT&T Bell Laboratories. Han gjorde det for å kunne kjøre spillet "space travel" på minicomputeren PDP-7 (9 Kbyte internminne). Thompson lagde senere programmeringsspråket B og var også med å lage språket Go mens han jobbet hos Google. Programmeringsspråket C ble laget av Dennis Ritchie rundt 1970 for å kunne skrive UNIX i et plattformuavhengig språk; noe som ble gjort i 1973. UNIX og C hadde stor betydning for utviklingen av dataindustrien.



Figure 9: Ken Thompson(foran) og Dennis Ritchie ved en PDP-11.

3.4 Shell-variabler

I et Linux-shell finnes det systemvariabler som `$HOME`, `$PATH`, `$PS1`, `$CLASSPATH` og mange andre som systemet og mange applikasjoner som f. eks. Jbuilder bruker. En bruker kan også definere sine egne lokale variabler med selvlagde variabelnavn. Det er vanlig, men ikke påkrevet å bruke små bokstaver i variabelnavnet til lokale variable.

```
$ os=Linux
$ dato='Thu Jan 25'
```

Det må *ikke* være mellomrom før og etter `=`. Variabler refereres til med et `$`-tegn foran navnet:

```
$ echo $os
```

```
Linux
$ echo $dato
Thu Jan 25
```

Navnet kan avgrenses med

```
$ echo $osfrelst

$ echo ${os}frelst
Linuxfrelst
```

Kommandoer for å liste og fjerne variabler:

```
$ set          # Lister alle definerte variabler og funksjoner
$ unset os     # Fjerner definisjonen av os
```

3.5 Globale shell-variabler

I utgangspunktet er shell-variabler lokale og bare kjent i det shellet de har blitt definert. Man kan gjøre variabler globale med kommandoen **export**:

```
$ date='30. januar'
$ os=Linux
$ export os
$ export nyglobal='ny global verdi'
```

Variabelene `$os` og `nyglobal` er nå globale og vil arves av nye shell som startes, men det vil ikke `$date`:

```
$ bash
$ echo $os
Linux
$ echo $nyglobal
ny global verdi
$ echo $date
```

men som vi ser kjenner det nye shellet ikke til variabelen `$date`, den er lokal og bare kjent i shellet den ble definert i.

- vanlig å skrive globale variabler med store bokstaver
- kalles ofte ENVIRONMENT variabler (`$HOME`, `$PATH`, `$EDITOR`, `$CLASSPATH`)
- leses av programmer som ønsker å vite default editor, classpath, etc
- Globale variabler kan listes med **export**
- Variabler kan defineres direkte som globale i bash: `$ export os=Linux`
- Kommandoen `$ env` lister alle definerte globale variabler

3.6 Hvor ligger alle kommandoene egentlig? Svar: PATH

`$PATH` er en global variabel som inneholder en streng med alle mapper (separert med kolon) som shellet leter i for å finne kjørbare filer når et kommandonavn tastes inn til shellet

```
studssh$ echo $PATH
/opt/bin:/local/lu/bin:/local/gnu/bin:/local/bin:/usr/X11R6/bin:/usr/bin:/bin:..
studssh$ type ls          # type gir hvilket program shellet starter
ls is /bin/ls
```

Når du taster inn

```
$ ls
```

leter bash igjennom alle mappene i `$PATH` etter en fil med navn `ls` helt til det finner en i den nest siste (`/bin`) og kjører den. Hvis du gjør `PATH` tom:

```
$ PATH=""
```

vil shellet ikke finne vanlige kommandoer som `mv` og `ls` fordi mappen `/bin` ikke er med i `$PATH`.

3.7 Prosesser

Hver gang vi starter et program lager Linux en uavhengig prosess:

```
$ emacs
```

Starter vi fra et shell, venter shellet på at programmet skal bli ferdig.

```
$emacs&
```

```
$
```

Dette starter emacs som en bakgrunnsprosess

kommando	virkning
<code>\$ ps</code>	Lister shellets prosesser
<code>\$ ps aux (eller -Al)</code>	Alle prosesser
<code>\$ top</code>	Dynamisk ps -aux
<code>\$ kill 1872</code>	Drep prosess med PID 1872
<code>\$ kill -9 1872</code>	Forsert drap av 1872
<code>\$ time mittScript.bash</code>	Måler tidsforbruket

3.8 Apostrofer

I bash brukes 3 forskjellige typer apostrofer `'`, ``` og `"` som alle har forskjellig betydning:

```
$ 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!
fil1 fil2 fil.txt
```

Huskeregel

```
' -> / -> Linux      -> vet hva man får
' -> \ -> Windows -> aner ikke hva man får
```

Et mer robust alternativ når en kommando skal utføres:

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

Dette gir også mer lesbar kode. I tillegg kan man ikke ha uttrykk med apostrofer inni hverandre:

```
rex$ line='grep 'whoami' /etc/passwd'
Usage: grep [OPTION]... PATTERN [FILE]...
Try 'grep --help' for more information.
bash: /etc/passwd: Permission denied
```

Men dette går fint med `$()` syntaksen:

```
rex$ line=$(grep $(whoami) /etc/passwd)
rex$ echo $line
haugerud:x:5999:9002:Hårek Haugerud,,,:/home/haugerud:/bin/bash
```

Så generelt anbefales det å bruke denne. Om man prøver følgende uttrykk med apostrofer inni hverandre:

```
haugerud@studssh:~/mappe$ var='/bin/ls 'pwd' '
```

Det vil si, det ga ingen feilmeldinger. Men om man ser nøyer på hva resultatet ble

```
haugerud@studssh:~/mappe$ echo $var
fil.txtpwd
haugerud@studssh:~/mappe$ pwd
/iu/nexus/ua/haugerud/mappe
haugerud@studssh:~/mappe$ ls
fil.txt
```

så ser vi at de ikke fungerte som ønsket. Igjen er det bedre å bruke `$()` konstruksjonen:

```
haugerud@studssh:~/mappe$ var=$(/bin/ls $(pwd))
haugerud@studssh:~/mappe$ echo $var
fil.txt
```

3.9 Tilordne output fra kommando til en variabel

```
$ mappe='pwd'
$ echo $mappe
/iu/nexus/ud/haugerud
$ tall='seq 5 10'
$ echo $tall
5 6 7 8 9 10
```

Anbefalt alternativt:

```
$ mappe=$(pwd)
$ tall=$(seq 5 10)
$ echo $tall
5 6 7 8 9 10
$ tall=$(echo {5..10})
$ echo $tall
5 6 7 8 9 10
```

3.10 Omdirigering (viktig!)

Den store fleksibiliteten det gir å kunne dirigere datastrømmer til og fra filer og programmer har alltid vært en sentral og nyttig egenskap ved Linux.

De fleste Linux programmer og kommandoer har alltid tre åpne kanaler:

nummer	navn	fullt navn	funksjon	default
0	stdin	standard in	input kanal	fra tastatur
1	stdout	standard out	output kanal	til skjerm
2	stderr	standard error	kanal for feilmelding	til skjerm

Dataene som strømmer inn og ut av disse kanalene (streams) kan omdirigeres til og fra filer og programmer.



```

$ echo hei
hei
$ echo hei > fil1
$ cat fil1
hei
$ echo hei2 >> fil1
$ cat fil1
hei
hei2

```

omdirigering	virkning
> fil.txt	omdirigerer stdout til fil.txt. Overskriver
>> fil.txt	legger stdout etter siste linje i fil.txt
>& fil.txt	sender også stderr til fil.txt
2> err.txt	sender stderr til err.txt
2> /dev/null	stderr sendes til et 'sort hull' og forsvinner
> fil.txt 2> err.txt	stdout -i fil.txt stderr -i err.txt
find / -name passwd 2>&1 grep -v Permission	sender stderr til samme kanal som stdout
prog < fil.txt	sender fil til stdin for program

Flere eks:

```

$ ehco hei
ehco: Command not found
$ ehco hei >& fil1
$ cat fil1
ehco: Command not found
$ echo hei > fil1 2> err.txt # Sender error til err.txt
$ mail haugerud < fil1

```

Konstruksjonen 2>&1 betyr: send stderr til samme kanal som stdout. Man limer da stderr-kanalen til kommandoen på samme linje inn i stdout-kanalen. I eksempelet over vil det føre til at man kan grep'e på både stdout og stdin fra find. Uten 2>&1 ville man bare kunne grep'e på stdout. Denne konstruksjonen kan også brukes som alternativ til >&, kommandoen

```
ls /tmp/finnesikke > fil.txt 2>&1
```

vil sende stderr til fil.txt (men må stå etter >).

Konstruksjonen 1>&2 betyr: send stdout til samme kanal som stderr. Inne i et script vil

```
echo "Error!" 1>&2
```

sende feilmeldinger til stderr, slik programmer vanligvis gjør. Vanligvis sender echo output til stdin, men 1>&2 gjør at stdout istedet sendes til stderr.

3.11 Omdirigering til og fra kommandoer; pipes

En pipe | er et data-rør som leder et programs stdout til et annet programs stdin.
Uten pipe:

```
$ ps aux > fil
$ more fil
```

Med pipe:

```
$ ps aux | more
```

Dette gjør at man kan kombinere alle Linux-kommandoene på en rekke måter. Noen eksempler:

```
$ ps aux | grep haugerud | more
$ cat /etc/passwd | sort > fil.txt
$ sort /etc/passwd > fil.txt
$ ps aux | awk '{print $1}' | sort | uniq | wc -l
$ ps -eo user | sort | uniq | wc -l
```

Forklaring til det siste eksempelet: ps -eo user gir bare brukernavnet i ps-listingen. sort sorterer listen med brukernavn alfabetisk. uniq fjerner identiske linjer, men kun de som kommer etter hverandre, derfor sort først. wc -l returnerer antall linjer. En slik 'pipeline' gir dermed antall brukere som kjører prosesser på maskinen.

3.12 Piping standard error

Når man setter en pipe etter en kommando, er det bare stdout som sendes dit. Men ved hjelp av konstruksjonen `|&` sender man også stderr videre:

```
$ ehco hei | grep found
No command 'ehco' found, did you mean:
  Command 'echo' from package 'coreutils' (main)
ehco: command not found
$ ehco hei |& grep found
No command 'ehco' found, did you mean:
ehco: command not found
```

Finnes det en annen måte å gjøre det samme på?

3.13 Sub-shell

Konstruksjonen `(kommando;kommando)` gir et såkalt subshell. Da startes et nytt shell og man mottar den samlede output til stdout og stderr i shellet som kjører kommandoen. Det kan f.eks. brukes til å slå sammen output fra to filer:

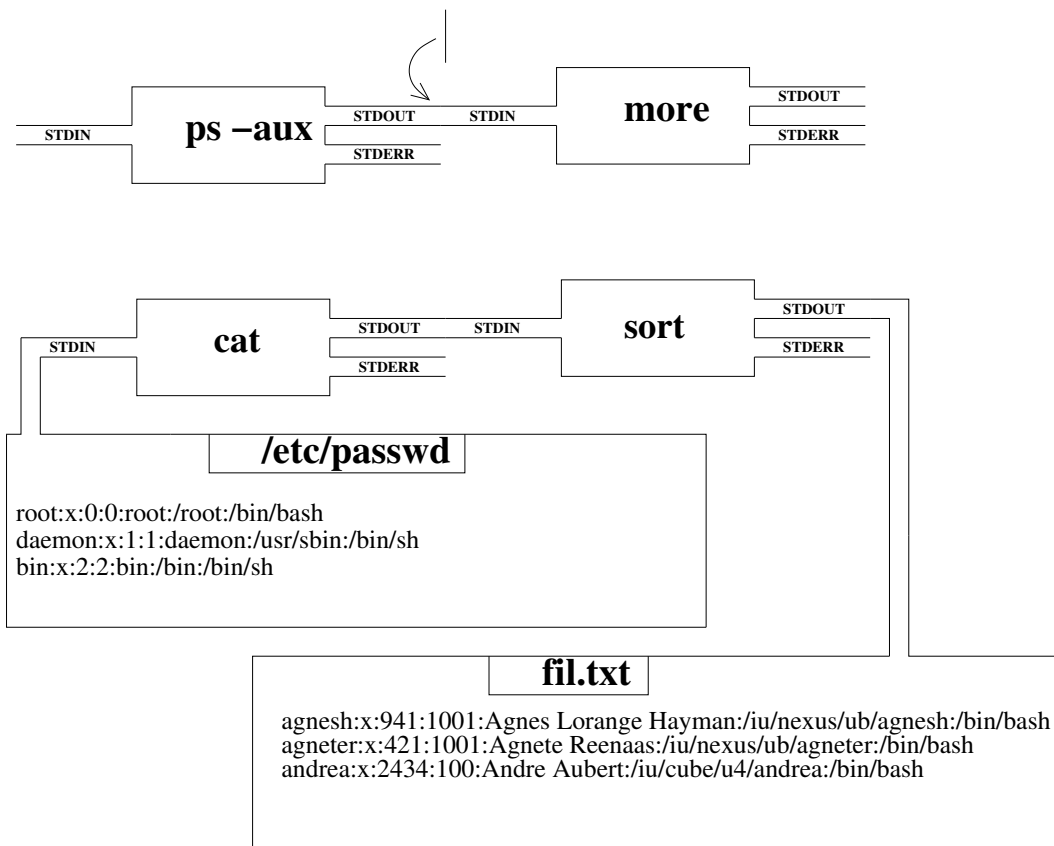


Figure 11: Øverst: `ps -aux` — `more`. Nederst: `cat /etc/passwd` — `sort` i `fil.txt`.

```
$ cat > fil1
en
to
tre
$ cat > fil2
tre
to
fire
$ sort fil2
fire
to
tre
```

Hvis man ønsker å skrive ut de to filene med en kommando, kan man gjøre

```
$ cat fil2;cat fil1
tre
to
fire
en
to
tre
```

Men anta at man ønsker å sortere output fra de to filene; da kan man prøve følgende:

```
cat fil1;cat fil2 | sort
en
to
tre
fire
to
tre
```

Men det som skjer er at bare fil2 blir sortert. Om man lager et sub-shell

```
$ (cat fil1;cat fil2)
en
to
tre
tre
to
fire
```

vil output fra dette subshellet samlet sendes til sort:

```
$ (cat fil1;cat fil2) | sort
en
fire
to
to
tre
tre
```

og man oppnår det man ønsket.

3.14 source

\$ emacs change
#!/bin/bash
cd /usr/bin
pwd

```
$ pwd
/
$ change
/usr/bin
$ pwd
/
```

Når scriptet change kjøres, starter en ny prosess; et nytt shell som utfører

```
cd /usr/bin
```

og avsluttes. All shell-info blir da borte (variabler og posisjon i filtreet).

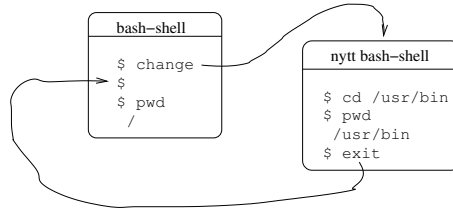


Figure 12: Når et script kjøres startes et helt nytt shell. Alt som har skjedd i dette shellet blir borte når scriptet avsluttes (exit utføres selvom det ikke står eksplisitt på slutten av scriptet).

Kommandoen **source** utfører linje for linje i argumentfilen uten å starte noe annet shell.

```
$ pwd
/
$ source change
/usr/bin
$ pwd
/usr/bin
```

I Bash er **.** og **source** ekvivalent:

```
$ . change    # samme som $ source change
```

3.15 Kommandoer brukt under forelesningen

Kommandoer³⁴

4 Forelesning 6/2-24(2 timer). Bash-scripting

4.1 Forelesningsvideoer

Denne forelesningen ble ikke holdt live, men opptak av forelesningene er samlet her. Opptak av forelesningen inndelt etter temaer:

linux4del1.mp4³⁵ (03:17) Slide: Innledning om shell-programmering

linux4del2.mp4³⁶ (10:52) Demo: If-tester

linux4del3.mp4³⁷ (04:37) Demo: Vanlige feil i if tester. Bruk doble apostrofer rundt variabler! (

³⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/forelesningsKommandoer/Fri07.02.2020.html>

³⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del1.mp4>

³⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del2.mp4>

³⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del3.mp4>

"\$var")

linux4del4.mp4³⁸ (02:05) Demo: Vær pinlig nøyaktig med mellomrom i bash-syntaks!

linux4del5.mp4³⁹ (07:23) Demo: Flere filtester, sammenligning og logiske operasjoner

linux4del6.mp4⁴⁰ (06:28) Demo: For-løkker

linux4del7.mp4⁴¹ (02:32) Demo: IFS, splitting på andre tegn enn mellomrom

linux4del8.mp4⁴² (03:46) Demo: Break og continue

linux4del9.mp4⁴³ (02:56) Demo: Numerikk i shellet

linux4del10.mp4⁴⁴ (04:55) Demo: Script og argumenter

linux4del11.mp4⁴⁵ (06:44) Demo: while-løkker

linux4del12.mp4⁴⁶ (04:51) Demo: /proc - et vindu mot operativsystem-kjernen

4.2 Sist

- Globale og lokale shell variabler
- Prosesser
- Apostrofer
- Omdirigering
- pipes

4.3 Shell-programmering

Vi kan lage script som er

- nye kommandoer
- oppstartsprogrammer/program som gjør systemarbeid
- installasjonsprogrammer
- daemons; prosesser som alltid går i bakgrunnen og utfører tjenester.

`$mittprog&` vil fortsette å kjøre etter logout.

Når du skriver script:

- begynn med et skjelett (som f. eks. kun behandler argumentene) og få det til å virke
- utvid med en detalj av gangen; test for hver gang
- test små deler med copy&paste til et kommandovindu

³⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del4.mp4>

³⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del5.mp4>

⁴⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del6.mp4>

⁴¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del7.mp4>

⁴²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del8.mp4>

⁴³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del9.mp4>

⁴⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del10.mp4>

⁴⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del11.mp4>

⁴⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux4del12.mp4>

Debugging:

- `bash -x mittscript` viser hva som skjer
- Legg inn linjer som `echo "Har nå kommet hit"`

Feilmeldinger fra `bash` er ofte kryptiske og misvisende, så det er vanskelig å finne feil i et stort script.

4.4 if-test

```
if test1
then
    kommando1
elif test2
then
    kommando2
else
    kommando3
fi
```

4.5 if-eksempel; fil-testing

```
#!/bin/bash

fil=$1
if [ -f "$fil" ]
then
    echo $fil er en fil
elif [ -d "$fil" ]; then
    echo $fil er en mappe
else
    echo $fil er hverken fil eller mappe
fi
```

Man **må** ha mellomrom før og etter [og før] i if-testene. Semikolon adskiller generelt to kommandoer på samme måte som et linjeskift.

Følgende script prøver å teste om det ikke er gitt noe argument:

```
#!/bin/bash

argument1=$1
if [ $argument1 = "" ]
then
    echo "Ingen argumenter"
fi
```

Men det gir en feilmelding når det kjøres uten argumenter

```
$ ./script
./script: line 5: [: =: unary operator expected
```

fordi det ikke er anførselstegn rundt \$argument1 og da ser bash en linje som ser ut som `if [ = "" ]` og gir feilmelding. Analogt vil testen `if [ -f ]` alltid slå til. Bruk derfor alltid anførselstegn rundt det som skal testes, slik at det blir en tom streng og dermed syntaktisk riktig om variabelen du tester ikke eksisterer eller er tom.

4.6 Flere filtester og sammenligning

Filtest	Sann hvis
-L fil	fil er en link
-r fil	fil er lesbar
-w fil	fil er skrivbar
-e fil	fil eksisterer
-x fil	fil er eksekverbar
s1 = s2	strengene s1 og s2 er like
s1 != s2	strengen s1 og s2 er forskjellige
x -eq y	heltallene x og y er like
x -lt y	x er mindre enn y ⁴⁷
x -gt y	x er større enn y
t1 -a t2	Logisk og - sann hvis t1 OG t2 er sanne
t1 -o t2	Logisk eller - sann hvis t1 ELLER t2 er sanne

4.7 Logiske operatorer

Operator	betydning
!	logisk NOT
-a	logisk AND
-o	logisk OR

4.8 for-løkke

```
for variable in list
do
    kommandoer
done
```

hvor list er en liste av ord adskilt av mellomrom. F. eks. gir

```
for var in h1 h2 h3
do
    echo $var
done
```

til output

```
h1
h2
h3
```

Info om nøkkelord som for

```
haugerud@studssh:~$ type for
for is a shell keyword
```

kan man få med

```
haugerud@studssh:~$ help for
for: for NAME [in WORDS ... ] ; do COMMANDS; done
    Execute commands for each member in a list.
```

Eksempler på list eller WORDS:

- \$(ls)
- \$(cat fil)
- *
- dir/*.txt
- \$filer # (filer="fil1 fil2 fil3")

En liste splittes med mellomrom. Variabelen IFS (Internal Field Separator) kan settes om man ønsker å splitte på et annet tegn. Det skjer ved at IFS-tegnet byttes ut med mellomrom når verdien av en shellvaribael benyttes. Den må fjernes med unset om man ønsker å splitte på mellomrom igjen.

```
$ var=fil1:fil2:fil3
$ echo $var
fil1:fil2:fil3
$ IFS=:
$ echo $var
fil1 fil2 fil3
$ for fil in $var
> do
> echo $fil
> done
fil1
fil2
fil3
```

Å bruke kolon til å splitte er nyttig om man ønsker å løpe igjennom PATH eller en linje i passordfilen. Med noen opsjoner til cat, kan man se hva som skjuler seg bak \$IFS:

```
haugerud@studssh:~$ echo "$IFS" | cat -vet
^I$
$
```

Dette betyr at IFS splitter på de tre tegnene `space`, `tab`, `newline`, noe man kan se i bash sin manual-side om man søker på IFS.

Fra og med bash versjon 2.0 kan man også skrive vanlige for-løkker med en syntaks som ligner mer på Java:

```
for (( i=1;i < 30;i++ ))
do
    echo $i
done
```

4.9 Break og Continue

Break og Continue kan brukes mellom do og done til å avbryte for og while løkker:

continue Fortsett med neste runde i for/while løkke.

break Avslutt for/while løkke; hopp til etter done

break n Hopp ut n nivåer

4.10 Numerikk

Tungt å bruke **expr** som er eneste alternativ i Bourne-shell:

```
a='expr $a + 1'           # increment a
a='expr 4 + 10 \* 5'       # 4+10*5
```

Enklere innenfor `(( ))` for da kan Java-syntaks for numerikk brukes(men bare i bash 2.x).

```
$ (( x = 1 ))
$ echo $x
1
$ (( x += 1 ))
$ echo $x
2
$ (( total = 4*$x + x )) # Virker med x uten $ foran!
$ echo $total
10
```

Den numeriske testen

```
if ((x < y))
```

er et Bash 2.x alternativ til

```
if [ $x -lt $y ]
```

Den beste løsningen er å bruke syntaksen med doble paranteser, (()), selvom syntaksen (for å være bakoverkompatibel) er litt uvanlig.

