Lektionstillfälle 10

Skriptning 1

med Mikael Larsson

NACKADEMIN

Återblick

Förra gången arbetade vi med nätverk i Linux.

Vi satte upp flera VM:ar i ett litet nätverk med NFS emellan. Vi pingade, tittade på trafik etc.

Vi fick också möjlighet att felsöka konstigheter med maskin-identiteter som uppstod när VM:ar klonades.

Dagens lektion

Mål: Kunna skapa och köra skript som tar argument. Använda villkor och tester.

- Exekverbara script
- Variabler och argument
- Villkor med if
- test och ”modern test”

TLCL kapitel 24, 25, 27

Termer och begrepp

- hashbang, sha-bang, etc.
- Shell och subshell
- Variabelexpansioner
- Variabler, miljövariabler och argument
- ”here document”

Exekverbara skript

Vi har tidigare sett att man kan sätta ”exec” – rättigheter på en textfil, och på så sätt skapa ett skript.

Genom att helt enkelt skriva in shell-kommandon i textfilen, så blir filen ett skript som körs av skalet, en rad i taget.

Det vi inte tidigare pratat om är att det här är ett undantag, ett **specialfall** av exekverbara skript.

#! - hashbang

Vad som ser ut som en serietidnings-svordom, är en speciell syntax som markerar att en fil är ett skript.

”#” = hash och ”!” = ”bang”

Eftersom programmerare är lata så har det med tiden dragits ihop till bara ”sh” + ”bang” = ”she-bang”.

Såhär fungerar det:

När man ber skalet köra en exekverbar fil, så tittar det på den första raden i filen. Om raden börjar med ”#!” så använder skalet resten av raden som ett kommando att köra på själva filen.

Eftersom ”#” är **kommentarstecknet**, så påverkar det i övrigt inte skriptets funktion.

```
#!/bin/bash

echo ”hello, world!”
```

hashbang - exempel

Filen ”hello.sh”:

```
#!/bin/bash

echo ”hello, world!”
```

Vi skriver:

```
$ ./hello.sh
```

Skalet tar första raden i skriptet, plockar bort ”#!”, och kör sen kommandot med skript-filen som argument:

```
/bin/bash ./hello.sh
```

Hittar skalet ingen hashbang, och det tycker att filen ser ut att vara ”text” – så försöker det köra filens rader i ett subshell till sig självt.

Det är det specialfallet vi använt tidigare!

*OBS – filändelsen har **ingen** betydelse.*

Variabler

Eftersom shell-skript är listor med shell-kommandon, så fungerar alla kommandon och koncept vi lärt oss hittils.

Exempelvis så använder man miljövariabler för just – variabler. Precis som variabler i python eller Java.

```
#!/bin/bash

message=”hello, world!”
echo My message is $message
```

Argument

Precis som andra kommandon kan även shell-skript ta argument.

Argumenten läggs av skalet i variablerna $1, $2, et.c.

Vill man använda alla argument kan man använda ”$@”.

Man kan också plocka bort argument som redan hanterats med hjälp av ”shift”.

```
#!/bin/bash

message=$1
echo My message is $message

echo All the arguments: $@

shift
echo The rest of the arguments: $@
```

Enkla variabel-expansioner

Vi har tidigare sett att man kan expandera variabler direkt på kommandoraden eller i strängar:

```
$ echo ”I am $USER”
```

Ibland måste man avgränsa variabelnamnen:

```
$ echo ”This is ${USER}s computer!”
```

Man kan också använda expantion för att sätta värdet på variabler:

```
$ PATH=${PATH}:/opt/frasse/bin
```

Några variabel-expansioner till

Det finns många varianter på variabel-expansioner.
Här är två till som är användbara.

Kommando-expansion:

```
$ ls -l $(which man)

$ ls -l `which man`
```

OBS – bakåtfnuttar !

Aritmetisk expansion:

```
$ echo $((25 / 5))
```

OBS – bara heltals-beräkningar!

Använd citat-tecken!

När man expanderar variabler för att använda som kommandoargument är det viktigt att använda citat-tecken.