4.11 Script og argumenter

Argumenter blir lagret i spesielle variabler.

Kjøres scriptet argscript.bash med tre argumenter:

```
$ argscript.sh fil1 fil2 fil3
```

vil følgende variabler bli satt:

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

4.12 Argumenter i for-løkke og exit-verdier.

```
$ cat forarg
#!/bin/bash

if [ $# -lt 1 ]
then
    echo No arguments
    exit 1          # Avsluttet scriptet
fi

for arg in $*
do
    echo $arg was an argument
done

echo total number: $#

$ forarg hei du
hei was an argument
du was an argument
total number: 2
$ echo $?
0
$ forarg
```

```
No arguments
$ echo $?
1
```

Variabelen `$?` inneholder exit-verdien til programmet/kommandoen som sist ble utført. Det finnes andre variabler som er definert, som de to følgende:

```
echo "PID = $$, navn: $0"
```

4.13 while

syntaks:

```
while test
do
    kommandoer
done
```

Eksempel; alternativ måte å behandle argumenter: *bedre hvis "strenger med mellomrom" er blant argumentene*

```
#!/bin/bash

while [ $# -gt 0 ] # Så lenge det er noe i $*
do
    echo "arg: $1"
    shift          # skyver ut $1 og legger neste argument i $1
done
```

Deamon-eksempel

```
#!/bin/bash

while [ "true" ] # evig løkke; en streng er alltid 'sann'
do
    echo "Running $0, a process with PID $$"
    sleep 5      # sover i 5 sekunder
done
```

4.14 /proc - et vindu til kjernen

Linux tilbyr oss en enkel måte å undersøke tilstanden til operativsystemet på. Det finnes nemlig en mappe `/proc` som egentlig ikke finnes på harddisken, men som likevel inneholder "filer" som vi kan lese innholdet fra. Hver gang vi åpner en av filene i `/proc`, kjøres en metode i kjernen som skriver ut innholdet der og da. Man får altså et øyeblikksbildet av hva som skjer. Her er en liste over interessante filer og mapper i `/proc`:

- `cpuinfo`
- `meminfo`
- `interrupts`
- Hver prosess har en egen mappe som har samme navnsom prosessens PID. Spennende filer i en slik mappe kan være:
 - `status` - Navn og status på prosessen
 - `stat` - Teller prosessorbruk, minnebruk osv. se ‘man proc’
- `uptime`
- `version`
- `net/`

5 Forelesning 13/2-24(2 timer). Bash-scripting

5.1 Forelesningsvideoer

Denne forelesningen ble ikke holdt live, men opptak av forelesningene er samlet her. Opptak av forelesningen inndelt etter temaer:

linux5del1.mp4⁴⁸ (07:32) Demo: Passord-kryptering
 linux5del2.mp4⁴⁹ (06:17) Slides: Passord-cracking
 linux5del3.mp4⁵⁰ (05:23) Demo: find
 linux5del4.mp4⁵¹ (04:11) Demo: Sed og sort
 linux5del5.mp4⁵² (06:43) Demo: head og tail
 linux5del6.mp4⁵³ (03:20) Demo: cut
 linux5del7.mp4⁵⁴ (01:18) Demo: Input fra bruker til script
 linux5del8.mp4⁵⁵ (03:29) Demo: Lese en fil linje for linje med while og read
 linux5del9.mp4⁵⁶ (03:25) Demo: Lese passordfilen linje for linje med while og read og trekke ut kolonner
 linux5del10.mp4⁵⁷ (06:23) Demo: Gå igjennom ps aux linje for linje med while og read og trekke ut kolonner
 linux5del11.mp4⁵⁸ (06:45) Demo: Array i bash
 linux5del12.mp4⁵⁹ (04:02) Demo: Et vanlig problem med pipe til while og read
 linux5del13.mp4⁶⁰ (05:00) Demo: Assosiativ array

⁴⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del1.mp4>

⁴⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del2.mp4>

⁵⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del3.mp4>

⁵¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del4.mp4>

⁵²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del5.mp4>

⁵³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del6.mp4>

⁵⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del7.mp4>

⁵⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del8.mp4>

⁵⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del9.mp4>

⁵⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del10.mp4>

⁵⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del11.mp4>

⁵⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del12.mp4>

⁶⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del13.mp4>

linux5del14.mp4⁶¹ (05:56) Demo: Funksjoner og parametre
linux5del15.mp4⁶² (04:51) Demo: Signaler og trap

5.2 Passord-kryptering

Alle brukere på en Linux server har en linje i `/etc/passwd`. For eksempel:

```
rex:~$ grep haugerud /etc/passwd
haugerud:x:285:1001:Hårek Haugerud,,,:/home/haugerud:/bin/bash
```

Tidligere stod det en hash av passordet der istedet for x i andre kolonne. Denne hashen er en kryptert versjon av passordet og ligger nå i `/etc/shadow` og kan bare leses av root. Det er kun hashen som lagres og dette gjør at man kan logge inn på en datamaskin uten at passordet er lagret i klartekst noe sted. Om en slik liste med passord i klartekst blir lekket, ville det vært katastrofe. Men det er også alvorlig om en liste med passord-hashter blir fritt tilgjengelig.

```
rex:~$ sudo grep haugerud /etc/shadow
[sudo] passord for haugerud:
haugerud:$6$WXQf3H3AUREz8y$IRYwxMcpK/aTX4oF.xEJro1.Va7cjGY4V.93wkKc3Tcd9JV0mPIPjyqBuljB3UPw6.VPJx/ymicJlxs51Bv.:1783
```

Hashen kan (med rett 'salt') genereres med kommandoen `mkpasswd`:

```
rex:~$ mkpasswd -m sha-512 -S WXQf3H3AUREz8y
Passord:
$6$WXQf3H3AUREz8y$IRYwxMcpK/aTX4oF.xEJro1.Va7cjGY4V.93wkKc3Tcd9JV0mPIPjyqBuljB3UPw6.VPJx/ymicJlxs51Bv.
```

Hvilken kryptografisk hashing-metode som er brukt kan sees av de første tegnene i hashen:

Første tegn	Algoritme
To tegn	DES (13 totalt)
\$1\$	md5
\$5\$	sha256
\$6\$	sha512

Saltet er de etterfølgende tegnene frem til \$-tegnet. Dette saltet gjør det vanskeligere å bruke enkelte brute force metoder som rainbow tables for å cracke passord om man kjenner en hash for et passord. Ved login skjer følgende:

1. Man taster passord
2. Login-shellet krypterer passordet
3. Sjekker mot `/etc/shadow`
4. Hvis likt → login

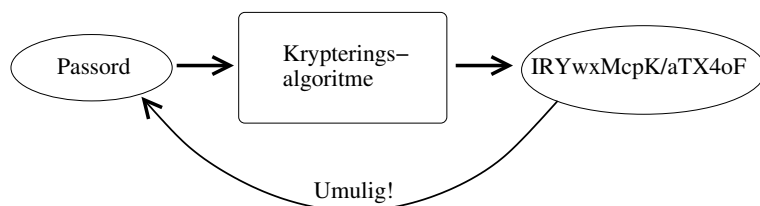


Figure 13: Passordkryptering

For 4 år siden lå alle bruker-hashene i `/etc/shadow`, men nå autentiseres den som logger seg inn på studssh ved hjelp av PAM (pluggable authentication module) mot AD (Active Directory) sentralt på OsloMet.

5.2.1 Hashing-algoritmer

Med `mkpasswd` kan man velge hvilken hashing-metode man vil bruke:

```
$ mkpasswd -m help
Available methods:
des standard 56 bit DES-based crypt(3)
md5 MD5
sha-256 SHA-256
sha-512 SHA-512
```

Dette er enveisalgoritmer som for et gitt passord generere en entydig lengre streng av tegn. DES var tidligere standard, men ble så avløst av MD5 som er noe bedre, og nå er sha-512 standard metode for Linux. De samme prinsippene gjelder for Windows-innlogging, der lagres hash-strengene i SAM-databasen på en lokal Windows-maskin eller på en sentral server i Active Directory. Tidligere var LM-hash(DES) standard, men den ble etterhvert avløst av NTLM-hash(MD5) og etterhvert av kraftigere algoritmer som AES(Advanced Encryption Standard).

5.2.2 Passord-cracking

Hvis man har tilgang til shadowfilen på Linux eller SAM-databasen på Windows, kan et crack-program kryptere alle ord og kombinasjoner av ord i en ordbok og sammenligne med de krypterte passordene. Hvis ett av de riktige passordene velges, vil det avsløres ved at det gir en av hashene. Slike program kan teste hundretusner av passord per sekund, slik at passord fra ordbøker veldig raskt kan knekkes. Jo lengre passordene er og jo flere tegn som brukes i passordene, desto mer tidkrevende er det for passord-cracker program å teste ut alle mulige kombinasjoner.

Hvis man går ut ifra 52 tegn (a-z og A-Z), 10 tall og 32 spesialtegn, har man totalt 94 mulige tegn i et passord. Og om man har en kraftig GPU-server kan man regne ut en million hasher i sekundet for algoritmer som sha512, mange flere for enklere algoritmer. For åtte tegn i passordet er det $8^{62} = 2.18 \cdot 10^{14}$ eller 218 billioner mulige kombinasjoner. For å finne hash'ene for alle mulige

⁶¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del14.mp4>

⁶²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux5del15.mp4>

kombinasjoner, tar det da i underkant av 7 år om man regner ut en million hash i sekundet. Tid for forskjellige lengder av passord er vist i følgende tabell:

Lengde	tid (36 tegn)	tid (62 tegn)	tid (94 tegn)
4	1.7 sekunder	15 sekunder	78 sekunder
6	36 minutter	16 timer	192 timer
8	32 dager	6.9 år	193 år
10	116 år	26.6 tusen år	1.7 millioner år

Til sammenligning er det ca en million ord i en norsk ordbok som tar med alle varianter av norske ord, slik at det kun tar ett sekund å finne et passord som er med i en ordbok om man kjenner hashen for passordet.

Man ser at lengden på passordet er veldig viktig for hvor raskt det kan være å gjette ved hjelp av et brute force angrep på en hash. I tillegg hjelper det at store bokstaver og spesialtegn tas med.

5.3 find

Denne kommandoen kan brukes til å finne filer på systemet. Det er mulig å spesifisere en rekke kriterier. Ønsker man å finne alle filer (og mapper) i hjemmekatalogen som har filendelse c, kan man bruke

```
haugerud@studssh:~$ find ~ -name "*.c"
/iu/nexus/ua/haugerud/os/funk.c
/iu/nexus/ua/haugerud/os/old/main.c
/iu/nexus/ua/haugerud/os/old/sum.c
```

For å finne alle filer og mapper som ble endret mer nylig enn 4 februar 2019 kl 19:55, kan man gjøre

```
haugerud@studssh:~$ find . -newermt "4 Feb 2019 19:55" -ls
  9714      4 drwxr-xr-x  20 haugerud drift      4096 feb.  5 10:28 .
 29479      4 -rw-----   1 haugerud drift      318 feb.  5 10:28 ../Xauthority
 29374      4 -rw-r--r--   1 haugerud drift       10 feb.  5 20:07 ./fil
```

der `-ls` gir en mer detaljert listing. Tilsvarende kan man finne alle filer som har blitt endret mellom to tidspunkt med

```
haugerud@studssh:~$ find . -type f -newermt "29 Jan 2019 19:55" ! -newermt "29 Jan 2019 22:55" -ls
 29422      4 -rwx-----   1 haugerud drift       56 jan. 29 22:40 ./mappe/arg.sh~
 29423      4 -rwx-----   1 haugerud drift       58 jan. 29 22:42 ./mappe/arg.sh
```

hvor `-type f` gir kun filer.

5.4 sed

Kommandoen `sed` kan brukes til å bytte ut forekomster av ord i setninger:

```
echo dette er en test | sed s/test/fisk/  
echo test og test | sed s/test/fisk/  
echo test og test | sed s/test/fisk/g
```

5.5 sort

I tillegg til at man kan sende output fra en Linux-kommando inn til input for en annen, kan man også legge til en rekke opsjoner. På denne måten kan man få til veldig mye med enlinjers sammensatte kommandoer. Manualsiden til sort avslører at opsjonen -k gjør at man kan velge hvilken kolonne man vil sortere med hensyn på, mens -n gjør at man sorterer numerisk slik følgende eksempel viser. Utgangspunktet er filen **biler** som ser slik ut:

```
$ cat biler  
student bmw 500000  
haugerud berlingo 150000  
kyrre elbil 90000
```

Denne filen kan man nå sortere som man ønsker med de rette opsjoner.

5.5.1 Sortert alfabetisk

```
$ sort biler  
haugerud berlingo 150000  
kyrre elbil 90000  
student bmw 500000
```

5.5.2 Sortert alfabetisk etter andre kolonne

```
$ sort -k 2 biler  
haugerud berlingo 150000  
student bmw 500000  
kyrre elbil 90000
```

5.5.3 Sortert numerisk etter tredje kolonne

```
$ sort -n -k 3 biler  
kyrre elbil 90000  
haugerud berlingo 150000  
student bmw 500000
```

Default sender sort output til shellet, hvis man ønsker at resultatet skal lagres i en fil må man be om det

```
sort -n -k 3 biler > sortertFil
```

5.6 head og tail

Hvis man ønsker å se kun de 6 første linjene av en utgave av passordfilen på studssh sortert etter femte kolonne kan man bruke **head** for å få til det:

```
studssh:~$ sort -t: -k 5 /etc/passwd | head -n 6
aasej:x:2748:1001:Aase Jenssen:/iu/nexus/uc/aasej:/bin/bash
s137153:x:2603:100:Aasmund Solberg:/iu/cube/u2/s137153:/bin/bash
s103726:x:1089:100:Abdi Farah Ahmad:/iu/cube/u3/s103726:/bin/false
s133988:x:1695:100:Abdi Hassan Abdulle:/iu/cube/u2/s133988:/bin/bash
s123860:x:1090:100:Abdinasir Omar Kahiye:/iu/cube/u2/s123860:/bin/bash
s141546:x:3449:100:Abdiqadir Said Jama:/iu/cube/u3/s141546:/bin/false
```

Legg merke til at **-t:** gjør at tegnet **:** betraktes som skilleledd mellom kolonnene. Evalueringen av en slik pipeline går fra venstre til høyre så hvis man istedet ønsker å se en sortert utgave av de 6 første linjene, får man det med:

```
studssh:~$ head -n 6 /etc/passwd | sort -t: -k 5
bin:x:2:2:bin:/bin:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
games:x:5:60:games:/usr/games:/bin/sh
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
sys:x:3:3:sys:/dev:/bin/sh
```

Kommandoen **tail** gir i motsetning til **head** de siste linjene. En spesielt nyttig bruk av **tail** er for å se på slutten av log-filer. Hvis man i tillegg tilføyer opsjonen **-f** vil man kontinuerlig følge med på om det kommer nye linjer til logfilen, for eksempel slik:

```
sudo tail -f /var/log/auth.log
```

5.7 cut

Cut brukes til å klippe ut deler av linjer. Cut leser fra standard input. En vanlig anvendelse er å klippe ut enkelte kolonner fra en tabell.

```
$cat /etc/passwd | cut -d: -f2 | tail -n 4
group16
mroot
noob
munin
```

Her ”pipes” innholdet i **/etc/passwd** til **cut**, som bruker **:** som kolonneskille (gitt ved **-d:**) og som klipper ut kolonne 2 (gitt ved **-f2**). Dette pipes videre til **head**, som viser de fire nederst linjene.

```
$cat /etc/passwd | cut -c -3 | tail -n 4
gro
```

```
mro
noo
mun
```

Med switchen `-c` angir vi at vi ønsker å klippe ut ”characters”. Med `-n`, der n er et tall, klipper vi ut bare de n første tallene. `n-` tar tegnene fra tegn n og ut linja, `n-m` tar alle tegnene mellom tegn n og m .

5.8 Input fra bruker

```
#!/bin/bash

echo -en "svar: \a" # -n dropper linjeskift
read answer
echo "Du svarte $answer"
```

opsjonen `-e` muliggjør bruk av kontrolltegn som `\a`, som gir ett pip.

5.9 Lese filer og output med `while` og `read`

Veldig ofte ønsker man å gå igjennom og prosessere en fil eller tekstoutput fra en kommando linje for linje. Da kan `while` brukes til å lese input og `read` til å behandle linje for linje, som i skriptet `read.sh`:

```
#!/bin/bash

i=0
while read LINE
do
    (( i++ ))
    echo "Linje nr $i: $LINE"
done
```

`read` leser linje for linje fra STDIN slik at

```
$ read.sh < /etc/passwd
```

vil lese passordfilen. `read LINE` returnerer 0 (alt OK) helt til EOF og da stopper `while`.

IFS kan endre hvordan `read` leser inn dataene og filen kan også sendes til `read` inne i skriptet::

```
#!/bin/bash

#haugerud:x:285:102:Hårek Haugerud:/iu/nexus/ud/haugerud:/bin/bash
IFS=:
while read brnavn x ID GR NAVN HOME SHELL
do
    echo "$brnavn: $NAVN"
done < /etc/passwd
```

read leser fra STDIN og dit kan linjene også sendes med en pipe:

```
#!/bin/bash

#haugerud 16662 0.0 0.1 2256 1280 pts/2 S 12:52 0:00 /bin/bash

# Sender all output fra ps aux til \verb+read+:
ps aux |
while read bruker PID x x x x x x x prog
do
    if [ "$bruker" = "haugerud" ]
    then
        echo "ProsessID = $PID $prog"
    fi
done
```

5.10 Arrays

Et array kan i bash fylles inn med elementer uten å deklarere arrayet først:

```
linux:~$ tall[0]=null
linux:~$ tall[1]=en
linux:~$ tall[2]=to
linux:~$ tall[3]=tre
```

Den mest naturlige måten å skrive ut et array-element på fungerer ikke:

```
linux:~$ echo $tall[1]
null[1]
linux:~$ echo $tall[2]
null[2]
linux:~$ echo ${tall[2]}
to
```

Men man må legge inn et sett med krøll-parenteser rundt elementnavnet for å få skrevet ut. Man kan også definere et array ved å skrive inn strenger innenfor en parentes adskilt av mellomrom:

```
$ tall=(null en to tre)
$ echo ${tall[to]}
null
$ echo ${tall[2]}
to
$ tall[4]=fire
$ echo ${#tall[@]} # Antall elementer
5
$ echo ${tall[@]} # Alle verdier
null en to tre fire
$ echo ${!tall[@]} # Index
0 1 2 3 4
$ for t in ${!tall[@]}
```

```

> do
> echo "Tall nr $t er ${tall[$t]}"
> done
Tall nr 0 er null
Tall nr 1 er en
Tall nr 2 er to
Tall nr 3 er tre
Tall nr 4 er fire

```

5.11 Et vanlig problem med pipe til while og read

Et naturlig forsøk på å sende output til en while-read løkke er følgende(se avsnittet nedenfor om hvordan man lager et array):

```

#!/bin/bash
i=0
ps aux | awk '{print $2}' |
while read pid
do
    (( i++ ))
    pidArr[$i]=$pid
done
echo Antall elementer: ${#pidArr[@]}

```

men om man kjører dette, får man følgende resultat:

```

$ ./pipe.sh
Antall elementer: 0

```

Dette skyldes at når den kommer etter en pipe vil while-konstruksjonen startes i en annen prosess og variabler som blir laget i denne prosessen (arrayet i eksempelet over) vil forsvinne når while-prosessen avsluttes. Dette kan løses ved å sende en fil direkte til konstruksjonen eller mellomlagre output fra pipe'n i en fil:

```

#!/bin/bash
ps aux | awk '{print $2}' > tmp.txt
i=0
while read pid
do
    (( i++ ))
    pidArr[$i]=$pid
done < tmp.txt
echo Antall elementer: ${#pidArr[@]}

```

Da vil ønsket resultat oppnås:

```

$ ./pipe.sh
Antall elementer: 742

```

Med den spesielle konstruksjon `<(kommando)` er det også mulig å direkte sende output fra `kommando` til `while-read` løkken som om det var en fil:

```
#!/bin/bash
i=0
while read pid
do
    (( i++ ))
    pidArr[$i]=$pid
done < <(ps aux | awk '{print $2}')
echo Antall elementer: ${#pidArr[@]}
```

Det gir samme ønskede resultat, dataene blir lagret i arrayet og kan brukes etter at løkken er fullført. Konstruksjon `<(kommando)` kalles "Process Substitution".

5.12 Assosiative array

Assosiative array har, istedet for tall, tekst-strenger som indeks. Et assosiativ array må deklarereres før bruk i bash:

```
$ declare -A mann
$ mann[eva]=adam
$ mann[kari]=per
$ mann["Gunn Kari"]="Per Olav"
$ echo ${#mann[@]}
3
$ echo ${mann[@]}
adam Per Olav per
$ echo ${!mann[@]}
eva Gunn Kari kari
$ for k in "${!mann[@]}"
> do
> echo "$k sin mann er ${mann["$k"]}"
> done
eva sin mann er adam
Gunn Kari sin mann er Per Olav
kari sin mann er per
```

5.13 funksjoner

En funksjon (function) brukes nesten på samme måte som et selvstendig script. Må inkluderes først i scriptet.

```
#!/bin/bash

users()          #deklarasjon av funksjon
{
    date          #skriver ut dagens dato
    who # Må ha linjeskift før }
```

```
}

users # kall paa en funksjon; ETTER deklarasjon
```

5.14 funksjoner og parametre

Parametre overføres som til bash-script. Og som for script kan kun exit-verdier (tall) returneres, men med return og ikke exit.

```
#!/bin/bash

findUser()          #deklarasjon av funksjon
{
    echo "funk-arg: $*"
    local bruker # Lokal variabel
    bruker=$1     # 1.u argument, dette er en lokal $1, uavhengig av den $1
                  # som er 1. argument til hovedscriptet
    funn=$(grep ^$bruker /etc/passwd)
    if [ "$funn" ]
    then
        return 0 # Alt ok
    fi
    return 1
}

# Hovedprog 'user.sh', syntaks: $ user.sh bruker1 bruker 2 .....
echo "Script-arg: $*"
for user in $*
do
    echo -e "\nLeter etter $user" # -e tillater \n
    findUser $user               # $user blir $1 i prosedyren
    if [ $? = 0 ]                # Returnverdi fra findUser i $?
    then
        echo "$user finnes"
        echo $funn               # $funn er global
    else
        echo "$user finnes ikke"
    fi
done

echo -e "\nScript-arg: $*"

#BUG: $ user.sh haug -> slår til på linjen haugerud i /etc/passwd
# Kan bruke 'while read'-konstruksjon for å trekke ut brukernavn fra linjene
```

Funksjoner kan også defineres direkte i et terminalvindu, men forsvinner når shelllet avsluttes. Man kan lagre egne funksjoner i en fil, f. eks. funk.bash og inkludere funksjonene i flere script ved å starte scriptene med som følger:

```
#!/bin/bash

. funk.bash # Alternativt 'source funk.bash' I begge tilfeller tilsvarer
```

```
# det å taste inn all koden i filen funk.bash inn i
# begynnelsen av dette scriptet
```

5.15 Signaler og trap

En prosess kan stoppes av andre prosesser og av kjernen. Det gjøres ved å sende et signal. Alle signaler bortsett fra SIGKILL (kill -9) kan stoppes og behandles av bash-script med kommandoen trap. Følgende script kan bare stoppes ved å sende (kill -9) fra et annet shell.

```
#!/bin/bash

# Definisjoner fra
# /usr/src/linux-headers-3.13.0-106/arch/x86/include/uapi/asm/signal.h

#define SIGHUP      1      /* Hangup (POSIX).  */
#define SIGINT      2      /* Interrupt (ANSI). */
#define SIGKILL     9      /* Kill, unblockable (POSIX). */
#define SIGTERM     15     /* Termination (ANSI). */
#define SIGTSTP     20     /* Keyboard stop (POSIX). */

trap 'echo -e "\rSorry; ignores kill -1 (HUP)\r"' 1
trap 'echo -e "\rSorry; ignores kill -15 (TERM)\r"' 15
trap 'echo -e "\rSorry; ignores CTRL-C\r"' 2
trap 'echo -e "\rSorry; ignores CTRL-Z\r"' 20
trap 'echo -e "\rSorry; ignores kill - 3 4 5\r"' 3 4 5
trap 'echo -e "\rCannot stop kill -9\r"' 9

while [ true ]
do
    echo -en "\a quit? Answer y or n: "
    read answer
    if [ "$answer" = "y" ]
    then break
    fi
done
```

5.16 Oversikt over shell-typer

Det finnes mange forskjellige typer shell og det shellet som nå er mest brukt i Linux-distribusjoner og som brukes her ved OsloMet er bash (vi skiftet fra tcsh i 2001). Det er en forbedring og utvidelse av det originale Linux-shellet Bourne Shell (sh)

Bourne-shell type		C-shell type	
sh	Bourne-shell, Det opprinnelige Linux-shell	csh	C-shell, C-syntaks
bash	Bourne-again-shell, forbedret sh	tcsh	forbedret csh, bedre interaktivt
ksh	Korn-shell, utvidet sh, mye fra csh		

- De fleste Linux/Linux system-script er skrevet i Bourne-shell, /bin/sh
- Bourne-again-shell (bash), er default Linux-shell

- bash kan kjøre alle Bourne-shell script
- bash er Free Software Foundation's (FSF) Linux-shell

Under Linux:

```
haugerud@studssh:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 feb. 19 2014 /bin/sh -> dash
```

Debian Almquist shell (dash) er et mindre og hurtigere shell enn bash. Det ligger nærmere det originale Bourne-shell og har for Ubuntu blitt brukt som Bourne-substitutt siden 2006.

5.17 Oppstartsfiler

Hver gang et nytt terminalvindu (f. eks. xterm) startes, startes bash og du får et prompt som du kan taste inn kommandoer ved. Hver gang bash startes leses en konfigurasjonsfil som ligger øverst i brukerens hjemmemappe og heter `.bashrc`. Alle kommandoer som står der blir utført. Hver gang du logger inn, utføres kommandoene i `/etc/profile` og `~/.bash_profile`.

Et nytt shell startes hver gang

- man logger inn på en maskin (ssh, telnet)
- et nytt terminalvindu åpnes (xterm)
- et nytt shell startes eksplisitt (\$ bash)

Shellet utfører først kommandoer i noen oppstartsfiler:

- `/etc/profile` ved hver innlogging, systemfil
- `~/.bash_profile` ved hver innlogging
- `/etc/bash.bashrc` for hvert nytt shell, men ikke ved innlogging, systemfil
- `~/.bashrc` for hvert nytt shell, men ikke ved innlogging

I disse filene kan f. eks. definisjoner av shell-variabler og alias'er legges.

Eksempel på innhold:

```
PS1="\h:\w$ "
alias move=mv
alias cp="cp -i"
```

Legges dette i `~/.bashrc` vil `move` alltid bety `mv` og promptet blir:

```
$ bash
studssh:~/www/os$ exit
exit
$ source ~/.bashrc
studssh:~/www/os$
```

source starter **ikke** et nytt shell, men utfører alt i det eksisterende shellet.

6 Forelesning 27/2-24(2 timer). Linux-VMer, Screen, ssh-keys, cron

6.1 Sist

- sort, head, tail, cut
- read while
- Array
- Associative array

6.2 Forelesningsvideoer

Denne forelesningen ble ikke holdt live, men opptak av forelesningene er samlet her. Opptak av forelesningen inndelt etter temaer:

linux6del1.mp4⁶³ (05:01) Demo: Root og sudo på Linux-VM
 linux6del2.mp4⁶⁴ (06:36) Slides: Linux-brukere og å lage en ny bruker på Linux-VM
 linux6del3.mp4⁶⁵ (05:32) Demo: Grupper og å lage en ny gruppe på Linux-VM
 linux6del4.mp4⁶⁶ (03:34) Rettighetsprinsipper på Linux
 linux6del5.mp4⁶⁷ (07:48) Demo: ssh-copy-id, passordløs innlogging
 linux6del6.mp4⁶⁸ (08:02) Demo: Direkte root aksess til Linux-VM for konto med sudo-rettigheter
 linux6del7.mp4⁶⁹ (07:47) Demo: Bakgrunnsjobber og screen
 linux6del8.mp4⁷⁰ (03:08) Demo: Å dele en screen med en eller flere andre
 linux6del9.mp4⁷¹ (04:22) Demo: Starte en prosess på en annen host med ssh
 linux6del10.mp4⁷² (04:06) Demo: scp, secure copy mellom ssh-hosts
 linux6del11.mp4⁷³ (04:35) Demo: backup med rsync
 linux6del12.mp4⁷⁴ (05:17) Demo: Kjøre script regelmessig med crontab
 linux6del13.mp4⁷⁵ (02:19) Demo: wget: Last ned websider

⁶³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del1.mp4>

⁶⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del2.mp4>

⁶⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del3.mp4>

⁶⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del4.mp4>

⁶⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del5.mp4>

⁶⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del6.mp4>

⁶⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del7.mp4>

⁷⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del8.mp4>

⁷¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del9.mp4>

⁷²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del10.mp4>

⁷³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del11.mp4>

⁷⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del12.mp4>

⁷⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del13.mp4>

6.3 root og sudo

På den virtuelle Linux-serveren, som egentlig er en ekstra isolert Docker container, dere har blitt tildelt i os-gruppene har dere rettigheter til å gjøre alt dere måtte ønske, som å installere programmer og lage nye brukere. Husk at Linux-VMene har offentlig IP og dermed kan bli utsatt for hacker-angrep fra hele verden.

Root er administratorbrukeren på Linux, som har alle rettigheter på systemet. For å bli root kan man logge inn direkte med root-passordet. Alternativt kan man logge inn med en vanlig brukerkonto og så bli root ved hjelp av kommandoen `sudo su` hvis man er tildelt rettigheter for å gjøre dette.

På Linux-VM vil du som den vanlige group-brukeren for eksempel ikke få lov til å se på shadowfilen:

```
$ whoami
group50
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

Men fordi group-brukeren er en såkalt sudo-user som har root-rettigheter om han ønsker det, kan han likevel se shadow-filen med kommandoen

```
sudo cat /etc/shadow
[sudo] password for group50:
```

etter å ha skrevet inn sitt vanlige passord. Det er også mulig for en sudo-user å få opp et root-shell ved å gjøre

```
sudo su
```

Man bør være forsiktig med å jobbe fra et root-shell, siden man da lettere glemmer at man har de allmektige root-rettighetene som potensielt kan føre til store feilgrep.

Om man jobber i et miljø med virtuelle maskiner og containere, noe som blir mer og mer vanlig, er ikke det å unngå å være logget på som root like viktig som før. Om man gjør et stort feilgrep, kan man raskt bygge en ny VM eller bare starte en ny container. Ihvertfall så lenge man har backup av alle data som man hadde på serveren. Og det er viktig: ta alltid backup av script og lignende filer som dere har på Linux-VM.

6.4 Brukere

En bruker på et UNIX/Linux system består av følgende:

⁷⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux6del14.mp4>

- En linje i filen `/etc/passwd` som inneholder bl.a
 - Bruker id - UID. Må være unik.
 - Brukernavn
 - Hjemmekatalog
- En linje i filen `/etc/shadow` som inneholder en hash av passordet
- En hjemmekatalog (vanligvis i `/home`)
- Eventuelle gruppemedlemskap i filen
- `/etc/group`

Man kan lage nye bruker med kommandoen `adduser` eller `useradd`. Sistnevnte er kun anbefalt dersom man skal legge til en bruker via et script eller lignende. Kommandoen `adduser` fungerer mer interaktivt og passer best for å legge til en og en bruker. Den kan brukes slik

```
group60@os60:~$ sudo adduser s123456
```

for å legge til brukeren `s123456`.

For å legge denne brukeren til sudo-gruppen slik at den blir en sudo-bruker, kan gjøres med

```
group60@os60:~$ sudo addgroup s123456 sudo
```

Et UNIX/Linux system har vanligvis en del brukere i tillegg til de "menneskelige" brukerne. Den mest kjente av dem er `root`, som har UID 0. De øvrige brukerne representerer tjenester som ikke bør kjøre som brukeren `root` av risikohensyn. Eksempler på slike brukere er `nobody`, `sshd` og `sys`. Dersom man installerer flere tjenester på et system, f.eks en webserver, vil man sansynligvis få opprettet de tilsvarende brukerne automatisk.

6.5 Grupper

- gruppe av brukere
- Definert i `/etc/group`
- må defineres av `root`

```
$ groups haugerud                # lister gruppene haugerud er med i
$ chgrp studgruppe fil.txt        # fil.txt tilhører nå studgruppe
$ chmod 770 fil.txt               # alle rettigheter til alle i studgruppe
$ sudo adduser s123456 studgruppe # Gjør s123456 til medlem av studgruppe
```

Man kan lage en ny gruppe med `addgroup`:

```
mroot@os50:~$ sudo addgroup newgroup
[sudo] password for mroot:
Adding group 'newgroup' (GID 1003) ...
Done.
mroot@os50:~$ grep newgroup /etc/group
newgroup:x:1003:
```

6.6 Rettighetsprinsipper i Linux/UNIX miljøer

Følgende regler gjelder mellom brukere:

- Alle prosesser/programmer som startes av en bruker vil være eid av den brukeren og filrettigheter vil være i forhold til eierskapet. Unntaket er kommandoen **su** som forandrer bruker og tjenester som forandrer status på seg selv, f.eks **sshd** (startes som root, men blir til **sshd** kort tid etter).
- Brukere kan kun sende signaler til sine egne prosesser (f.eks med kommandoen **kill**). Unntak er root.
- Brukere kan kun forandre på rettighetene til filer som de eier selv. Unntak er root.
- Brukere kan ikke "gi" filer til andre brukere ved å forandre hvem som er eier av filen med kommandoen **chown**. Root kan.
- Brukere KAN forandre hvilken gruppe som eier en fil med kommandoen **chgrp**, men KUN til grupper som den selv er medlem av. Root kan uansett.
- Brukere kan ikke lage nye brukere. Bare root kan det.

Det finnes to måter å bli root på. Den ene er å bruke kommandoen **su**, som ber deg om root sitt EGET passord. Denne metoden er mest kjent. I de nyeste versjonene av Linux er det blitt mer vanlig å bruke **sudo**, som gir rettigheter til vanlige brukere dersom de kan autentisere seg. Man må vanligvis være medlem av en spesiell gruppe for å få lov til å kjøre kommandoen **sudo su**, som ber brukeren om sitt eget passord istedenfor root sitt passord.

Det er også mulig å logge seg rett på systemet som root dersom man har root passordet. Det er derimot ikke anbefalt av følgende årsaker:

- Alt den brukeren gjør vil bli gjort med root-rettigheter.
- Ved å bruke **su** eller **sudo su** kan vi se i logfilene HVEM som ble root. Denne typen informasjon kan være veldig viktig når man vil finne ut hva som har skjedd (og hvem som skal få skylden).
- Root skal kun brukes når man har behov for det, og ikke behandles som en normal bruker/person.

6.7 ssh-copy-id

Det kan være nyttig å kunne logge inn med **ssh** på andre servere uten å måtte skrive passord hver gang. Sikkerhetsmessig kan dette også være en fordel, da man kan velge å kun tillate innlogging ved hjelp av **ssh**-nøkler og dermed unngå alle brute-force **ssh**-angrep. Først må man lage **ssh**-keys:

```
group1@os1:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/group1/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/group1/.ssh/id_rsa.
Your public key has been saved in /home/group1/.ssh/id_rsa.pub.
group1@os1:~$
```

Deretter kan man med `ssh-copy-id` (som kopierer over `id_rsa.pub`) sørge for at man senere kan logge inn uten passord:

```
group1@os1:~$ ssh-copy-id group49@os49.vlab.cs.oslomet.no
group49@os49.vlab.cs.oslomet.no's password:

group1@os1:~$ ssh group49@os49.vlab.cs.oslomet.no
Linux os49 2.6.32-5-xen-amd64 #1 SMP Fri Feb 5 17:48:36 UTC 2016 x86_64
group49@os49:~$
```

Man kan deretter også utføre kommandoer direkte på andre servere:

```
group1@os1:~$ ssh group49@os49.vlab.cs.oslomet.no whoami;hostname
group49
os1
group1@os1:~$ ssh group49@os49.vlab.cs.oslomet.no "whoami;hostname"
group49
os49
```

6.8 Root aksess til server med sudo-rettigheter

Hvis man har en bruker på en server som har sudo-rettigheter, har man vanligvis ikke mulighet til å logge seg inn med `ssh` direkte til root-brukeren på serveren. Det er mulig å sette det opp, men det er sikrere og vanlig å skru av muligheten for å logge inn som root med passord. Man kan derfor heller ikke overføre `ssh`-nøkler med `ssh-copy`, men det er likevel mulig å sette opp `ssh-key` innlogging for root. Det som behøves er at nøkkelen også legges i root sin `.ssh`-mappe. Det er viktig å forsikre seg om at root faktisk har en mappe `/root/.ssh` og en enkel måte å sørge for at den blir opprettet (om den ikke har blitt opprettet tidligere) med riktige rettigheter er å kjøre `ssh-keygen` som root:

```
# ssh-keygen
```

og taste return på alle spørsmål.

Hvis du allerede har satt opp innlogging til din vanlige bruker på serveren, i dette eksempelet til `group60@os60`, kan du nå overføre nøkkelen derfra slik som dette:

```
group60@os60:~$ sudo cp .ssh/authorized_keys /root/.ssh
```

Hvis mappen `/root/.ssh` ikke eksisterer fra før blir den laget som en fil og da vil innlogging ikke virke. Dermed skal du kunne logge inn også som root fra den serveren, f.eks. `studssh`, som du satte opp `ssh-key` innlogging fra til `group60` i utgangspunktet. Når man overfører sin offentlige nøkkel til en annen server på denne måten, er det en enveis-innlogging. Det vil ikke være mulig å gå tilbake til serveren man kom fra med denne nøkkelen. Men man må passe godt på sin private nøkkel (`id_rsa`) for om andre får tilgang til den, kan de logge seg inn til de serverene som du har overført den offentlige nøkkelen til (`id_rsa.pub`). Når du logger deg på en server med `ssh-key`, er det din private nøkkel (`id_rsa`) som brukes til å autentisere deg (bevise at det er deg).

6.9 Bakgrunnsjobber og screen

Hvis man ønsker å sette igang en jobb i et terminalvindu og la det stå og jobbe selvom man selv logger ut, kan man bruke `screen`. Da kan man senere logge seg inn på nytt til samme host og så koble seg opp til terminalvinduet som ligger i bakgrunnen. Man installerer `screen` og starter en ny `screen-session` med

```
$ sudo apt update -y
$ sudo apt install screen
$ screen
Screen version 4.03.01 (GNU) 28-Jun-15

$ uname
Linux
$ ./loop.sh
loop nr 1
loop nr 2
loop nr 3
loop nr 4
```

og dermed får man et terminalvindu som ser ut som et helt vanlig terminalvindu. Her kan man kjøre kommandoer og for eksempel sette igang en jobb som `loop`-jobben over. Når man så kobler seg fra dette `screen`-vinduet med `CTRL-a CTRL-d`, kommer man tilbake til shellen hvor man kjørte kommandoen `screen`.

```
$ screen
[detached from 9833.pts-0.os1]
$
```

Så kan man logge ut og inn igjen(eventuelt fra et helt annet sted) og liste alle `screen-sessions` med

```
$ screen -ls
There are screens on:
9833.pts-0.os1 (14. feb. 2017 kl. 18.15 +0100) (Detached)
2216.compile (13. feb. 2017 kl. 09.01 +0100) (Detached)
923.pts-1.os1 (13. feb. 2017 kl. 08.57 +0100) (Detached)
3 Sockets in /var/run/screen/S-group1.

$
```

Så kan man koble seg til den `screen` man ønsker seg med

```
$ screen -r pts-0.os1
```

og man vil se at jobben har fortsatt å kjøre i terminalvinduet når man var logget ut.

```
group1@os1:~$ ./loop.sh
loop nr 1
loop nr 2
loop nr 3
loop nr 4
loop nr 5
loop nr 6
loop nr 7
loop nr 8
loop nr 9
loop nr 10
```

Om man ønsker å se om man er inne i en screen og vise hvilken det er, kan man oppnå det med

```
$ echo $STY
9833.pts-0.os1
```

For å gi navn til en screen kan man starte den med

```
$ screen -S compile
```

og den listes med navn som vist over i eksempelet med **screen -ls**. Om man fra screen-vinduet taster CTRL-d (uten CTRL-a foran) vil screen-prosessen drepes. På samme måte som om man drepte den med kill. For å scrolle opp og ned må man bruke kommandoen CTRL-a ESC og så opp og ned piltaster. ESC igjen for å avslutte scrolling.

Om man leser manual-siden for screen, finner man kommandoer man trenger i litt mer spesielle tilfeller, men dette er alt man behøver for enkel bruk.

6.9.1 Å dele en screen med samme bruker

Hvis to personer er logget inn på samme Linux-server med samme bruker, kan de enkelt dele en felles screen der begge kan utføre kommandoer og dermed jobbe sammen. Først må en av dem lage en ny screen med opsjonene -d -m som gjør at man lager en screen men ikke kobler seg til den:

```
screen -d -m -S felles
```

Deretter kan begge koble seg til med

```
screen -x felles
```

og de vil begge kunne utføre kommandoer og se hva den andre gjør.

6.9.2 Bakgrunnsjobber og screen

Screen er også et fint verktøy å bruke for å sette igang batch-jobber som potensielt kan bruke lang tid på å bli ferdig og som kan finne på å skrive til både stderr og stdout. Det siste kan være problematisk om man har koblet seg opp med ssh for å starte jobbene, uten å sørge for å redigere stderr og stdout. Default skriver de da tilbake til terminalen ssh-tilkoblingen kom fra og da kan hele jobben crashe om disse tilkoblingene ikke lenger finnes fordi du har logget ut. Forøvrig er følgende en sikker metode å sette igang en bakgrunnsjobb på en annen maskin med ssh:

```
ssh haugerud@studssh.cs.oslomet.no '/home/haugerud/back.sh </dev/null >/home/haugerud/backup.log 2>&1 &'
```

Legg merke til at både stdin, stdout og stderr er tatt hånd om, slik at alle koblinger til ssh-kanalen er brutt. I tillegg er jobben (back.sh) avsluttet med &, slik at den legges i bakgrunnen. I tillegg brukes full path for å sikre at det ikke er noen avhengighet av PATH. Dermed vil denne fortsette å kjøre i bakgrunnen etter at man logger seg ut fra shellet som startet jobben med ssh. Alternativt kunne man startet jobben fra en screen-session, som ville stått og tatt i mot eventuell output. Men da kunne man potensielt fått problemer om serveren som kjører screen hadde gått ned.

6.10 scp

scp (secure copy) kopierer filer mellom maskiner og sender alt kryptert:

```
mroot@os50:~$ scp haugerud@rex.cs.oslomet.no:~/www/regn .
haugerud@rex.cs.oslomet.no's password:
regn                                100% 194   123.7KB/s   00:00
mroot@os50:~$
```

Meget nyttig og mye brukt for sikker filoverføring. Hvis man er på studssh og vil overføre en fil r.sh derfra, kan man gjøre det med

```
haugerud@studssh:~$ scp r.sh mroot@os50.vlab.cs.oslomet.no:
mroot@os50.vlab.cs.oslomet.no's password:
r.sh                                100% 39    0.0KB/s    00:00
```

Med opsijonen -r kan man overføre mapper(inkludert undermapper):

```
haugerud@studssh:~$ ls -l nymappe/
total 16
-rwxr-xr-x 1 haugerud drift 34 jan.   8 10:17 mitt.sh
-rwxr-xr-x 1 haugerud drift 39 jan.   8 10:10 r.sh
-rw-r--r-- 1 haugerud drift 24 jan.   8 10:09 tomfil
haugerud@studssh:~$ scp -r nymappe mroot@os50.vlab.cs.oslomet.no:
mroot@os50.vlab.cs.oslomet.no's password:
tomfil                                100% 24    0.0KB/s    00:00
r.sh                                  100% 39    0.0KB/s    00:00
mitt.sh                              100% 34    0.0KB/s    00:00
```

6.11 Backup med rsync og cron-tab

Når man har satt opp passord-løs innlogging med ssh-keys som vist over, er det enkelt å sette opp systematisk backup. Anta man ønsker å ta backup av `/home/group1` på en os-VM. Det kan man nå gjøre med

```
scp -r /home/group1/ haugerud@studssh.cs.oslomet.no:/home/haugerud/kopiavos1
```

og hele mappen og alle undermapper blir kopiert over. Men når man gjør det på nytt en gang til, vil alle filer kopieres over enda en gang. Linux-kommandoen `rsync` gjør det samme, men den kopierer bare over filer som har endret seg fra gang til gang: (Gjør først `$ sudo apt install rsync`)

```
rsync -a /home/group1/ haugerud@studssh.cs.oslomet.no:/home/haugerud/rsynckopiavos1
```

Når man lager en ny fil eller gjør en endring, er det bare dette som kopieres over neste gang. Den enkleste måten å få dette til å bli en daglig rutine (eventuelt hver time) er å bruke `cron`. På Linux-VM må cron først installeres med

```
group100@os100:~$ sudo apt install cron
```

Om man kjører

```
crontab -e
```

får man opp en fil som man kan bruke til å kjøre jobber til visse tider på døgnet. Følgende linje i `crontab`

```
# Edit this file to introduce tasks to be run by cron.
#
# m h dom mon dow    command
30 1 * * * /home/group1/bin/rsyncStudssh.sh
```

fører til at scriptet kjører hver natt kl 01.30. Hvis man bytter ut tallet 1 med `*`, vil scriptet `rsyncStudssh.sh` kjøres 30 minutter etter hver hele time. Forkortelsene i linjen over forklarer syntaksen i `crontab`:

m minute

h hour

dom day of month

mon month

dow day of week

På siden crontab.guru⁷⁷ er det enkelt å teste ut hva forskjellige crontab-koder gir.

Om man har root aksess kan man også legge script i mappene definert i `/etc/crontab`:

```
group1@os1:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab`
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
```

Script som for eksempel legges i `/etc/cron.hourly` vil kjøres 17 minutter over hver hele time.

6.12 wget

`wget` henter web (og ftp) sider fra kommandolinjen.

```
mroot@os50:~$ wget https://www.cs.oslomet.no/~haugerud/regn
--2018-02-25 22:29:42--  https://www.cs.oslomet.no/~haugerud/regn
```

På denne måten kan store nedlastninger kjøres i bakgrunnen. Opsjonen `-O` - sender output til STDOUT, slik at det kan brukes i en pipe.

```
wget -r https://www.gnu.org -o log.txt
```

Opsjonen `-r` henter hele web-stedet rekursivt (default dybde er 5). Med opsjonen `--mirror` og kjørt regelmessig, kan man opprettholde en mirror-site.

6.13 gzip, bzip2, tar og zip

`gzip` komprimerer en fil mens `gunzip` dekomprimerer den.

```
$ ll
-rw-r--r--  1 haugerud drift      130680 Oct  5 20:11 passwd
$ gzip passwd
$ ll
```

⁷⁷<https://crontab.guru>

```
-rw-r--r--    1 haugerud drift      39093 Oct  5 20:11 passwd.gz
$ gunzip passwd.gz
$ ll
-rw-r--r--    1 haugerud drift      130680 Oct  5 20:11 passwd
```

De nyere `bzip2` og `bunzip2` gjør det samme, er ikke like raskt, men komprimerer bedre.

```
$ bzip2 passwd
$ ll
-rw-r--r--    1 haugerud drift      29246 Oct  5 20:11 passwd.bz2
$ bunzip2 passwd.bz2
```

Kommandoen `tar` pakker en hel mappestruktur til en enkelt fil.

```
$ tar cf dir.tar dir # Pakker alt ned i dir.tar, c = create
$ tar xf dir.tar     # Pakker alt opp, lager dir, x = extract
```

Man kan kjøre `gzip` på en tar fil, men det er enklere å gjøre alt på en gang:

```
$ tar cfz dir.tgz dir # Pakker alt ned i dir.tar, c = create, z = gzip
$ tar xfz dir.tgz     # Pakker alt opp, lager dir, x = extract, z = gzip
$ tar cfz obl.tgz snort.bash info.bash # Pakker de to filene i obl.tgz
$ tar cfj dir.tbz dir # j = bzip2
```

Man kan gjøre det samme med kommandoen `zip` som kan kjøre på mange plattformer og er kompatibel med windows sin PKZIP.

```
$ zip -r dir.zip dir # -r = rekursivt i mappen dir
$ unzip dir.zip
```

`tar cfz` komprimerer noe mer:

```
-rw-----    1 haugerud drift      728780 Sep 27 00:58 forelesning.tgz
-rw-----    1 haugerud drift      1086314 Sep 27 01:00 forelesning.zip
```

6.14 awk (ward)

Syntaks: `awk 'awk program' fil`

```
$ ps u
haugerud 23419  0.0  0.1  2032 1268 pts/1    S    09:07   0:00 -bash

$ ps aux | awk '/haugerud/ {print $2}'
23419
4396
4397
```

```
$ ps aux | awk '/haugerud/ {print $2,$11}'
23419 -bash
4403 ps
4404 awk
$ ps aux | awk '{if ($1 == user) {print $2}}' user='whoami'
23419
4416
4417
$ awk -F: '{ if ($1 == "haugerud") {print $NF}}' /etc/passwd
/bin/bash
```

Spørsmål

Hvordan fjerne et alias? Svar: med unalias: unalias move

7 Forelesning 5/3-24(2 timer). Containere og Docker

En fin måte å komme igjennom alle assignmentene i ukeoppgavene er å gå igjennom Mike Longs slides og hver gang det kommer en slide med "Your turn", klikke deg inn på neste assignment og gjøre den. Slidene ligger under filer i Canvas og heter Docker Kickstart.pdf. . Pensum er til og med slide 44, Attaching and executing.

7.1 Forelesningsvideoer

Her er en link til et uredigert opptak av Containere og Docker-forelesningen⁷⁸ fra våren 2020 hvor Docker Kickstart-slidene blir gjennomgått. Docker containere kan kjøres på samme måte på årets VMer. Om du har problemer med å få sett videoen på Firefox, prøv Chrome.

Opptak av forelesningen inndelt etter temaer:

linux7del1.mp4⁷⁹ (01:54) Om slidene, Docker Kickstart og continuous delivery

linux7del2.mp4⁸⁰ (04:04) Slides: Hva er Docker og hvorfor bruke containere?

linux7del3.mp4⁸¹ (01:07) Slides: Avhengigheter, isolasjon, portabilitet

linux7del4.mp4⁸² (04:45) Slide: ressurs-utnyttelse: virtuelle maskiner sammenlignet med containere

linux7del5.mp4⁸³ (02:41) Slide: Utviklingen av applikasjonsmiljøet fra 2000 til idag

linux7del6.mp4⁸⁴ (03:23) Slides: En docker container er ... Mer sammenligning av VMer og containere, terminologi

linux7del7.mp4⁸⁵ (05:25) Slides og demo: Kjøre Hello world med docker på Linux-VM

linux7del8.mp4⁸⁶ (02:00) Slides: Problemet med "Go to first exercise" forklart, i url må 1 endres

⁷⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux7.mp4>

⁷⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del1.mp4>

⁸⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del2.mp4>

⁸¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del3.mp4>

⁸²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del4.mp4>

⁸³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del5.mp4>

⁸⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del6.mp4>

⁸⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del7.mp4>

⁸⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del8.mp4>

til 01

linux7del9.mp4⁸⁷ (05:05) Demo: Starte opp en ubuntu-container interaktivt; liste containere

linux7del10.mp4⁸⁸ (03:40) Slides: Laste ned alpine-image med pull og kjøre det

linux7del11.mp4⁸⁹ (02:49) Slides og demo: Docker-hub, Image-tags

linux7del12.mp4⁹⁰ (04:28) Spørsmål: Hvorfor er docker nyttig og hva brukes det til? Kan det sammenlignes med JVM?

linux7del13.mp4⁹¹ (03:06) Demo: Flere eksempler med docker container run alpine og listing av containere

linux7del14.mp4⁹² (02:08) Slides: Sletting av images og containere

linux7del15.mp4⁹³ (05:30) Demo: Sletting av images og containere

linux7del16.mp4⁹⁴ (02:37) Slides: Port forwarding

linux7del17.mp4⁹⁵ (02:31) Demo: Port forwarding med port 7979 til 80

linux7del18.mp4⁹⁶ (08:23) Demo: Attach og execute, CTRL-P CTRL-Q

linux7del19.mp4⁹⁷ (15:28) Demo: docker container commit for å lage en kopi av den kjørende containeren. Apache2 på både VM og container

8 Forelesning 12/3-24(2 timer). Docker og Dockerfile

Forelesning slides⁹⁸.

8.1 Forelesningsvideoer

Uredigert opptak av hele forelesningen (01:31:19)⁹⁹

Opptak av forelesningen inndelt etter temaer:

linux8del2.mp4¹⁰⁰ (08:58) Slides: Hvorfor docker?

linux8del3.mp4¹⁰¹ (08:13) Demo: info om containere, sletting av containere og image, prune

linux8del4.mp4¹⁰² (01:36) Spørsmål: Hva er forskjellen på image og container

linux8del5.mp4¹⁰³ (06:06) Demo av forskjellen på image og container, stopp, start og tilkobling

linux8del6.mp4¹⁰⁴ (06:41) Docker volumes, slides

linux8del7.mp4¹⁰⁵ (19:35) Demo: docker files, docker build av apache webserver

⁸⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del9.mp4>

⁸⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del10.mp4>

⁸⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del11.mp4>

⁹⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del12.mp4>

⁹¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del13.mp4>

⁹²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del14.mp4>

⁹³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del15.mp4>

⁹⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del16.mp4>

⁹⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del17.mp4>

⁹⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del18.mp4>

⁹⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/2021/linux7del19.mp4>

⁹⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/docker.pdf>

⁹⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8.mp4>

¹⁰⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del2.mp4>

¹⁰¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del3.mp4>

¹⁰²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del4.mp4>

¹⁰³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del5.mp4>

¹⁰⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del6.mp4>

¹⁰⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del7.mp4>

linux8del8.mp4¹⁰⁶ (06:38) Demo: hvordan endre dockerfile slik at apache startes når containeren starter
 linux8del9.mp4¹⁰⁷ (02:12) Demo: Hvordan bygge ett image og starte mange webservere på forskjellige portnummere
 linux8del10.mp4¹⁰⁸ (01:37) Demo: Hvordan endre på innholdet for en kjørende docker webserver
 linux8del11.mp4¹⁰⁹ (02:04) Spørsmål: Har man flere image i en container?
 linux8del12.mp4¹¹⁰ (04:06) Demo: Hvordan få en lokal fil på serveren til å vises av en docker webserver (Siden ble ikke vist pga manglende rettigheter på serveren. `chmod 755 /root` gjorde at den ble løst, docker-kommandoen for å starte containeren var riktig)

8.2 Endring siden innspillingen ble laget

Apache2-installasjonen spør nå interaktivt om tidssone og byggingen vil henge. Legg derfor inn riktig tidssone som andre og tredje linje i Dockerfile:

```
ENV TZ=Europe/Oslo
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
```

9 Forelesning 12/3-24(2 timer). Docker og Dockerhub, shell script ytelse

Uredigert opptak av hele forelesningen¹¹¹ Mangler litt av starten av 1. og 2. time, se slides og notater.

Opptak av forelesningen inndelt etter temaer:

linux9del1.mp4¹¹² 9.4 Uke-oppgave 12.10, oppsett av programmeringsmiljø med Dockerfile
 linux9del2.mp4¹¹³ 9.4 Sammenligning av programmeringsspråk, hvilket er raskest?
 linux9del3.mp4¹¹⁴ 9.4 Resultater fra sammenligningen
 linux9del4.mp4¹¹⁵ 9.4 Fortsettelse av oppgave 12.10, docker-container ferdig bygget
 linux9del5.mp4¹¹⁶ 9.4 Script for å teste effektivitet i Dockerfile (problem ved 7-8 min: glemmer å bygge på nytt)
 linux9del6.mp4¹¹⁷ 9.4 Effektivitet ved å lese og skrive til fil
 linux9del7.mp4¹¹⁸ 9.5 Regulære uttrykk i bash (referansestoff, ikke pensum)

¹⁰⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del8.mp4>

¹⁰⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del9.mp4>

¹⁰⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del10.mp4>

¹⁰⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del11.mp4>

¹¹⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux8del12.mp4>

¹¹¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9.mp4>

¹¹²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del1.mp4>

¹¹³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del2.mp4>

¹¹⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del3.mp4>

¹¹⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del4.mp4>

¹¹⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del5.mp4>

¹¹⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del6.mp4>

¹¹⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del7.mp4>

linux9del8.mp4¹¹⁹ 9.5 Problem med mellomrom i regulært uttrykk, løsning: definer regexp som en variabel (referansestoff, ikke pensum)

9.1 Sist

- Hvorfor docker
- Dockerfile
- docker build

9.2 Dockerhub

Vanligvis laster man ned docker image fra Dockerhub¹²⁰ hvor det ligger tusensvis av ferdilagde image som man kan laste ned og bruke, som ubuntu og nginx. Men man kan også lage sitt eget repository og laste opp egne ferdiglagde image dit som andre så kan laste ned og bruke. Anta man vil lage en ubuntu container som har jed installert og bruker følgende Dockerfile:

```
root@os110:~/osbuntu# cat Dockerfile
from ubuntu
RUN apt-get -y update
RUN apt install -y jed
```

Deretter bygger man den og lager et image som man kaller for eksempel osbuntu:

```
root@os110:~/osbuntu# docker build -t osbuntu .
```

I dette tilfellet har jeg en bruker på dockerhub som heter haugerud og jeg må derfor tagge imaget på riktig måte for å laste det opp dit:

```
root@os110:~/osbuntu# docker tag osbuntu haugerud/osbuntu:latest
```

Dette kan også gjøres direkte når man bygger. Default versjon er 'latest', men man kan også lage flere versjoner med egendefinerte versjonsnummer som 1.0, 2.0, etc.

For å kunne pushe et image til Docker Hub må man først logge inn:

```
root@os110:~# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://
Username: haugerud
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

Login Succeeded

¹¹⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux9del8.mp4>

¹²⁰<https://hub.docker.com>

og så kan man pushe et image:

```
root@os110:~# docker push haugerud/osubuntu:latest
The push refers to repository [docker.io/haugerud/osubuntu]
58657d799cb5: Pushed
d0729e7aec69: Pushed
8aaae71fccde: Pushed
03cb13a80250: Pushed
b93c1bd012ab: Mounted from library/ubuntu
1.0: digest: sha256:5d9e384a97484e2af982d940833102b14638b057a286b0ca950906903e4b0220 size: 1368
root@os110:~#
```

Deretter kan hvem som helst laste ned dette imaget fra hvor som helst og sikre seg en fin Ubuntu-container med jed ferdig installert:

```
root@os800:~# docker pull haugerud/osbuntu
Using default tag: latest
latest: Pulling from haugerud/osbuntu
Digest: sha256:55bd19eebe21a365b789dbf7930df2027e3c860f87b31d161f8b143be361bbab
Status: Image is up to date for haugerud/osbuntu:latest
docker.io/haugerud/osbuntu:latest

root@os800:~# docker run -it haugerud/osbuntu /bin/bash
root@601858e8cd75:/# type jed
jed is /usr/bin/jed
```

Hvis man ønsker å laste ned en annen versjon enn latest, for eksempel versjon 1.0, kan man gjøre det på følgende måte:

```
root@os800:~# docker pull haugerud/osbuntu:1.0
```

9.3 Shell-programmering, oppsummering

- + Fint til enkle oppgaver der Linux-kommandoer gjør jobben
- + Slipper å kompilere
- + Pipes, omdirigering: Kraftige verktøy
- + bra til enkle systemscript
 - dårlig programstruktur (variabel-typer, parameter-overføring, klasser, etc.)
 - dårlige feilmeldinger/debugging vanskelig
 - kryptisk syntaks
 - Veldig langsom sammenlignet med kompilert kode

9.4 Hastighet til programmer skrevet i bash, python, perl, Java og C

Vi skal nå sammenligne hastigheten til programmer skrevet i bash, python, perl, Java og C som utfører følgende kode:

```
for(j=0;j < TIMES;j++)
{
    sum = 0;
    for(i=0;i < 40000;i++)
    {
        tall = (i + 1)*(i + 1) - i*i;
        sum = sum + tall;
    }
}
```

I 2010 ble hastigheten til disse programmen testet på to av serverene ved oslomet og resultatene så da slik ut på cube (Linux) og nexus (Solaris):

Språk	CPU-tid i sekunder på cube	CPU-tid i sekunder på nexus	TIMES
sh	248	-	1
bash	5.7	11.8	1
php	5.6	13.3	21
perl	5.6	16.4	75
Java	5.6	-	13500
C/C++	5.6	4.3	63000

Det viste altså at C-programmet var 63000 ganger raskere enn et bash-script.

På forelesningen i 2019 ble dette resultatene med 500.000 runder i innerste løkke (12 ganger mer enn i 2010):

```
sum.bash:  TIMES = 1;
sum.perl:  TIMES = 36;
sum.py:    TIMES = 44
sum.php:   TIMES = 400;
sum.c:     TIMES = 4100;
Sum.java:  TIMES = 20000;
sum0.c:    TIMES = 28000;
```

på en Linux PC med en Intel Core i7-3770 CPU med klokkefrekvens på 3.40GHz. Det er overraskende at forskjellene er så store. Og at Java er så nær C i effektivitet. Det siste skyldes at java bruker en såkalt JIT(Just-In-Time) kompilator, som kan compilere hele eller deler av bytekoden om til maskinkode, tilsvarende som når C kompileres, rett før programmet kjøres. Forøvrig bruker java for å gjøre dette delvis to CPUer. Om java-programmet tvinges til å kjøre på en CPU, reduseres TIMES til 17000.

Forøvrig er resultatet for sum.c om man kompilerer med gcc uten å bruke opsijonen -O. Da kompiles programmet hurtig, men ikke med fokus på at det skal kjøre raskt. Programmet sumO.c er kompilert med -O og da går C-programmet mye raskere.

Å finne ut om forskjellen på Linux-VM'ene er den samme er en av ukens øvingsoppgaver.

Men det er viktig å teste på nøyaktig det man er ute etter. Følgende kode leser en fil og skriver den til en annen fil med linjenummer:

```
#!/bin/bash

echo "" > /tmp/ny.fil
nr=0
while read line
do
    (( nr++ ))
    echo "$nr: $line" >> /tmp/ny.fil
done < /tmp/stor
echo "Read $nr lines"
```

Testes dette på å lese en 13 Mbyte tekstfil på studssh gir det:

Språk	CPU-tid i sekunder
bash	68.57
Java	17.04
php	9.3
perl	2.18
C++	2.00

Og som man ser er forskjellene mye mindre.

Følgende avsnitt om regulære uttrykk betegnes i dette kurset som "referansestoff". Dette er stoff som det er interessant og viktig å vite om og som kan være nyttig å bruke i praksis, men dette stoffet vil det ikke bli spurt om til eksamen og er slik sett ikke en del av pensum.

9.5 Regulære uttrykk (referansestoff, ikke pensum)

Med grep og egrep kan man matche enkle strenger, men det er mange begrensninger og det ville vært bedre om man kunne gjøre det i bash som en del av språket. De fleste programmeringspråk, som Java, JavaScript og Python, bruker regulære uttrykk som er kompatible med PCRE (Perl Compatible Regular Expressions) opprinnelig utviklet i Perl. Siden bash versjon 3 finnes det en innebygget operator for regulære uttrykk, =~ som støtter det meste av uttrykkene som er mulig med PCRE, men ikke alle. Det som støttes er POSIX Extended Regular Expressions (ERE).

Siden det meste av konfigurasjon og filer i Linux består av vanlig tekst, er regulære uttrykk et meget nyttig verktøy. Tidligere var Perl en del av pensum i kurset Operativsystemer, det er et mer moderne og komplett programmeringspråk enn bash og er samtidig systemnært. Men med mulighet for å bruke både assosiative array og innebygde regulære uttrykk i bash, kan man løse de fleste oppgaver som er relevante for Linux med bash. Og eventuelt bruke objektorienterte språk som Python, Java eller C++ om det er større og mer komplekse oppgaver som skal løses. Som vi senere skal se er det ytelsen til bash som i tillegg til kryptisk syntaks er den største begrensningen.

Regulære uttrykk er et meget kraftig verktøy for å gjenkjenne tekstmønstre i strenger. Det kan ta litt tid å mestre det, men man når langt med litt kunnskap. Følgende er et enkelt eksempel på hvordan man kan bruke regulære uttrykk i bash:

```
#!/bin/bash

regex="ho"

echo "Matcher $regex"

while read line
do
    if [[ $line =~ $regex ]]
    then
        echo "match!"
    fi
done
```

Vanligvis er kilden naturligvis en av Linux sine mange tekstfiler eller tekst-strømmer fra pipes. Den vesentlige syntaksen er `$line =~ $regex` hvor nøyaktig det som er i `$regex` blir matchet, hverken mer eller mindre.

```
$ linje="hei og ho"
$ if [[ $linje =~ (.*?)ho ]]; then echo "M:${BASH_REMATCH[1]}"; fi
M:hei og
```

Settes parenteser `()` rundt deler av en regexp blir disse delene fortløpende lagt i spesielle nummererte variabler `${BASH_REMATCH[1]}`, `${BASH_REMATCH[2]}` etc. Dermed kan man plukke ut nøyaktig det man ønsker:

```
$ line="en linje"
$ if [[ $line =~ (^.*)n(.*)n ]]; then echo match av "${BASH_REMATCH[1]}" og av "${BASH_REMATCH[2]}" ; fi
match av "e" og av " li"
```

Der den første parentesen matcher starten av linjen og så null eller flere tegn, og det er som vi ser av output bokstaven e som matcher. Så kommer bokstaven n og så kommer den andre parentesen med matching av " li" før en ny n. Dermed kan vi plukke ut nøyaktig de delene av setningen vi ønsker.

Men om det er et mellomrom i det regulære uttrykket, gir bash en feilmelding, slik det skjedde på forelesningen:

```
$ line="en linje"
$ if [[ $line =~ (^.*) (.*)n ]]; then echo match av "${BASH_REMATCH[1]}" og av "${BASH_REMATCH[2]}" ; fi
bash: syntaksfeil i betinget uttrykk: forventet ikke symbolet (
bash: syntaksfeil ved (.
```

Her er det mellomrommet i det regulære uttrykket som er problemet. Om man erstatter det med `[:space:]` vil det fungere, men det matcher mer enn bare et mellomrom. Det er mulig å escape mellomrommet `"\ "`, noe som fjerner feilen. En generelt bedre måte å gjøre dette på er å definere det regulære uttrykket i en egen variabel og så gjennomføre matchingen, da unngår man denne og andre potensielle syntaksfeil:

```
$ line="en linje"
$ reg='(^.*) (.*n)'
$ if [[ $line =~ $reg ]]; then echo match av "${BASH_REMATCH[1]}" og av "${BASH_REMATCH[2]}" ; fi
match av "en" og av "li"
```

Dermed kan man for eksempel plukket ut tall fra en linje:

```
$ line="1 og 2 og 3"
rex:~$ reg='([0-9]) og ([0-9])'
rex:~$ if [[ $line =~ $reg ]]; then echo match av "${BASH_REMATCH[1]}" og av "${BASH_REMATCH[2]}" ; fi
match av "1" og av "2"
```

9.6 Regulære uttrykk med grep(referansestoff, ikke pensum)

Man kan match regulære uttrykk med grep om man bruker opsijonen P. Men grep matcher på en litt annen måte enn det som er vanlig, den fortsetter å matche videre i linjen om samme mønster oppstår igjen:

```
$ echo "1 og 2 og 3 og 4" | grep -P "\d og"
1 og 2 og 3 og 4
```

matcher egentlig `"\d og"` tre ganger. Vanligvis vil et slikt regulært uttrykk kun matche `"1 og"`. Man kan se det ved å legge til opsijonen -o, da skriver grep ut hver match på en egen linje:

```
$ echo "1 og 2 og 3 og 4" | grep -P -o "\d og"
1 og
2 og
3 og
```

Legg merke til at grep støtter alle PCRE, inkludert `"\d"` og tilsvarende uttrykk som `=~` matchingen ikke støtter og man må bruke `"[0-9]"` istedet.

9.7 Spesielle tegn i Regulære uttrykk(referansestoff, ikke pensum)

Tegn	betydning
<code>^</code>	matcher begynnelsen av strengen
<code>\$</code>	matcher slutten av strengen
<code> </code>	skiller alternativer
<code>.</code>	matcher alt untatt <code>\n</code>
<code>\</code>	neste tegn skal ikke tolkes (<code>\\$ matcher \$</code>)
<code>()</code>	grupperer

Regexp test	matcher
\$line =~ ^ho	ho på starten av strengen
\$line =~ ho\$ hi\$	ho eller hi på slutten av strengen
\$line =~ .\s\$	Vilkårlig tegn fulgt av \$

9.8 Kvantifisering i Regulære uttrykk II(referansestoff, ikke pensum)

- $a \rightarrow$ en a
- $a^+ \rightarrow$ en eller flere a'er

Kvantifisering	tegnet før må forekomme
{n,m}	minst n og maksimalt m ganger
{n,}	minst n ganger
{n}	eksakt n ganger
*	0 eller flere ganger (= {0,})
+	1 eller flere ganger (= {1,})
?	0 eller 1 gang (= {0,1})

Regexp	matcher
[A-Z]^ho	en eller fler store bokstaver fulgt av ho
ho{3}	hooo
(ho){3}	hohoho

9.9 Spesielle koder i Regulære uttrykk(referansestoff, ikke pensum)

Kode	Matcher
[af8]	et av de tre tegnene a, f eller 8
[0-9]	siffer
[a-z]	små bokstaver
[a-zA-Z_0-9]	ord-tegn
[[:space:]]	[\t\n\r], mellomrom (eller tab, newline,return)
[[:alnum:]]	[a-zA-Z0-9]

Regexp	matcher
ho[09]-	ho fulgt av siffer
[A-Z][[:space:]]ho	En stor bokstav fulgt av mellomrom og ho
[a-z][a-zA-Z_0-9]^	liten bokstav fulgt av ett eller flere ord tegn

Kvantifiseringen er grådig og matcher så mye som mulig.

```
if [[ $linje =~ (.*)(.*) ]]; then echo "M:${BASH_REMATCH[1]}"; fi
M:en % to % tre
```

9.10 Bytte ut med regexp(referansestoff, ikke pensum)

```
$ linje="very old, very old"
```

```
$ echo ${linje/old/new}
very new, very old
$ echo ${linje//old/new}
very new, very new
$ echo ${linje//[a-z]/new}
newnewnewnew newnewnew, newnewnewnew newnewnew
```

10 Forelesning 2/4-24(2 timer). Docker Compose

Ingen videoer på forhånd om dette temaet, det vil bli gått igjennom på forelesningen på tirsdag. Og det vil være noen tilhørende oppgaver som vil være de siste obligatoriske oppgavene som skal med i oblig3 og dermed de siste obligatoriske oppgavene i kurset.

10.1 Docker Compose og docker-compose.yaml

Vi har tidligere sett at det er en ryddigere og mer systematisk måte å bygge containere på ved å bruke en Dockerfile. Da kan man definere alt man ønsker skal være med når man starter containeren, som vilken programvare som skal være installert, hvilke filer som skal kopieres inn og så videre. Dette er et bedre alternativ enn å starte en container, installere det som trengs av programvare og innhold og så lagre denne containeren som et image og senere bruke denne. Da er det vanskeligere å gjøre endringer, vanskeligere å huske hva containeren egentlig inneholder og generelt vanskeligere å gjenskape det samme imaget med noen endringer til å bruke i andre sammenhenger.

Docker compose med den tilhørende docker-compose.yaml filen er en metode som gjør noe av den samme forenklingen for å kjøre containere som Dockerfile gjør for å bygge containere. Ofte trenger man å legge til mange flagg og opsjoner når man starter en container og dette kan gi lange og uoversiktlige docker container run-kommandoer. Man kan gjøre dette på en mye mer ryddig og systematisk måte ved å definere alt som skal skje når man starter en container i en docker-compose.yaml fil. Man kan i en slik fil også velge å starte flere samtidige containere som skal samarbeide om å gi den tjenesten man ønsker. For eksempel kan man med Docker compose samtidig starte både en webserver og en database-server som webserveren henter dataene sine fra. Generelt kan man bruke dette til å sette opp mange forskjellig typer oppsett av samtidige containere på en ryddig og overiktlig måte. Dermed er det også enkelt å endre på konfigurasjonen og stoppe og starte hele clusteret av containere for å få alt til å virke som man ønsker.

YAML sto opprinnelig for "Yet Another Markup Language" og er som XML et maskinlesbart format som også er inspirert av Python i den forstand at riktig innrykk i teksten er viktig og det fører til feilmeldinger om dette ikke er riktig definert. Derfor må man være svært nøye med innrykk/antall mellomrom og også med å ha mellomrom på riktige steder. Dette gir noe av de samme syntaks-problemene som ved bash-scripting, derfor er det også her en god strategi å sakte bygge opp en docker-compose.yaml fil og teste hver gang man gjør endringer.

10.2 Docker Compose hello-world

Det gir kanskje ikke så mye mening å bruke Docker compose for å kjøre en hello-world container, men det kan være nyttig å teste ut for å gjøre seg kjent med konseptene. En docker-compose.yaml

som starter en hello-world container kan se slik ut:

```
version: '3.0' # Yaml-versjon
services:
  hello:
    image: hello-world:latest
```

Første linje forteller hvilken Yaml-versjon som skal brukes og deretter listes alle services som skal være med. I dette er det kun en som vi gir navnet 'hello'. Deretter følger hvilket image som skal brukes. Dermed er man klar til å kjøre hello-world containeren med:

```
# docker-compose up
Starting hello_hello_1 ... done
Attaching to hello_hello_1
hello_1 |
hello_1 | Hello from Docker!
```

Hvis man her ikke markerer at 'hello' er en av tjenestene og skriver filen slik:

```
version: '3.0' # Yaml-versjon
services:
  hello:
    image: hello-world:latest
```

får man med en gang en feilmelding:

```
# docker-compose up
ERROR: In file './docker-compose.yaml', service 'image' must be a mapping not a string.
```

10.3 Docker Compose nginx

Hvis man ønsker å starte en nginx-container kan man bruke følgende yaml-fil:

```
version: '3.1'
services:
  nginx:
    image: nginx:latest
```

og starte tjenesten med

```
# docker-compose up -d
Creating network "test_default" with the default driver
Creating test_nginx_1 ... done
# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
df21da9d3c67	nginx:latest	"/docker-entrypoint."	5 seconds ago	Up 4 seconds	80/tcp	test_nginx

Hvis dette er første gang man laster ned nginx-imaget, vil det først lastes ned på samme måte som når man kjører `docker run nginx`. Deretter kan man stoppe det hele med:

```
# docker-compose down
Stopping test_nginx_1 ... done
Removing test_nginx_1 ... done
Removing network test_default
```

Docker-compose rydder opp etter seg ved å fjerne containeren som ble kjørt.

Hvis man ønsker at port 80 som nginx default bruker som source port skal vises som port 8080 på host'en som kjører containeren, kan man gjøre det ved å definere følgende i yml-filen:

```
version: '3.1'
services:
  nginx:
    image: nginx:latest
    ports:
      - 8080:80
```

Generelt kan alle parametre og opsjoner man kan gi til `docker container run` defineres i yml-filen. Deretter kan man starte nginx:

```
# docker-compose up -d
Starting test_nginx_1 ... done
root@os800:~/test# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
9b37c7a488e8	nginx:latest	"/docker-entrypoint."	59 seconds ago	Up 10 seconds	0.0.0.0:8080->80/tcp	test_nginx_1

og man vil kunne få en hilsen fra nginx:

```
# curl localhost:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Videre kan man på en enkel måte definere en mappe på host'en som nginx skal hente sine web-sider fra ved å legge til følgende rett under ports: i yml-filen:

```
volumes:
  - ./innhold:/usr/share/nginx/html:ro
```

Dette gjør at man nå vil få opp `innhold/index.html` filen på host'en når man starter nginx.

10.4 Tjenester med flere samtidige containere

Det er først og fremst med tanke på å sette opp flere samtidige containere som jobber sammen om å tilby en gitt tjeneste at docker compose virkelig er et kraftig verktøy. Hvis man definerer flere services innen samme docker.compose.yaml fil, vil docker compose også sette opp et lokalt privat nettverk som containerene kan kommunisere på. Dermed kan man for eksempel sette opp et lite nettverk med en database og en web-server som kommuniserer med hverandre.

Følgende yaml-fil definerer to containere som leverer forskjellig innhold på henholdsvis port 8080 og 8081:

```
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - 8080:80
    volumes:
      - ./innhold:/usr/share/nginx/html:ro
  nginx2:
    image: nginx:latest
    ports:
      - 8081:80
    volumes:
      - ./innhold2:/usr/share/nginx/html:ro
```

Dermed kan man starte begge containerene samtidig og se at de virker som de skal. Og stoppe begge etterpå.

```
# docker-compose up -d
Starting test_nginx_1 ... done
Starting test_nginx2_1 ... done
# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
f507d9c369c5   nginx:latest   "/docker-entrypoint."    About a minute ago   Up 4 seconds   0.0.0.0:8081->80/tcp               test_nginx_1
fe8b6ca0922d   nginx:latest   "/docker-entrypoint."    About a minute ago   Up 4 seconds   0.0.0.0:8080->80/tcp               test_nginx2_1
fb87e1b854ba   ubuntu         "tail -f /dev/null"      4 weeks ago        Up 4 weeks
# curl localhost:8080
Container innhold!
# curl localhost:8081
Container innhold 2!
# docker-compose down
Stopping test_nginx2_1 ... done
Stopping test_nginx_1 ... done
Removing test_nginx2_1 ... done
Removing test_nginx_1 ... done
```

Om man går inn i den ene nginx og installerer ping, vil man kunne se at man kan kommunisere over det lokale private nettverket med den andre containeren ved å bruke navnet som er definert i yaml-filen:

```
# docker exec -it f2 bash
```

```
root@f2d45f6e9bf5:/# ping nginx2
PING nginx2 (192.168.32.3) 56(84) bytes of data.
64 bytes from test_nginx2_1.test_default (192.168.32.3): icmp_seq=1 ttl=64 time=0.137 ms
```

De to containerene har IPer 192.168.32.3 og 192.168.32.2 og kan kommunisere med hverandre som på andre nettverk. Dette gjør det mulig å sette opp realistiske systemer, ikke minst for å teste kode som man utvikler for dette service-scenariet.

10.5 docker-compose build

Man kan kombinere docker-compose med en eller flere Dockerfiles ved å spesifisere en mappe der den tilhørende Dockerfile ligger ved å angi 'build' istedet for image i yaml-filen:

```
version: '3'
services:
  nginx:
    build: ./nginx
    ports:
      - 8080:80
```

Dermed vil docker-compose prøve å bygge et image fra Dockerfile i mappen ./nginx og starte en container med det image't som er resultatet av denne byggingen. Og man kan sette opp en oversiktlig mappestruktur som definerer alle containerene som er med i et compose-prosjekt.

11 Forelesning 9/4-24(2 timer). Virtuelle maskiner

Slides brukt i forelesningen¹²¹

11.1 Forelesningsvideoer

Uredigert opptak av hele forelesningen (01:16:17)¹²²

Opptak av forelesningen inndelt etter temaer:

linux10del1.mp4¹²³ (04:26) Grunnleggende om hva virtualisering er. Hardware, hypervisor og gjeste-OS

linux10del2.mp4¹²⁴ (17:30) Hvorfor virtualisering? Isolasjon, Ressurssparing, Fleksibilitet, Programvare-utvikling, Skytjenester

linux10del3.mp4¹²⁵ (07:39) Historie, Sensitive og privilegerte instruksjoner

linux10del4.mp4¹²⁶ (03:09) Hardwarestøttet virtualisering, hypervisor i kernel mode, gjeste-OS i

¹²¹<https://www.cs.oslomet.no/haugerud/vm.pdf>

¹²²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10.mp4>

¹²³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del1.mp4>

¹²⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del2.mp4>

¹²⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del3.mp4>

¹²⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del4.mp4>

user mode

linux10del5.mp4¹²⁷ (02:56) Poll med gjennomgang av svar på hva som IKKE er en av fordelene ved virtualisering: effektivitet

linux10del6.mp4¹²⁸ (04:20) Oppsummering av hardwarestøttet virtualisering

linux10del7.mp4¹²⁹ (08:01) Type 1 & Type 2 hypervisor, binær oversettelse, paravirtualisering

linux10del8.mp4¹³⁰ (04:47) Ytelsestap ved virtualisering

linux10del9.mp4¹³¹ (04:31) Live migration og nested virtualisering

linux10del10.mp4¹³² (03:53) Qemu og KVM, kjøre qemu i Linux kommandolinje

linux10del11.mp4¹³³ (05:25) Effektivitet for 4 C-programmer når de kjører bare metal og virtuelt

linux10del12.mp4¹³⁴ (01:29) Unikernel (En slide med IncludeOS falt ut av opptaket)

11.2 Virtualisering

- Virtualisering av server/desktop hardware
- En hypervisor simulerer hardware ved å gi samme grensesnitt som virkelig hardware gir
- Operativsystemene som kjører på en virtuell maskin, tror de kjører på ekte hardware

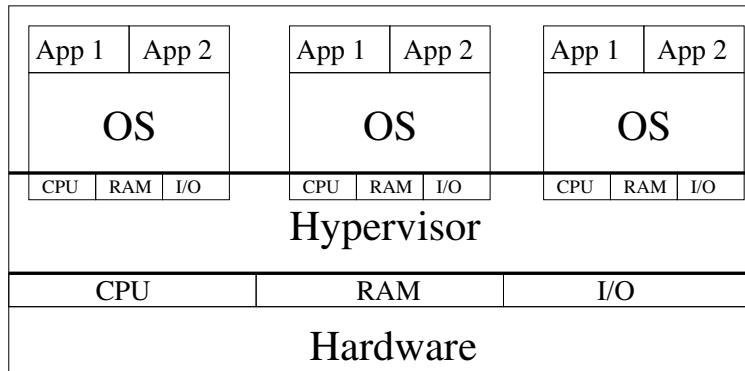


Figure 14: .

11.3 Hvorfor virtualisering?

- Isolasjon
- Ressurssparing
- Fleksibilitet
- Programvare-utvikling

¹²⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del5.mp4>

¹²⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del6.mp4>

¹²⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del7.mp4>

¹³⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del8.mp4>

¹³¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del9.mp4>

¹³²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del10.mp4>

¹³³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del11.mp4>

¹³⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux10del12.mp4>

- Skytjenester

Alle disse fordelene gjelder også for containere og Docker, bortsett fra den første, isolasjon og sikkerhet. Men fleksibiliteten blir enda større med containere.

11.4 Isolasjon

- Tjenester og programmer kan kjøre på hver sin dedikerte server
- Unngår at de forskjellige tjenestene ødelegger for hverandre og gir ryddigere drift
- Men hva om den fysiske serveren eller hypervisor feiler?
- Det meste av nedetid og feil skyldes ikke hardware men software. Og software for en hypervisor er generelt mindre kompleks enn all programvaren på en hel maskin
- Sikkerhet: hvis en tjeneste blir hacket, vil det ikke påvirke de andre tjenestene
- Dette er fordi Operativsystemet og applikasjonene kun kommuniserer mot det virtuelle hardware-API'et som hypervisor gir dem tilgang til. De har ingen mulighet til å kommunisere med andre deler av en hypervisor eller andre VMer.

11.5 Ressurssparing

- Man kan oppnå isolasjon ved å ha en fysisk server for hver tjeneste, men det gir store driftskostnader
- Med virtualisering kan det samme oppnås på en enkelt server
- Virtuelle maskiner (VMer) som for eksempel bruker lite CPU kan settes på samme fysiske server
- VMer kan enkelt flyttes til og fra fysiske servere og man kan dermed spare hardware og strøm
- Hage, Thomas: *The CERES project - A Cloud Energy Reduction System*, Veileder: Kyrre Begnum (Masteroppgave i Nettverk og Systemadministrasjon, oslomet/UiO)
- <https://www.cs.oslomet.no/teaching/materials/MS100A/html/NSA.html>

11.6 Fleksibilitet

- Kapasiteten kan enkelt økes ved å legge til flere VMer, lastbalansering blir enklere
- Elastisitet: Man kan dynamisk tildele CPUer og internminne til VMer
- Har en VM blitt ødelagt eller kompromittert kan man enkelt starte opp en ny kopi
- Tradisjonelt er det arbeidskrevende å flytte en tjeneste eller et softwareprosjekt til en ny server på grunn av avhengighet av operativsystemet og annen programvare: når noe er utviklet på en VM så kan hele VMen flyttes eller kopieres
- Live migration: Hele VM flyttes til annen fysisk server uten nedetid på tjenestene

- Ung, Fredrik: *Towards efficient and cost-effective live migrations of virtual machines*, NSA masteroppgave
- Ahmad, Bilal: *Coordinating vertical and horizontal scaling for achieving differentiated QoS*, NSA masteroppgave, Veiledere: Anis Yazidi og Hårek Haugerud

11.7 Skalering av ressurser

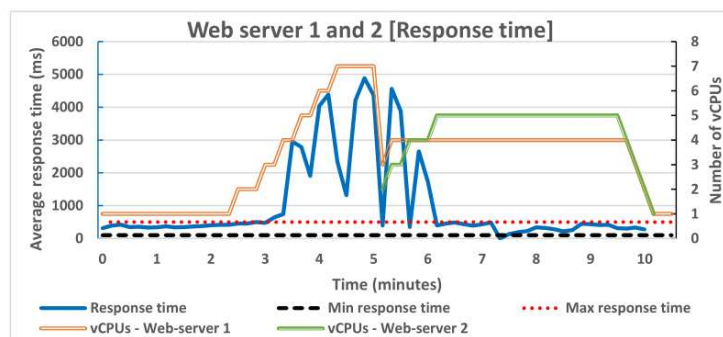


Figure 6.5: Web server 1 and 2 - response time in relation to vCPUs

Figure 15: .

11.8 Programvare-utvikling

- Man kan raskt teste ut programvare på forskjellige operativsystemer, Window, Linux, Mac, etc. ved å kjøre VMer med en rekke forskjellige OS
- Det er enklere å automatisere tester på flere plattformer (Test Driven Development)
- Ønsker man å teste ut nye ideer, kan man raskt sette opp miljøer for å teste dem ut

11.9 Skytjenester

- Virtualisering er grunnlaget for fleksible skytjenester
- Kunder kan gis egne VMer med et antall CPUer, disk og minne
- Disse kundene kan dele fysiske servere, noe som gir store besparelser av hardware
- Nettbokhandelen Amazon startet med skytjenester fordi de bare hadde bruk for store mengder hardware til webserverene sine før jul og tenkte at de kunne leie ut hardware-ressursene resten av året

11.10 Historie

- IBM startet med virtualisering av stormaskiner på 1960-tallet
- En VMM (Virtual Machine Monitor) styrte flere virtuelle maskiner på samme fysiske maskin
- Første virtualiseringsløsning for x86: VMware i 1999
- Deretter fulgte Xen, VirtualBox, KVM og mange andre
- Hardware-støtte for x86 virtualisering kom først i 2005

11.11 Krav til virtualisering

- Popek og Goldberg, 1974: En maskin kan bare virtualiseres hvis alle **sensitive instruksjonene** også er **priviligerte instruksjoner**
- En **Sensitiv instruksjon** kan bare utføres i kernel mode
- En **Priviligert instruksjon** forårsaker en trap til kernel mode hvis den gjøres i user mode

Eksempel:

- X86 instruksjonen POPF (skruv av og på interrupts) er en sensitiv instruksjon
- Hvis den utføres i user mode vil ingenting skje, som for NOP (No Operation)
- Instruksjonen CLI (Clear Interrupt flag) er en sensitiv instruksjon, men den er også privilegert. Hvis den utføres i user mode, gjøres en trap til kernel mode
- Vanlige instruksjoner som ADD, CMP og MOV er hverken sensitive eller privilegerte.

11.12 Hardware støttet virtualisering

- Hardware-støtte for x86 virtualisering kom først i 2005
- Inntil da fantes det sensitive instruksjoner (som POPF) som bare ble droppet
- Gjeste-OS kjører i user mode og vil ikke fungere som på vanlig hardware om en slik instruksjon droppes. Det vil føre til uforutsigbar oppførsel og kan føre til at OS crasher i verste fall.
- Med Intel VT-x og AMD-V vil alle sensitive instruksjoner trap'e til kernel mode når de utføres i user mode

11.13 Type 1 hypervisor

- En type 1 hypervisor kjører direkte på hardware som et OS
- Alle sensitive instruksjoner som utføres i user mode av gjeste-OS må trap'e til kernel mode og fanges opp av hypervisor
- Eksempler: VMware ESX og vSphere, Xen, Hyper-V(Microsoft)

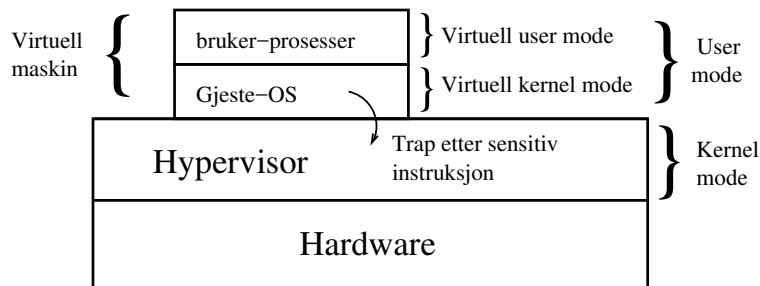


Figure 16: Hardware støttet virtualisering.

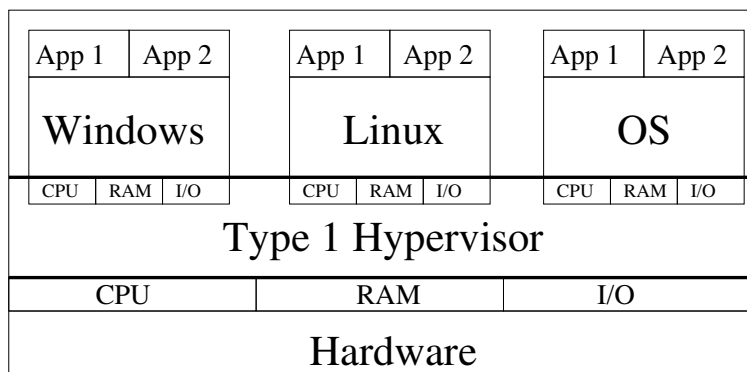


Figure 17: Type 1 Hypervisor.

11.14 Type 2 hypervisor

- En type 2 hypervisor kjører oppå et eksisterende OS
- Deler av hypervisor kan inngå i det underliggende OS'et i form av kjernemoduler
- Det kan være litt flytende grenser mellom type 1 og type 2
- KVM/Qemu(Linux), VirtualBox, VMware Fusion (Mac)

11.15 Binær oversettelse

- Før 2005 måtte alternative metoder brukes, uten hardware-støtte
- VMware lagde en hypervisor som mens et program kjører scanner koden etter sensitive instruksjoner
- Dette gjøres for hver kodeblokk som ender i jump, call, trap eller lignende
- Sensitive instruksjoner oversettes til kall til VMware-prosedyrer i hypervisor
- De oversatte kodeblokkene caches og dette gjør kjøringen effektiv
- Hardware med VT-støtte genererer mange traps og dette tar lang tid
- Noe binær oversettelse finnes også i VirtualBox

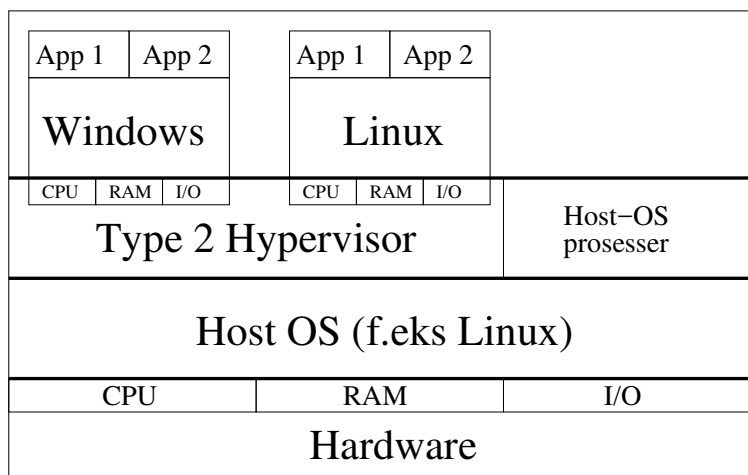


Figure 18: Type 2 hypervisor.

11.16 Paravirtualisering

- Paravirtualisering krever at Gjest-OS endres
- Alle sensitive instruksjoner erstattes med kall til hypervisor
- Gjest-OS kan optimaliseres for virtualisering
- Ved å installere drivere laget for paravirtualisering, kan denne metoden bli meget effektiv

11.17 Ytelse

- Ytelesen reduseres generelt noe ved virtualisering
- Binær-oversettelse kan i noen tilfeller gi bedre ytelse enn bare metal (kjøre rett på hardware)
- Yaqub, Naveed: Comparison of Virtualization Performance: VMWare and KVM, NSA masteroppgave (2012)
- Mest nedlastede masteroppgave fra UiO (mer enn 15.000 nedlastninger)
- Sammenlignet: KVM vs VMWare ESXi 5.0 vs Bare Metal
- Red Hat Enterprise Linux 6.1, kernel 2.6.32, 8 CPUer, 72 GB RAM

11.18 Disk ytelse

11.19 RAM ytelse

Cache: L1: 256 KB, L2: 1 MB, L3: 12 MB

11.20 CPU ytelse

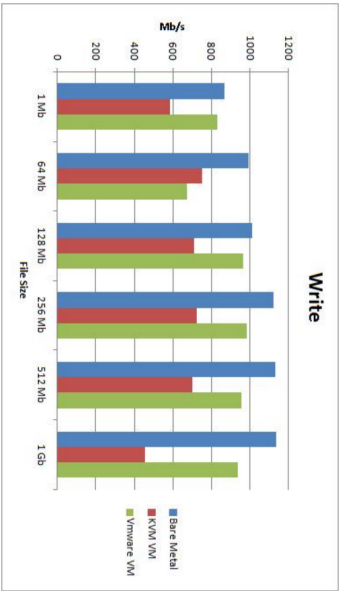


Figure 5.1: Iozone average write

Figure 19: .

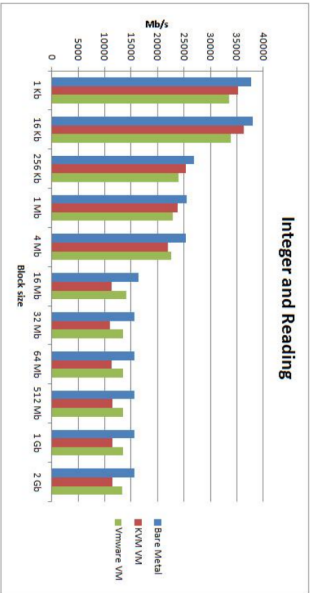


Figure 5.17: Ram speed average integer and reading

Figure 20: .

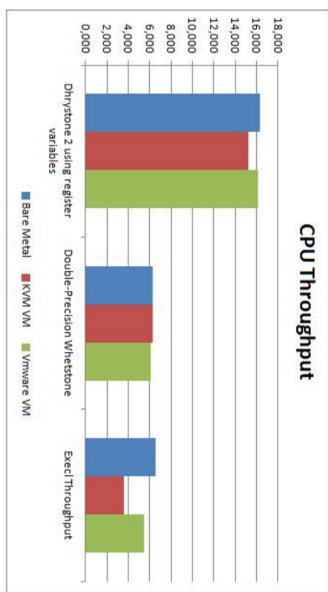


Figure 5.30: UnixBench CPU Throughput

Figure 21: .

11.21 Live migration

- Ved live migration flyttes en VM fra en fysisk server til en annen samtidig som den kjører kontinuerlig
- De statiske delene av minnet kopieres over mens VM'en kjører og tas imot på den andre serveren
- Når nesten alt er kopiert, stoppes VM'en, siste rest kopieres over og den startes opp igjen på den andre serveren
- Disken med VM-imaget og VM sitt filsystem er vanligvis et nettverksfilsystem, for eksempel NFS
- Wu, Hai: *Simulating Live Migration using Micro Virtual Machines* NSA masteroppgave (2015)

11.22 Simulering av migration

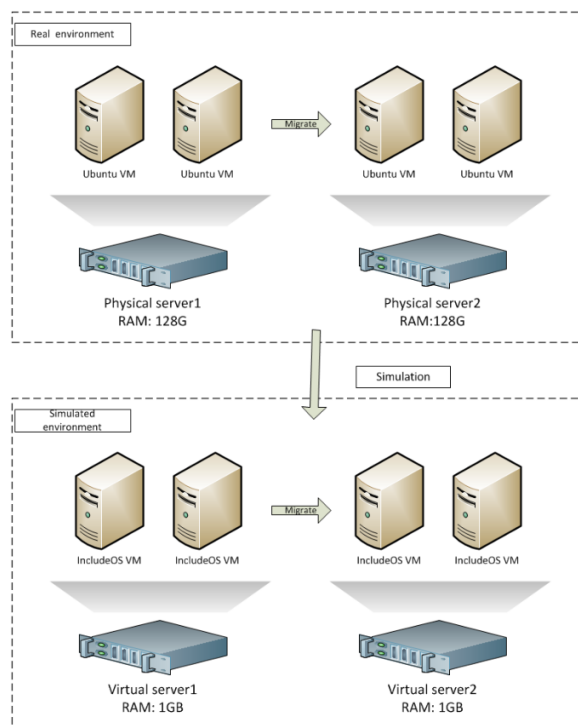


Figure 3.2: Modeling the data center

Figure 22: .

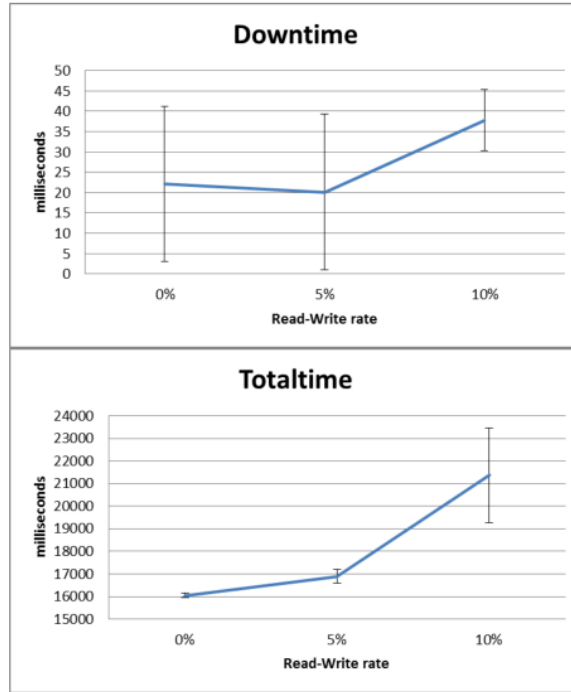


Figure 23: .

11.23 Migration av Ubuntu

11.24 Migration av IncludeOS

11.25 Nested virtualisering

- Ved nested virtualisering kjører man en Hypervisor inne i en annen VM, slik at det blir tre lag
- Dette gir ytterligere ytelsesreduksjon
- Intel har hatt hardwarestøtte for nested virtualisering siden 2013
- Kan brukes til å flytte VMer mellom ulike Cloud-providere

11.26 Qemu og KVM

- QEMU (Quick EMUlator) er en open source programvare som kan emulere en komplett datamaskin
- QEMU kan emulere mange instruksjonssett, inkludert x86, MIPS, 32-bit ARMv7, ARMv8, PowerPC
- QEMU kan boote en rekke OS: Linux, Windows, Solaris, DOS, BSD

- KVM (Kernel Virtual Machine) er en Linux kjernemodul som gjør at user mode programmer kan unytte hardware virtualisering for en rekke forskjellige prosessorer, inkludert Intel og AMDs x86-prosessorer
- For å utnytte KVM må man kjøre QEMU-prosessen med opsjonen `-enable-kvm`

11.27 Test av QEMU og KVM ytelse

Følgende viser hvordan man på en Linux-server kan starte en virtuell maskin med QEMU:

```
root@intel:~/vm# qemu-system-x86_64 -enable-kvm -name server2 -m 1024 -realtime mlock=off -smp 1,sockets=1,cores=1,th
```

Opsjonene til kommandoen angir hvilket image som skal startes (`-hda`), hvor mye RAM VMen skal ha (1024 MByte) og hvor mange CPU'er (1). Opsjonen `-enable-kvm` angir at qemu skal bruke KVM, kjernemoduler som unytter hardware-virtualisering og gjør at ytelsen blir mye bedre. Med `lsmod` kan man se at de nødvendige kjerne-modulene er lastet inn:

```
root@intel:~/vm# lsmod | grep kvm
kvm_intel          172032  6
kvm                540672  1 kvm_intel
```

De følgende avsnittene viser fire C-program med vekslende bruk av systemkall og dermed instruksjoner som fører til at koden må trap'e til hypervisor.

11.27.1 sum.c

Dette programmet kjører stort sett bare i user mode og bør ha omtrent samme ytelse native (uten virtualisering også kaldt 'bare metal') som under virtualisering.

```
#include <stdio.h>
#define uint64_t unsigned long int

int main(void) {
    uint64_t i, s = 0;
    for (i=0; i<5000000000; i++) {
        s = s + i;
    }
    return(0);
}
```

11.27.2 getppid.c

Dette programmet består av å gjøre systemkallet `getppid` om og om igjen.

```
#include<unistd.h>
```

```

int main(void) {
    int i;
    for (i=0; i<20000000; i++) {
        getppid();
    }
    return(0);
}

```

11.27.3 gettimeofday.c

Her kalles gettimeofday om og om igjen, og det er et litt mer komplekst systemkall som fører til mange andre, som bruker noe RAM og som potensielt kan gi forskjeller mellom bare metal og virtuelt.

```

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/time.h>

int main(void)
{
    struct timeval c;
    int i;
    for(i=0; i<40000000; i++) {
        gettimeofday(&c,NULL);
    }
    return(0);
}

```

11.27.4 forkwait.c

Det siste eksempelet gir også mange systemkall og en god del minne-bruk.

```

#include <unistd.h>
#include <sys/wait.h>

#include<unistd.h>

int main(void) {
    int i,pid;
    for (i=0; i<10000; i++) {
        pid = fork();
        if (pid < 0) return -1;
        if (pid == 0) return 0;
        waitpid(pid,NULL,0);
    }
    return(0);
}

```

11.27.5 Bare metal

Programmene kompiles med

```
gcc -O prog.c -o prog
```

før de kjøres. Hvert program kjøres med

```
/usr/bin/time -f %e prog
```

og dette gir total tid programmet bruker i sekunder.

```
root@intel:~/virt# ./runiter.bash
sum getppid gettimeofday forkwait
1.47;0.89;0.70;2.49;
1.46;0.89;0.71;2.54;
1.46;0.85;0.70;2.48;
1.47;0.89;0.71;2.49;
1.46;0.85;0.71;2.45;
```

11.27.6 qemu med -enable-kvm

Testprogrammene kjøres så på en virtuell maskin, kjørt med qemu-kommandoen vist over:

```
root@server2:~/virt# ./runiter.bash
sum getppid gettimeofday forkwait
1.46;0.75;1.69;0.44;
1.39;0.75;1.69;0.43;
1.39;0.75;1.69;0.44;
1.47;0.75;1.68;0.51;
1.39;0.75;1.69;0.46;
```

De to første programmene går som forventet omtrent like raskt, men noe overraskende går det litt raskere på den virtuelle maskinen. Kall til gettimeofday går vesentlig saktere på VM'en og det er ikke unaturlig da trap til hypervisor tar mye tid. Men det er meget overraskende at fork-kall går raskere på den virtuelle maskinen. QEMU gjør binær-oversettelse av koden og dette kan potensielt gi raskere kode enn nå man kjører den samme koden direkte på en fysisk maskin.

11.27.7 qemu uten -enable-kvm

Testprogrammene kjøres så på samme virtuelle maskin, kjørt med qemu-kommandoen vist over, men hvor opsjonen `-enable-kvm` er fjernet:

```
root@server2:~/virt# ./runiter.bash
sum getppid gettimeofday forkwait
10.91;4.38;3.67;10.47;
```

```
11.06;4.28;3.42;10.51;  
10.51;4.38;3.35;10.73;  
10.80;4.29;3.50;10.24;  
10.04;4.34;3.54;10.58;
```

Vi ser at det går vesentlig saktere med en ren qemu-emulering uten bruk av hardware-aksellerasjon. Programmet `sum` som kjører i user mode bruker omtrent åtte ganger så langt tid og `forkwait` bruker 20 ganger så lang tid.

11.28 Unikernels og IncludeOS

En Unikernel er en minimalistisk operativsystemkjerne som også inneholder selve tjenesten eller applikasjonen som skal kjøres. Typisk inneholder en unikernel kun akkurat det som er nødvendig for at tjenesten som skal tilbys kan kjøre. Den er altså spesialtilpasset til akkurat denne oppgaven og dermed unngår man å ha med kode i kjernen som er overflødig i akkurat dette tilfellet. Generelle operativsystem er laget for å kunne kjøre en hvilken som helst applikasjon.

IncludeOS er et operativsystem som ble laget helt fra bunnen av og skrevet i assembly og C++. Det er laget med tanke på at det skal kjøres som en virtuell maskin og bare det helt nødvendige for at en tjeneste skal kunne kjøres er tatt med i operativsystemet. Utgangspunktet er en tjeneste skrevet i C++ og man inkluderer i programmet bokstavelig talt operativsystemet med

```
#include <os>
```

i starten av programmet. IncludeOS er single threaded, all kode kjører i kernel mode og det er ikke mulig å logge inn som en bruker.

Alfred Bratterud er skaperen av IncludeOS og han har hele tiden ledet utviklingen siden starten ved oslomet i 2013.

For mer info om IncludeOS, se

- <https://github.com/includeos/IncludeOS>¹³⁵
- <https://www.includeos.org/>¹³⁶
- <https://www.digi.no/artikler/na-er-det-norske-operativsystemet-includeos-klar-til-bruk/358515>¹³⁷
- <https://www.basefarm.com/no/nyheter-og-media/basefarm-includeos-partner>¹³⁸
- <https://www.kode24.no/koden-bak/includeos-gjengen-tror-ikke-linux-er-egna-til-alt-det-brukes-til-i-dag/70345115>¹³⁹
- https://www.youtube.com/watch?v=t4etEwG2_LY¹⁴⁰

¹³⁵<https://github.com/includeos/IncludeOS>

¹³⁶<https://www.includeos.org/>

¹³⁷<https://www.digi.no/artikler/na-er-det-norske-operativsystemet-includeos-klar-til-bruk/358515>

¹³⁸<https://www.basefarm.com/no/nyheter-og-media/basefarm-includeos-partner>

¹³⁹<https://www.kode24.no/koden-bak/includeos-gjengen-tror-ikke-linux-er-egna-til-alt-det-brukes-til-i-dag/70345115>

¹⁴⁰https://www.youtube.com/watch?v=t4etEwG2_LY

- <https://www.youtube.com/channel/UCB39zTsMj3dPz88ngUMz54Q>¹⁴¹
- Bratterud, A., & Haugerud, H. (2013). *Maximizing hypervisor scalability using minimal virtual machines*. In Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on (Vol. 1, pp. 218-223). IEEE.
- Bratterud, A., Walla, A. A., Haugerud, H., Engelstad, P. E., & Begnum, K. (2015). *IncludeOS: A minimal, resource efficient unikernel for cloud services*. In Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on (pp. 250-257). IEEE.

Følgende liste viser avlagte masteroppgaver i Nettverk og Systemadministrasjon ved OsloMet/UiO relatert til IncludeOS (idag gis denne mastergraden ved OsloMet og heter ACIT, Cloud-based Services and Operations):

- Kot, Martin: Evaluation of Bare Metal CPU and Memory Performance of the Unikernel IncludeOS and Ubuntu
- Muggerud, Erlend Andresen: Optimizing the efficiency of anIncludeOS unikernel DDOS filter
- Oddsund, Thomas Stenberg: On flower and bees; A framework for unikernel-based MapReduce
- Tambs, Tobias: Unikernel Firewall Performance Evaluation: IncludeOS vs. Linux
- Storvik, Jehans Jr: Developing, and running an API Gateway in the cloud using a unikernel operating system
- Goudarzi, Farhoud: A self aware approach for optimizing round trip time in web services
- Akubue, Arinze George: Lean MapReduce: A B-tree Inspired MapReduce Framework
- Czipri, Tamas : A performance comparison of KVM, Docker and the IncludeOS Unikernel
- Storvik, Jehans Jr : Developing, and running an API Gateway in the cloud using a unikernel operating system
- Aasen, Tim: Read only file system for the IncludeOS Unikernel
- Morshedi Chinibolagh, Maghsoud : Development of Multicore Computing for a Cloud-Based Unikernel Operating System
- Wu, Hai : Simulating Live Migration using Micro Virtual Machines

Hårek Haugerud og Kyrre Begnum har vært veiledere for de fleste av oppgavene.

12 Forelesning 16/4-24(2 timer). Windows og PowerShell

Enkelte avsnitt omtales som ”referansestoff”. Dette er stoff som det kan være interessant å lese for å få en bredere forståelse, men dette stoffet vil det ikke bli spurt om til eksamen.

Forelesning slides¹⁴².

¹⁴¹<https://www.youtube.com/channel/UCB39zTsMj3dPz88ngUMz54Q>

¹⁴²<https://www.cs.oslomet.no/haugerud/os/Forelesning/windows.pdf>

12.1 Forelesningsvideoer

Uredigert opptak av hele forelesningen (01:21:47)¹⁴³

Opptak av forelesningen inndelt etter temaer:

linux11del1.mp4¹⁴⁴ (02:02) Introduksjon om Windows og PowerShell
linux11del2.mp4¹⁴⁵ (02:36) Windows historikk
linux11del3.mp4¹⁴⁶ (03:24) Windows brukere, grupper og Local Security Policy
linux11del4.mp4¹⁴⁷ (02:45) Windows filsystemer og filrettigheter
linux11del5.mp4¹⁴⁸ (04:16) Windows kommandolinje, cmd.exe, batch og VBScript
linux11del6.mp4¹⁴⁹ (09:58) Windows PowerShell, introduksjon-slides
linux11del7.mp4¹⁵⁰ (07:11) Demo av Windows PowerShell i Windows 10 i Virtual Box, Set-ExecutionPolicy
linux11del8.mp4¹⁵¹ (03:48) PowerShell-demo: Get-Command og Get-Help
linux11del9.mp4¹⁵² (03:04) PowerShell-demo: likheter med Linux bash
linux11del10.mp4¹⁵³ (02:58) PowerShell-demo: variabler og environment-variabler
linux11del11.mp4¹⁵⁴ (03:21) PowerShell-demo: Apostrofer og \$(ls \$dir)
linux11del12.mp4¹⁵⁵ (05:38) PowerShell-demo: Fil-objekter (viktig!)
linux11del13.mp4¹⁵⁶ (06:18) PowerShell-demo: Prosess-objekter (viktig!)
linux11del14.mp4¹⁵⁷ (02:46) PowerShell-demo: Array av Objekter
linux11del15.mp4¹⁵⁸ (01:42) PowerShell-demo: Telle prosesser
linux11del16.mp4¹⁵⁹ (05:22) PowerShell-demo: foreach og ForEach-Object online
linux11del17.mp4¹⁶⁰ (03:00) PowerShell-demo: Select-String som erstatning for Linux-grep

12.2 Kort Windows historikk

- Microsoft ble grunnlagt av de to PC entusiastene Bill Gates og Paul Allen i 1975.
- I 1980 kom de ut med et ikke-grafisk OS kalt MS-Dos(Microsoft Disk Operating System).
- De skjønner fort at et OS må være mer brukervennlig, Grafisk Windows 1.0 gis ut i 1985.
- Helt ny Windows kjerne introduseres med Windows NT i 1993.
- PowerShell ble lansert i 2006, kommando-linje og script-språk.
- Windows 11 er siste versjon.

¹⁴³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11.mp4>

¹⁴⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del1.mp4>

¹⁴⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del2.mp4>

¹⁴⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del3.mp4>

¹⁴⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del4.mp4>

¹⁴⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del5.mp4>

¹⁴⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del6.mp4>

¹⁵⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del7.mp4>

¹⁵¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del8.mp4>

¹⁵²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del9.mp4>

¹⁵³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del10.mp4>

¹⁵⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del11.mp4>

¹⁵⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del12.mp4>

¹⁵⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del13.mp4>

¹⁵⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del14.mp4>

¹⁵⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del15.mp4>

¹⁵⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del16.mp4>

¹⁶⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux11del17.mp4>

12.3 Windows brukere

- Brukere kan defineres i Control Panel → System and Security → Administrative Tools → Computer Management → Local Users and Groups → Users. Eller enklere, trykk Windows-tasten og skriv Computer Management. Eventuelt Windows-tasten og x som gir en meny hvor du kan velge Computer Management.
- Det er som i Linux grovt sett to typer kontoer, Administrator og Standard.
- Administrator-typen(Linux root) har alle rettigheter mens standard har begrensede.
- Som i Linux, vanlig brukere kan bruke systemet men har begrenset mulighet til å endre systemet.

12.4 Windows grupper

- Grupper kan også defineres i Control Panel → System and Security → Administrative Tools → Computer Management → Local Users and Groups → Groups.
- På samme måte som i Linux finnes det noen predefinerte grupper som Administrators og users.
- Administrator brukeren kan lage nye grupper.
- For at en bruker skal ha administrator-rettigheter må han være medlem av gruppen Administrators.

Under Control Panel → User Accounts kan man endre noen egenskaper for brukerkontoer. Under Local Users and Groups får man tilgang til en mer omfattende og detaljert styring av brukere. For eksempel kan man stenge en konto for en bruker eller forlange at brukeren skal skifte passord ved neste innlogging.

12.5 Local Security Policy

Under Control Panel → Administrative Tools → Local Security Policy kan man i stor detalj definere hvilke sikkerhetsregler som skal gjelde for brukerne på maskinen og for eksempel

- sette krav til lengden på passord
- bestemme hvor lang tid det skal gå før et passord må skiftes
- kreve at man ikke bruker tidligere passord omigjen
- velge at en konto stenges automatisk etter et gitt antall mislykkede forsøk på å logge seg inn

Alt dette er metoder for å gjøre bruken av passord sikrere slik at det blir vanskeligere for hackere å komme seg inn.

12.6 Windows filsystemer og rettigheter

I et Windows-OS kan filsystemet være FAT, FAT32, NTFS eller det aller nyeste filsystemet, ReFS, Resilient File System. NTFS er fortsatt default filsystem i Windows 10.

- Hvilke fil-rettigheter i Windows(eller andre OSer) som kan settes, er bestemt av filsystemet som er i bruk.
- Med filsystemet FAT er det relativt begrenset hva som kan settes av rettigheter.
- Med FAT har man for eksempel ikke mulighet for å definere en mappe som privat slik at andre brukere ikke kan lese innholdet.
- NTFS har vært default filsystem på Windows siden Windows XP og Windows NT i 1996.
- NTFS støtter bl.a Access Control Lists(ACL) og da har man mulighet til å styre i stor detalj hvilke rettigheter brukere og grupper skal ha til enhver fil og mappe.
- Dette kan gjøres ved å høyreklikke på en fil eller katalog velge Properties og deretter velge Security fanen.
- Dette valget vil man ikke ha med et FAT-filsystem.
- Filsystemet velges under installasjonen av OSet, men man kan *i noen tilfeller* konvertere filsystemet.
- ext3 eller ext4 filsystemer som brukes av Linux, støtter ACL, men må slås på. Uten ACL, har ext et enklere og mer begrenset rettighetssystem der hver fil eller mappe kan gis rettigheter for eier, en gruppe og alle andre brukere på systemet.

12.7 Windows kommandolinje script

Fra kommandolinjen man får opp ved å kjøre cmd.exe kan man utføre en rekke kommandoer som ipconfig og tracert. På samme måte som man skriver Linux bash-script kan man skrive Windows batch script som utføres linje for linje. Det følgende eksempelet viser et script som rebooter PC'en etter 5 sekunder:

```
shutdown -r -t 05
```

Forøvrig gir `shutdown -?` info om denne kommandoen. Generelt er søk på nettet som inneholder windows commandline og det du ser etter, en bra fremgangsmåte.

12.8 Windows Script Host

Det er relativt begrenset hva man kan få til fra standard kommandolinje i Windows. Spesielt får man ikke tak i objektene som Windows-OS er bygget opp av. Rammeverket Windows Script Host (WSH) gir tilgang til noe av dette og støtter en rekke forskjellige scriptspråk inkludert, VBScript, JavaScript, Perl og Python, men det er bare VBScript og JScript (Windows egendefinerte JavaScript) som støttes i en default Windows-installasjon. Av disse er VBScript mest brukt og er

standard scriptspråk for OS-relaterte oppgaver på samme måte som bash er det på Linux. Muligens vil PowerShell etterhvert ta over. Mens `/bin/bash` er programmet som tolker Linux shellsript, er det `C:\Windows\system32\cscript.exe` og `C:\Windows\system32\wscript.exe` som gjør dette i Windows. Wscript bruker Windows-GUI og viser output fra script i popup-vinduer. Cscript gir output på kommandolinjen på samme måte som bash i Linux.

- **WSH** Windows Script Host. Støtter flere scriptspråk.
- **VBScript** Standard scriptspråk for WSH, mest brukte i OS-sammenheng
- **WScript** Default script-tolker(som bash), Wscript.exe, .vbs assosieres med denne
- **CScript** Script-tolker som kjører script fra kommandolinjen, Cscript.exe

12.9 VBScript

Det er nyttig å kjenne til VBScript siden dette er standard og mest utbredt. Det ble introdusert av Microsoft i 1996. Et VBScript har filendelse `vbs` og vil om man kjører det som

```
C:\> test.vbs
```

tolkes av Wscript. Ønsker man å få output til kommandolinjen, kan man eksplisitt kjøre det med Cscript ved kommandoen

```
C:\> cscript test.vbs
```

12.10 VBScript Hello World!

Ett objekt er allerede forhåndsdefinert når du kjører et VBScript og det er WScript-objektet. Det har mange metoder som man kan bruke, for eksempel metoden `echo` som brukes som brukes i følgende `Hello.vbs` script:

```
WScript.Echo "Hello World!"
```

12.11 VBScript-eksempler(referansestoff)

Dette avsnittet, og noen av avsnittene som kommer senere, omtales som "referansestoff". Dette er stoff som det kan være interessant å lese for å få en bredere forståelse, men dette stoffet vil det ikke bli spurt om til eksamen.

WScript-objektet har også andre metoder som `sleep` og `quit`. Som i bash deklarerer man ikke variabler, men i VBScript refereres de ikke til med `$`-tegn foran. Følgende eksempel viser de viktigste språk-konstruksjonene:

```
intTall = 4
WScript.Echo "taller er: " & intTall & "....."
```

'Kommentar: Der er ikke forskjell på for og For i et VBScript

```
For counter = 1 To 3
    WScript.Echo counter
Next

For Each element in WScript.Arguments
    WScript.Echo element
Next

i=4
while i > 0
    WScript.Echo i
    i = i - 1
wend

if i > 0 then
    WScript.Echo "ja"
else
    WScript.Echo "nei"
end if
```

12.12 Input og popup-menyer(referansestoff)

Å lese input fra en bruker er rett frem:

```
input = InputBox("Skriv noe")
wscript.echo "Du skrev " & input
```

For å bruke mer avanserte metoder enn de som WScript-objektet har, må man opprette disse objektene. For å lage popup-bokser må man for eksempel lage et WScript.Shell-objekt:

```
'Lager shell-objekt
Set objShell = WScript.CreateObject("WScript.Shell")
strText = "Vil du lage backup?"
strTitle = "Valg"
nType = vbYesNoCancel ' + vbQuestion

intResult = objShell.Popup(strText,,strTitle,nType)

Select Case intResult
    Case vbYes
        WScript.Echo "OK. Kopierer"
        objShell.Run "cmd.exe /C copy popup.vbs backup.vbs"
    Case vbNo
        WScript.Echo "Neivel"
    Case vbCancel
        WScript.Echo "Avbryter..."
```

Følgende er konstanter som kan brukes i popup-menyer:

```

'BUTTONS
vbOKOnly = 0
vbOKCancel = 1
vbAbortRetryIgnore = 2
vbYesNoCancel = 3
vbYesNo = 4
vbRetryCancel = 5

'DEFAULT BUTTONS
vbDefaultButton1 = 0
vbDefaultButton2 = 256
vbDefaultButton3 = 512
vbDefaultButton4 = 768

'ICONS
vbCritical = 16
vbQuestion = 32
vbExclamation = 48
vbInformation = 64

'RETURNS
vbCancel = 2
vbAbort = 3
vbRetry = 4
vbIgnore = 5
vbYes = 6
vbNo = 7

```

https://www.devguru.com/Technologies/vbscript/QuickRef/vbscript_intro.html er praktisk for å slå opp VBScript-syntaks.

12.13 Windows PowerShell

Windows PowerShell er, som bash for Linux, både kommandolinje og scriptspråk for Windows og ble innført av Microsoft i 2006. Fra og med Windows 2008 Server og Windows 7 har PowerShell vært installert som default. Det finnes en rekke aliaser som gjør at man kommer veldig langt med å skrive vanlige bash kommandoer.

Det finnes fire kategorier kommandoer i PowerShell:

Cmdlets Tilsvarende bash shell builtins som pwd og echo (og er en del av shellet). De fleste kommandoer er Cmdlets.

Applications Eksisterende Windowsprogrammer som ping og ipconfig. (tilsvarende /bin/mv)

Scripts Tekstfiler med endelse .ps1 (også for PS versjon 2 og høyere), tilsvarende bash-script

Functions Tilsvarende funksjoner i bash

Operativsystemet Windows er i utgangspunktet objektorientert og konfigurasjonen er ikke basert på tekstfiler som i Linux, men på binære filer og databaser. Derfor må et kraftig Windows shell også være objektorientert og det er PowerShell. Som bash er PowerShell bygd opp av mange

små programmer eller Cmdlets som gir en fleksibel måte å løse oppgaver på. Under Linux har vi sett at disse kommandoene kan settes sammen med pipes og omdirigering og det er da tekst som streames mellom kommandoene. PowerShell tar dette ett steg videre og sender hele objekter mellom Cmdlets med pipes.

12.13.1 Verdens korteste Hello World program

Hvis du har problemer med keyboard-tegn og norsk tastatur, kan du velge Settings → Time & Language → Region & Language → norsk → options og så legge til US keyboard. Så kan du switche mellom keyboard med Windows-tasten og space.

```
"Hello World!"
```

Lagrer man dette som en fil med for eksempel navnet `hello.ps1` har man laget verdens korteste Hello World program. PowerShell script må ha filendelse `ps1`. Det gjelder også for versjon 2 av PowerShell. Med tanke på script som lastes ned av virus og ormer er det i utgangspunktet ikke lov å kjøre script i det hele tatt fra PowerShell. Men hvis du setter

```
PS> set-executionPolicy remoteSigned
```

vil du kunne kjøre egne script. Men dette får du bare lov til å gjøre hvis du kjører PowerShell med "elevated privileges", det vil si som administrator. Det kan du få til ved å trykke Windows-tasten, skrive "PowerShell" og så høyreklikke og velge "Run as administrator".

Hvis PowerShell scriptet åpner et nytt vindu som umiddelbart forsvinner, slik at du ikke ser "Hello World!"-teksten, kan du finne noen mulige løsninger her: powershell-window-disappears¹⁶¹.

12.14 To viktige kommandoer

De kanskje to viktigste kommandoene i powershell er de som gir deg hjelp: "Get-Command" gir liste over alle kommandoer og "Get-Help kommando" gir informasjon om kommandoene. Disse kan sammenliknes med "man" og "help" i bash.

```
Get-Command
```

CommandType	Name	Version	Source
-----	----	-----	-----
Alias	Add-ProvisionedAppxPackage	3.0	Dism
Alias	Apply-WindowsUnattend	3.0	Dism
Alias	Disable-PhysicalDiskIndication	2.0.0.0	Storage
Function	A:		
Function	Add-BCDataCacheExtension	1.0.0.0	BranchCache
Function	Add-BitLockerKeyProtector	1.0.0.0	BitLocker
Cmdlet	Read-Host	3.1.0.0	Microsoft.Power...
Cmdlet	Receive-DtcDiagnosticTransaction	1.0.0.0	MsDtc
...			

¹⁶¹<https://stackoverflow.com/questions/1337229/powershell-window-disappears-before-i-can-read-the-error-message>

Uten argument listes alle kommandoene

```
Get-Command ls
```

CommandType	Name	Version	Source
-----	----	-----	-----
Alias	ls -> Get-ChildItem		

CommandType	Name	Definition
-----	----	-----
Alias	ls	Get-ChildItem

Med kommando som argument listes informasjon om kommandoen.

```
Get-Help Get-ChildItem
```

NAME

Get-ChildItem

SYNOPSIS

Gets the items and child items in one or more specified locations.

SYNTAX

```
Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>] [-Exclude <string[]>
```

```
...]
```

For å få informasjon om alle kommandoer, må du først kjøre Update-Help som Administrator.

12.15 Likheter med bash

Det er definert en rekke alias som gjør at mange av de kjente bash-kommandoene kan brukes direkte i powershell. Dette gjør at en del bash-script lett kan oversettes. Her er noen av de vanligste, kommandoen `alias` gir alle som er definert:

<code>set-alias cat</code>	<code>get-content</code>
<code>set-alias cd</code>	<code>set-location</code>
<code>set-alias cp</code>	<code>copy-item</code>
<code>set-alias history</code>	<code>get-history</code>
<code>set-alias kill</code>	<code>stop-process</code>
<code>set-alias ls</code>	<code>get-childitem</code>
<code>set-alias mv</code>	<code>move-item</code>
<code>set-alias ps</code>	<code>get-process</code>
<code>set-alias pwd</code>	<code>get-location</code>

```
set-alias rm          remove-item
set-alias rmdir       remove-item
set-alias echo        write-output
```

12.15.1 Omdirigering

Omdirigering og pipes virker på samme måte som i bash. For eksempel er

```
PS> ls | sort > fil.txt
```

en gyldig powershell-kommando. Vanlig output og feilmeldinger er også delt på samme måte, slik at

følgende virker som i bash:	omdirigering	virkning
	> fil.txt	omdirigerer stdout til fil.txt. Overskriver
	>> fil.txt	legger stdout etter siste linje i fil.txt
	Write-Error "oops" 2> \$null	sender stderr til "/dev/null"
	> fil.txt 2> err.txt	stdout -i fil.txt stderr -i err.txt

Men som vi skal se senere er det ikke tekst som streames mellom Cmdlets, men hele objekter!

12.16 Variabler

Variabler lages som i PHP ved å sette et \$-tegn foran navnet:

```
PS > $var = "min nye var"
PS > echo $var
min nye var
PS > $var
min nye var
```

Slike variabler er lokale. PowerShell er ikke så nøye på mellomrom som bash, men teksstrenger må skrives innenfor apostrofer. Legg merke til at echo er overflødig, en tekststreng blir skrevet ut selvom du ikke skriver echo først.

Cmdlet'en Get-Variable (kan forkortes til gv) viser hvilke variabler som er definert, følgende viser et utdrag:

```
PS > Get-Variable
```

Name	Value
----	-----
MaximumErrorCount	256
MaximumVariableCount	4096
MaximumFunctionCount	4096
MaximumAliasCount	4096
null	
false	False

true	True
PWD	C:\Documents and Settings\group10\My Documents\ps
MaximumHistoryCount	4000
HOME	C:\Documents and Settings\group10
PSVersionTable	{CLRVersion, BuildVersion, PSVersion, PSCompatible...
PID	3976
Culture	nb-NO
ShellId	Microsoft.PowerShell
PSHOME	C:\WINDOWS\system32\WindowsPowerShell\v1.0\
ErrorView	NormalView
NestedPromptLevel	0
OutputEncoding	System.Text.ASCIIEncoding
CommandLineParameters	{}
args	{}
PROFILE	C:\Documents and Settings\group10\My Documents\WindowsPowerShell\Microsoft.PowerShell_
var	min nye var
dir	mappe

12.17 Environmentvariabler

I tillegg finnes det et sett med environmentvariabler i namespace'et env: som kan listes ut med ls:

```
PS > ls env:
```

Name	Value
Path	C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32...
TEMP	C:\DOCUME~1\group10\LOCALS~1\Temp
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;...
USERDOMAIN	IU-VM
PROCESSOR_ARCHITECTURE	x86
SystemDrive	C:
APPDATA	C:\Documents and Settings\group10\Application Data
windir	C:\WINDOWS
TMP	C:\DOCUME~1\group10\LOCALS~1\Temp
USERPROFILE	C:\Documents and Settings\group10
ProgramFiles	C:\Program Files
HOMEPATH	\Documents and Settings\group10
COMPUTERNAME	IU-VM
USERNAME	group10
NUMBER_OF_PROCESSORS	1
PROCESSOR_IDENTIFIER	x86 Family 16 Model 4 Stepping 2, AuthenticAMD
SystemRoot	C:\WINDOWS
ComSpec	C:\WINDOWS\system32\cmd.exe
LOGONSERVER	\\IU-VM
ALLUSERSPROFILE	C:\Documents and Settings\All Users
OS	Windows_NT
HOMEDRIVE	C:

Dette betyr at det for eksempel finnes en variabel \$env:path

```
PS > $env:path
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\system32\WindowsP
owerShell\v1.0\;C:\Program Files\OpenSSH\bin;C:\lcc\bin
```

Legg merket til at variabler og cmdlets ikke er case-sensitive som de er i bash. Tilsvarende kan man liste funksjoner og aliaser med `ls function:` og `ls alias:`

```
PS > $alias:pwd
Microsoft.PowerShell.Management\Get-Location
PS > & $alias:pwd

Path
----
C:\Documents and Settings\group10\My Documents\ps
```

Her ser vi at en `&` i starten av en linje gjør at innholdet av en streng kjøres. Nyttig om man må ha apostrofer rundt en path fordi den inneholder mellomrom.

12.18 Apostrofer

Apostrofer virker stort sett som i bash:

```
PS> $dir="mappe"

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

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

PS> echo "File: $(ls $dir)"      # utfører kommandoen \verb+ls mappe+ inne i strengen
fil fil2.txt
```

12.19 Objekter og Get-Member

Det som virkelig gir "Power" i PowerShell er at hele objekter sendes mellom Cmdlets. Dette gjør det veldig intuitivt og enkelt å trekke ut informasjon fra for eksempel lister av prosesser og filer. Og det geniale er at det kan gjøres på mer eller mindre samme måte uansett hvilke objekter vi ser på, objektene har bare litt andre egenskaper og metoder.

For eksempel returnerer en listing av filene i en mappe

```
PS > ls
Mode                LastWriteTime         Length Name
----                -
d-----          02.03.2010    20:38         mappe
-a---          01.03.2010    12:26         73 ps.ps1
```

ikke bare tekst som viser filene i mappen slik som i bash. Egentlig returneres et array av objekter, ett for hver fil eller mappe. Om man ønsker kun et objekt for filen `ps.ps1`, kan man slik legge det i variabelen `$fil`:

```
PS > $fil = ls ps.ps1
```

Hvis man på kommandolinjen nå skriver `$fil`, kan man tabbe seg gjennom alle metoder og properties for dette objektet. En måte å vise alle på en gang, er å sende objektet til cmdlet'en `Get-Member`:

```
PS > $fil | Get-Member
```

TypeName: System.IO.FileInfo

Name	MemberType	Definition
----	-----	-----
Mode	CodeProperty	System.String Mode{get=Mode;}
AppendText	Method	System.IO.StreamWriter AppendText()
Delete	Method	System.Void Delete()
PSPath	NoteProperty	System.String PSPath=Microsoft.PowerShell...
CreationTime	Property	System.DateTime CreationTime {get;set;}
DirectoryNam	Property	System.String DirectoryName {get;}
Extension	Property	System.String Extension {get;}
FullNa	Property	System.String FullName {get;}
IsReadOnly	Property	System.Boolean IsReadOnly {get;set;}
LastAccessTime	Property	System.DateTime LastAccessTime {get;set;}
LastWriteTime	Property	System.DateTime LastWriteTime {get;set;}
Length	Property	System.Int64 Length {get;}
Name	Property	System.String Name {get;}

Bare et lite utvalg er vist over. Dermed kan man for eksempel få tak i tidspunktet filen sist ble aksessert, lest eller sett på:

```
PS > $fil.LastAccessTime
1. mars 2010 12:26:50
```

Dette tidspunktet er også et objekt, et `System.DateTime` objekt. På samme måte kan dette objektet tilordnes en variabel

```
PS > $date = $fil.LastAccessTime
```

Og denne kan vi lese ut metoder og egenskaper fra med `Get-Member`:

```
PS > $date | Get-Member
```

TypeName: System.DateTime

Name	MemberType	Definition
----	-----	-----
AddHours	Method	System.DateTime AddHours(Double value)
IsDaylightSavingTime	Method	System.Boolean IsDaylightSavingTime()
ToLongDateString	Method	System.String ToLongDateString()
ToLongTimeString	Method	System.String ToLongTimeString()
DayOfWeek	Property	System.DayOfWeek DayOfWeek {get;}
DayOfYear	Property	System.Int32 DayOfYear {get;}

Hour	Property	System.Int32 Hour {get;}
Millisecond	Property	System.Int32 Millisecond {get;}
Minute	Property	System.Int32 Minute {get;}
Month	Property	System.Int32 Month {get;}
Second	Property	System.Int32 Second {get;}
TimeOfDay	Property	System.TimeSpan TimeOfDay {get;}
Year	Property	System.Int32 Year {get;}

Igjen er bare et lite utvalg vist. Dermed kan man trekke ut disse egenskapene, for eksempel året:

```
PS > $date.year
2010
```

Det er også mulig å gjøre dette direkte uten å gå veien om et dato-objekt:

```
PS > $fil.LastAccessTime.year
2010
```

Man kan til og med trekke det ut direkte fra kommandoen som listet filen:

```
PS > (ls ps.ps1).LastAccessTime.year
2010
```

12.20 Undersøke typen til et objekt

Med `get-member` vil vi få listet opp alle egenskaper (properties) og metoder som er tilgjengelige på det aktuelle objektet. Dette er viktig informasjon, fordi det forteller oss hva vi kan gjøre med objektene.

```
PS C:\> ls | get-member
```

```
TypeName: System.IO.FileInfo
```

Name	MemberType	Definition
----	-----	-----
...		
Length	Property	System.Int64 Length {get;}
Name	Property	System.String Name {get;}
...		

Øverst ser vi at typen til det som returneres fra `ls` er `System.IO.FileInfo`. Men vi ser bare typen til det siste elementet på denne måten. Kommandoen `ls` returnerer altså et array av `FileInfo`-objekter. Dette kan vi se ved å kalle metoden `GetType()` på returverdien slik:

```
PS C:\> (ls).GetType()
```

IsPublic	IsSerial	Name	BaseType
-----	-----	----	-----
True	True	Object[]	System.Array

For mer informasjon om de ulike typene, kan man søke dem opp på Microsofts dokumentasjonssider, <https://msdn.microsoft.com/>. Søker man for eksempel etter **System.Array** får man som første treff **Array Class**. Her finner du definisjonen av klassen og kodeeksempler i flere språk.

Om man kaller metoden `getType` for ett av elementene får man vite hva slags type dette er:

```
PS C:\> (ls)[1].getType()
```

IsPublic	IsSerial	Name	BaseType
True	True	FileInfo	System.IO.FileSystemInfo

12.21 ps

Når man istedet for å liste filer med `ls` lister prosesser med `ps`, gir dette også objekter.

```
PS > ps
Handles  NPM(K)    PM(K)      WS(K) VM(M)    CPU(s)    Id ProcessName
-----
194      6         3276      6012    49      1,20      4936 Adobe_Updater
102      5         1156      360     32      0,28      1264 alg
320      5         1528      908     22      1,28      696  csrss
```

Det som returneres fra `ps` er et array av prosessobjekter der første element er første prosess i listingen:

```
PS > $ps = ps
PS > $ps[0].name
Adobe_Updater
PS > $ps[0].id
4936
PS > $ps[1].name
alg
```

Lengden av dette arrayet vil da gi antall prosesser på maskinen:

```
PS > $ps.length
44
```

12.22 Foreach

Det er først når man bruker mulighetene objektorienteringen gir i script at PowerShell virkelig viser sin styrke:

```
foreach ($ls in ls *.ps1){
    $sum += $ls.length
}
$sum
```

eller i onelinere som dette:

```
ls | ForEach-Object {$sum += $_.Length}
```

Disse mulighetene vil bli utdypet i senere avsnitt.

12.23 Installasjon av programmer fra PowerShell

På samme måte som man installerer programmer i et Linux shell med apt-get, kan man installere programmer i PowerShell med Chocolatey; eller bare choco som kommandoen heter. Da må man først installere choco og det kan i PowerShell gjøres med:

```
wget -OutFile install.ps1 http://chocolatey.org/install.ps1
```

hvor `wget` er et alias for `Invoke-WebRequest` og virker på omtrent samme måte som i Linux. Etter å ha kjørt dette installasjons-scriptet, kan man installere annen programvare som ssh og scp med:

```
PS C:\> choco install openssh
```

og deretter bruke det fra PowerShell på samme måte som man bruker det fra bash i Linux.

12.24 Select-String, PowerShells svar på grep

For å gå igjennom et array av objekter, som for eksempel alle prosessene som listes med `ps`, er `foreach` en meget nyttig konstruksjon. Allikevel er det viktig å sjekke hva mer man kan gjøre med kommandoen `ps`, før man overkompliserer oppgaven. I bash bruker vi kommandoen

```
$ ps | grep power
```

for å finne alle prosesser med "power" i navnet. Vi har en funksjon i powershell som likner på grep, nemlig `select-string` eller kortversjonen `sls`. Men kommandoen `ps | select-string power` returnerer ikke helt det vi ønsker. Grunnen til dette er at kommandoen `ps`, i PowerShell, returnerer et array av objekter, og ikke en tekst. For å gjøre tilsvarende søk i PowerShell skriver man heller følgende:

```
$ ps power*
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
48	2	764	2696	28	0,05	2752	poweroff
257	5	28968	28196	135	0,47	5412	powershell

Her sender vi ordet `power`, med wildcardt `*`, til kommandoen `ps`, som er et alias for kommandoen `get-process`. Sjekk dette ved å skrive `get-command ps`. For detaljert informasjon om hva du kan gjøre med `get-process`, skriv `get-help get-process -detailed`

Fra Linux er vi vant til å velge ut linjer som inneholder et gitt ord med kommandoen **grep**. Det finnes ikke en helt tilsvarende PowerShell-kommando og ofte bruker man andre metoder for å oppnå det samme. Men det er mulig å bruke CmdLet'en **Select-String** som ligner:

```
ls | Select-String fil
```

er et forsøk på å velge alle linjer som inneholder ordet **fil** i ls-listingen. Men Select-String virker på objekter og går derfor inn i mapper i listingen og det virker ikke helt som ønsket. Ved å pipe output fra ls til Out-String gjøres objekt-strømmen om til vanlige strenger og man får det til å virke som i bash:

```
ls | Out-String -Stream | Select-String fil
```

Man må også ha med opsijonen **-Stream** til **Out-String** for at det skal virke.

12.25 Logiske operatorer

Operator	Betydning
-lt	Less than
-gt	Greater than
-le	Less than or equal to
-ge	Greater than or equal to
-eq	Equal to
-ne	Not equal to

Operator	Betydning
-not	Not
!	Not
-and	And
-or	Or

Merk at i powershell kan vi bruke logiske operatorer rett i shellet. I bash vil `2 -lt 3` ikke returnere noen ting, fordi det er exit-verdien av denne kommandoen som indikerer om testen slo til eller ikke. I powershell er dette litt enklere:

```
PS > 2 -eq 3
False
PS > 2 -lt 3
True
PS > "hei" -eq "hei"
True
PS > -not ("hei" -eq "heia")
True
```

12.26 Kommandoer brukt under forelesningen som ble gitt i 2018

Kommandoer¹⁶²

13 Forelesning 23/4-24(2 timer). Powershell, Windows registry

13.1 Forelesningsvideoer

Uredigert opptak av hele forelesningen (01:30:49)¹⁶³

linux12del3.mp4¹⁶⁴ (01:00) Jefferey Snover om scripting i PowerShell
linux12del4.mp4¹⁶⁵ (03:55) Demo: Funksjoner i PowerShell
linux12del5.mp4¹⁶⁶ (07:02) Demo: Scripting i PowerShell ISE, summasjon av fil-størrelser
linux12del6.mp4¹⁶⁷ (12:19) Demo: Scripting, drep prosesser, kill.ps1, argumenter
linux12del7.mp4¹⁶⁸ (05:17) Demo: PowerShell onlinere
linux12del8.mp4¹⁶⁹ (12:52) Demo: Windows Registry, endringer med SetItemProperty
linux12del9.mp4¹⁷⁰ (07:55) Demo: Sort-Object og Select-Object, Measure-Command (tar tiden)
linux12del10.mp4¹⁷¹ (06:33) Demo: DateTime-objekter
linux12del11.mp4¹⁷² (08:29) Demo: Finne filer som ble endret i et gitt tidsrom
linux12del12.mp4¹⁷³ (05:05) Demo: PowerShell arrays

13.2 Dagens sitat:Jeffrey Snover - PowerShell Architect

I use the Universal Admin Development Model. I figure out script fragments in real-time in the interactive shell. When they work, I put them into a file (can you believe that I actually use notepad?!) and then parameterize and generalize the script until it is production quality.

13.3 Windows script editor

En mulighet er å bruke Windows PowerShell ISE til å skrive PowerShell script. Da kan man få opp ett PowerShell vindu samtidig med et editor-vindu og kjøre scriptet ved å taste F5. Det finnes også mange generelle tekst-editorer for Windows som stort sett er GUI-baserte.

¹⁶²<https://www.cs.oslomet.no/haugerud/os/Forelesning/forelesningsKommandoer/Man12.03.2018.html>

¹⁶³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12.mp4>

¹⁶⁴<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del3.mp4>

¹⁶⁵<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del4.mp4>

¹⁶⁶<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del5.mp4>

¹⁶⁷<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del6.mp4>

¹⁶⁸<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del78.mp4>

¹⁶⁹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del9.mp4>

¹⁷⁰<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del10.mp4>

¹⁷¹<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del11.mp4>

¹⁷²<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del12.mp4>

¹⁷³<https://www.cs.oslomet.no/haugerud/os/Forelesning/video/linux12del1314.mp4>

En annen mulighet er å installere **nano** med `choco install nano` og deretter bruke **nano** fra kommandolinjen slik som i et Linux shell.

13.4 Funksjoner

I bash kan man lage og bruke en funksjon med følgende metode:

```
rex:~$ function hvorerjeg() { echo "Du er i $(pwd)"; }
rex:~$ hvorerjeg
Du er i /home/haugerud
```

Med ganske lignende syntaks kan man lage funksjoner i PowerShell:

```
PS C:> function hvorerjeg {"Er i $(pwd)"}
PS C:> hvorerjeg
Er i C:>
```

og det er enkelt å sende med parametre:

```
PS C:> function add($a,$b){ echo ($a + $b)}
```

13.5 Summere antall bytes i filer

Output fra PowerShell CmdLets er som vi har sett ikke bare tekst som i et bash-shell, men objekter. Dermed kan man ved hjelp av **foreach** gå igjennom alle objektene og trekke ut den informasjon man trenger:

```
foreach ($ls in ls *.ps1){
    $sum += $ls.length
}
$sum
```

Foreach er et alias for `ForEach-Object`, men når det står i starten av en setning, er det et PowerShell statement eller reservert ord, slik som `if` og `for`. Summasjon som inkluderer filer i alle undermapper får man med opsjonen `-r` til `ls`:

```
foreach ($ls in ls -r){
    if($ls.Extension -eq ".ps1"){
$sum += $ls.length
    }
}
```

Egentlig er `ls -r` et alias for `Get-ChildItem -Recurse`.

13.6 Stoppe prosesser med et gitt navn: `nkill.ps1`

En stor fordel med at kommandoene gir objekter er at man kan bruke de samme metodene på mange forskjellige typer kommandoer, for eksempel på `ps` som er et alias for `Get-Process`:

```
$s = $args[0] # Første argument

foreach ($p in ps ){
    foreach ($name in $args){
        if ($p.name -eq $name){
            kill -whatif $p.id
        }
    }
}
```

Opsjonen `-whatif` til `kill` er nyttig for å teste ut hva som kommer til å skje hvis man kjører scriptet. Når scriptet virker som det skal, kan man fjerne `-whatif`. En tilsvarende oneliner kan lages slik:

```
ps | foreach { if($_.name -eq "navn"){kill $_.id -whatif}}
```

13.7 PowerShell oneliner

Ofte kan man lage tilsvarende kraftige konstruksjoner med bare en enkelt kommandolinje, såkalte oneliners. For å gjøre det bruker man konstruksjoner som `ForEach-Object` og `Where-Object` og lage en indre løkke hvor hvert objekt behandles. Inne i en slik løkke vil den spesielle variabelen `$_` være en peker til objektet som er under behandling. Hele scriptet ovenfor kan lages som en oneliner slik:

```
ls | ForEach-Object {$sum += $_.Length}
```

Men man må på kommandolinjen passe på at variabelen nullstilles og hvis man i tillegg ønsker å skrive ut svaret kan man akkurat som i bash adskille kommandoer med semikolon:

```
$sum = 0; ls | ForEach-Object {$sum += $_.Length };$sum
```

Med `Where-Object` kan man velge ut objekter med spesielle egenskaper. For eksempel kan man plukke ut mapper fra en listing av filer og mapper på følgende måte:

```
ls | Where-Object {$_ .PSIsCointainer}
```

for `PSIsCointainer` er en `TRUE/FALSE` property som bare er sann for mapper. `Where-Object` kan kombineres med `foreach-Object`:

```
$sum = 0; ls | Where-Object {$_ .extension -eq ".txt"} | ForEach-Object {$sum += $_.Length };$sum
```

som legger sammen `Length` kun for filer med extension `.txt`.

13.8 Lese og sette registry verdier

Avsnitt 11.2.3 i Tanenbaum gir en kort innføring om Windows registry som er en mellomting mellom et filsystem og en database. Registry inneholder all konfigurasjonsinformasjon for Windows og består av en rekke keys som kan betraktes som en path. I hver key er finnes det flere values som inneholder de aktuelle konfigurasjonsverdiene, på samme måte som filer med et innhold finnes i mapper. Registry kan også sammenlignes med `/proc` i Linux, siden mange av filene der også er konfigurasjonsverdier, mens andre er informasjon om systemet. En key i Registry tilsvarer en path i `/proc` og en value i Registry tilsvarer innholdet i filen i `/proc`.

Ved å kjøre programmet `regedit.exe` med administrator-rettigheter kan man endre verdier i registry. Det kan også være nyttig å bruke `regedit` for å finne riktig navn på den key der variabelen man ønsker å endre ligger. Ofte ønsker man å automatisere prosessen med å endre registry-verdier, slik at man raskt kan sette en rekke verdier eller skru dem av og på i visse situasjoner. Hvis man ønsker å forklare andre hvordan man skal gjøre en endring, er det mye mindre sjanse for feil hvis man viser til et script enn om man sender en lang forklaring på hvor man skal klikke og hva man skal gjøre i `regedit`.

Følgende eksempel viser hvordan man setter executionpolicy for PowerShell ved hjelp av en registry setting:

```
$key = "HKLM:\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell"
Set-ItemProperty $key ExecutionPolicy Restricted
```

Her blir keyen først definert, deretter settes valuen ExecutionPolicy til Restricted. Med en mer detaljert syntaks kan verdien settes med

```
Set-ItemProperty -path $key -name ExecutionPolicy -value Restricted
```

hvor det er tydelig at key er en path. Bruker man denne path finnes det der en variabel/value med navn ExecutionPolicy og denne gis verdien Restricted. Forøvrig kan man også sette denne verdien direkte i PowerShell om man har administrator-rettigheter:

```
PS C:\> Get-ExecutionPolicy
PS C:\> Set-ExecutionPolicy Restricted
```

13.9 Sort-Object og Select-Object

Med disse to CmdLets kan man sortere og velge ut objekter. For eksempel vil følgende oneliner sortere filer etter lengde, med de største først (descending) og deretter plukke ut de fire første:

```
ls | Sort-Object Length -des | Select-Object -First 4
```

13.10 DateTime

`DateTime` er en CmdLet som uten argumenter gir et objekt som inneholder dato og klokkeslett (`DateTime`) akkurat nå:

```
PS C:\> Get-Date
søndag 19. mars 2017 19.41.59
```

Man kan også lage DateTime objekter for et vilkårlig tidspunkt:

```
PS C:\> Get-Date -Year 2016 -Month 5 -Day 17 -Hour 17 -Minute 30 -Second 00
tirsdag 17. mai 2016 17.30.00
```

```
PS C:\> Get-Date "17/5 2007 17:30"
torsdag 17. mai 2007 17.30.00
```

```
PS C:\> Get-Date "17 May 2007 17:30"
torsdag 17. mai 2007 17.30.00
```

Med utgangspunkt i en variabel som inneholder et DateTime-objekt, kan man med **Get-Member** finne egenskaper og metoder objektet har og for eksempel bruke det til å lage en ny variabel ett døgn tilbake i tid.

```
PS C:\> $now = Get-Date
PS C:\> $now
```

```
søndag 19. mars 2017 19.44.26
```

```
PS C:\> $now | Get-Member
```

```
TypeName: System.DateTime
```

Name	MemberType	Definition
----	-----	-----
Add	Method	datetime Add(timespan value)
AddDays	Method	datetime AddDays(double value)
AddHours	Method	datetime AddHours(double value)

```
PS C:\> $yesterday = $now.AddDays(-1)
PS C:\> $yesterday
```

```
lørdag 18. mars 2017 19.44.26
```

Anta at du husker at du har laget eller endret noen filer på mandag 13 mars, men ikke husker hvor de ligger. Da kan man først lage et par DateTime variabler tidlig på dagen og sent på kvelden:

```
PS C:\> $mm = Get-Date "13/3 2017 08:00" # Mandag morgen
PS C:\> $mk = Get-Date "13/3 2017 22:00" # Mandag kveld
PS C:\> $mm.DayOfWeek
Monday
```

Den siste kommandoen dobbeltsjekker at den 13 var en mandag. Deretter kan man lage en oneliner som lister alle filer som har **LastWriteTime** i dette tidsrommet:

```
PS C:\Users\haugerud> ls -r | Where-Object {$_.LastWriteTime -gt $mm -and $_.LastWriteTime -lt $mk}
```

Directory: C:\Users\haugerud

Mode	LastWriteTime		Length	Name
----	-----		-----	----
d-----	13.03.2017	17.05		.ssh
d-----	13.03.2017	18.28		mappe
-a----	13.03.2017	19.00	0	file6.txt
-a----	13.03.2017	10.38	13074	history13.03.2017

Directory: C:\Users\haugerud\.ssh

Mode	LastWriteTime		Length	Name
----	-----		-----	----
-a----	13.03.2017	17.05	189	known_hosts

Directory: C:\Users\haugerud\mappe

Mode	LastWriteTime		Length	Name
----	-----		-----	----
-a----	13.03.2017	09.05	146	hello.ps1
-a----	13.03.2017	18.36	64	loop.ps1
-a----	13.03.2017	18.25	307	nkill.ps1

Hvis man sender output til Format-Table blir det litt ryddigere og man kan velge hvilke felt man ønsker å ha med:

```
ls -r | Where-Object {$_.LastWriteTime -gt $mm -and $_.LastWriteTime -lt $mk} | Format-Table LastWriteTime,fullName
```

LastWriteTime	FullName	
-----	-----	
13.03.2017 17.05.18	C:\Users\haugerud\.ssh	
13.03.2017 18.28.23	C:\Users\haugerud\mappe	
13.03.2017 19.00.45	C:\Users\haugerud\file6.txt	
13.03.2017 10.38.02	C:\Users\haugerud\history13.03.2017	
13.03.2017 17.05.18	C:\Users\haugerud\.ssh\known_hosts	
13.03.2017 09.05.03	C:\Users\haugerud\mappe\hello.ps1	
13.03.2017 18.36.16	C:\Users\haugerud\mappe\loop.ps1	
13.03.2017 18.25.18	C:\Users\haugerud\mappe\ncill.ps1	

Hvis man ønsker en sortert liste på tidspunktet, må man sende objektene til sort (alias for Sort-Objekt) før man sender det til Format-Table:

```
ls -r | Where-Object {$_.LastWriteTime -gt $mm -and $_.LastWriteTime -lt $mk} | sort LastWriteTime | Format-Table LastWriteTime,fullName
```

LastWriteTime	FullName	
-----	-----	
13.03.2017 09.05.03	C:\Users\haugerud\mappe\hello.ps1	
13.03.2017 10.38.02	C:\Users\haugerud\history13.03.2017	
13.03.2017 17.05.18	C:\Users\haugerud\.ssh	
13.03.2017 17.05.18	C:\Users\haugerud\.ssh\known_hosts	
13.03.2017 18.25.18	C:\Users\haugerud\mappe\ncill.ps1	
13.03.2017 18.28.23	C:\Users\haugerud\mappe	
13.03.2017 18.36.16	C:\Users\haugerud\mappe\loop.ps1	

13.11 array og strenger

Et array kan defineres av en kommaseparert liste:

```
$a = 1,2,3,4
PS> $a[0]
1
PS> $a[1]
2
```

Om listen ikke er separert med komma, kan man lage et array ved hjelp av metoden Split. Denne metoden er en av mange som et string-objekt har:

```
$a = "1 2 3"
$sum = 0
foreach ($tall in $a.Split(" ")){
    $sum += $tall
}
```

En mer fleksibel metode er å først definere arrayet og deretter legge til elementer med += operatoren:

```
PS C:\> $tall = @()
PS C:\> $tall += "null"
PS C:\> $tall += "en"
PS C:\> $tall
null
en
```

Først etter at elementet er definert kan man tilordne verdier uten å bruke += operatoren:

```
PS C:\> $tall[0] = "zero"
PS C:\> $tall
zero
en
```

Man kan også starte med å definere et array med elementer på følgende måte:

```
PS C:\> $char = @("a","b","c")
PS C:\> $char[2]
c
```

13.12 Event view og PowerShell

Under forelesningen gikk Windows ned et par ganger og en student tipset om å sjekke system-loggen og antok at det skyldtes en utgått lisens. Og det stemte. Under ser vi et eksempel på hvordan PowerShell kan brukes til å lete i system-loggen ved hjelp av matching med regulære uttrykk (-match):

```
PS C:\Users\os> get-eventlog system | where-Object {$_.Message -match "license period"} | ForEach-Object {$_.TimeGenerated}
```

```
fredag 24. april 2020 12.10.29
```

```
The process C:\Windows\system32\wlms\wlms.exe (DESKTOP-KVODOUH) has initiated the shutdown of computer DESKTOP-KVODOUH
```

```
Reason Code: 0x80000000
```

```
Shutdown Type: shutdown
```

```
Comment: The license period for this installation of Windows has expired. The operating system is shutting down.
```

```
fredag 24. april 2020 11.09.55
```

```
The process C:\Windows\system32\wlms\wlms.exe (DESKTOP-KVODOUH) has initiated the shutdown of computer DESKTOP-KVODOUH
```

```
Reason Code: 0x80000000
```

```
Shutdown Type: shutdown
```

```
Comment: The license period for this installation of Windows has expired. The operating system is shutting down
```

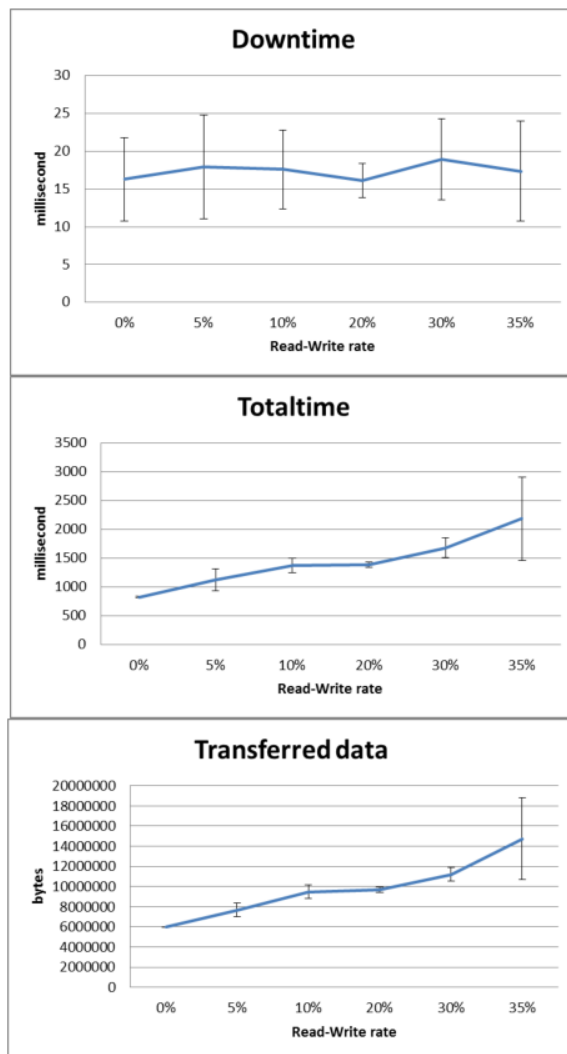


Figure 4.4: The status of migration 4M IncludeOS image

Figure 24: .

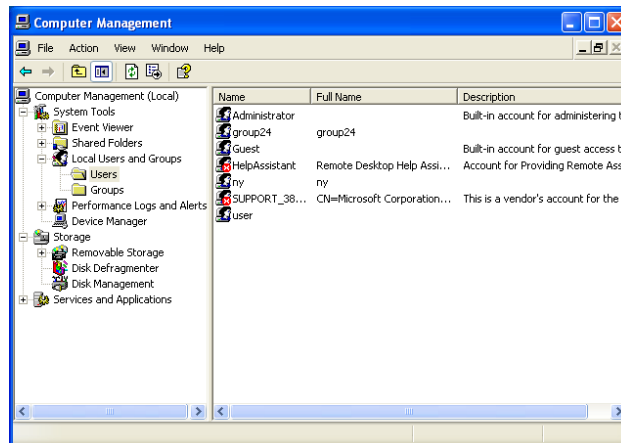


Figure 25: Under Computer Management kan man lage brukere og grupper. Dette er Win 7, ser omtrent likt ut i Win 10.

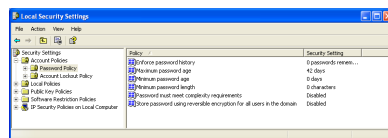


Figure 26: Under Administrative Tools → Local Security Policy kan man definere regler for passordbruk.

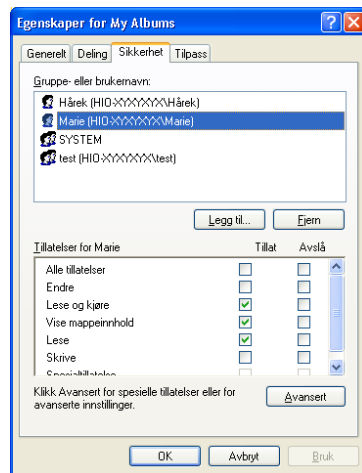


Figure 27: Med FAT filsystem vil man ikke kunne velge så detaljerte rettigheter som man her kan gjøre for en mappe på en NTFS-partisjon.