Exempel:

```
frasse@frasse:~$ touch "fil med space.txt"
frasse@frasse:~$ namn="fil med space.txt"
frasse@frasse:~$ ls $namn
ls: cannot access 'fil': No such file or directory
ls: cannot access 'med': No such file or directory
ls: cannot access 'space.txt': No such file or directory
frasse@frasse:~$ ls "$namn"
'fil med space.txt'
```

***Med** citat-tecken expanderas en variabel i **en** sträng som blir **ett** argument.*
***Utan** citat-tecken expanderas variabeln innan kommandoraden delas upp i argument, där varje mellanslag avgränsar ett nytt argument.*

”here documents”

Med citat-tecken kan värden vara flera rader långa:

```
$ echo ”Hej
på
dig!
”
```

Det finns ett annat sätt att skapa liknande strängar, som kallas för ”here documents”:

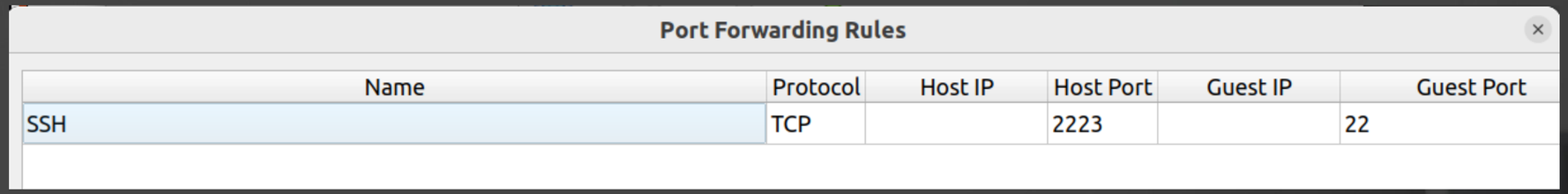
```
$ cat <<END
Hej
på
dig!
END
```

Observera att ”here documents” fungerar som en fil, som man sen kan omdirigera in i ett program.

*I det här fallet är det ”cat” som läser in filen, och skriver ut innehåller på **stdout**.*

Laboration 1

För att förenkla arbetet bör det gå att köra ssh till din VM.
Sätt upp en port forward från en ledig port på din dator 2222 eller liknande till 22 på din VM. **Inga IP ska skrivas in.**



Se labb-instruktion i portalen!

NACKADEMIN

if / then / elif / else / fi

Precis som i andra språk finns ”if / then” – konstruktionen i shell-skript.

Det kan se ut såhär:

```
if true; then
        echo it is true
else
        echo it is false
fi
```

Man kan skriva ”if” – satser direkt på kommandoraden.

test

Eftersom villkoret för ”if” är ett kommando, är det praktiskt med ett kommando som beter sig som ”vi är vana”.

```
if test a == a; then
    echo a equals a
fi
```

Viktigt:

- Jämförelseoperatorerna använder sträng-jämförelse.
- Mindre-än och större-än måste escapas!

Man kan så klart testa ”test” – kommandot direkt på kommandoraden.

Kompaktare ”test” - syntax

För all det ska se ännu mer ut som vanligt finns en variant av ”test” som heter ”[”.

Det är exakt samma program, men kräver att det sista argumentet är ”]”.

Då blir det istället:

```
if [ a == a ]; then
    echo a equals a
fi
```

Observera att ”[” alltså är ett kommando, en exekverbar fil:

```
$ which [
/usr/bin/[
```

Heltals-test

Vill man istället för strängar jämföra heltal använder man följande operatorer:

Table 27-3: `test` Integer Expressions

Expression	Is True If...
integer1 `-eq` *integer2*	*integer1* is equal to *integer2*.
integer1 `-ne` *integer2*	*integer1* is not equal to *integer2*.
integer1 `-le` *integer2*	*integer1* is less than or equal to *integer2*.
integer1 `-lt` *integer2*	*integer1* is less than *integer2*.
integer1 `-ge` *integer2*	*integer1* is greater than or equal to *integer2*.
integer1 `-gt` *integer2*	*integer1* is greater than *integer2*.

Filtest

Det finns också ett antal ”fil-test”, som liknar testen för ”find” som vi använt tidigare. Ett litet urval:

`file1 -nt file2`	*file1* is newer than *file2*.
`file1 -ot file2`	*file1* is older than *file2*.
`-b file`	*file* exists and is a block-special (device) file.
`-c file`	*file* exists and is a character-special (device) file.
`-d file`	*file* exists and is a directory.
`-e file`	*file* exists.
`-f file`	*file* exists and is a regular file.

Modern test-syntax

Den syntax vi set hittills är fortfarande den vanligast förekommande i skript. Det finns en modernare, bekvämare syntax också:

```
[[ uttryck ]]
```

Den största skillnaden är att den klarar av mönster och reguljära uttryck.

Exempel med mönster:

```
[me@linuxbox ~]$ FILE=foo.bar
[me@linuxbox ~]$ if [[ $FILE == foo.* ]]; then
> echo "$FILE matches pattern 'foo.*'"
> fi
foo.bar matches pattern 'foo.*'
```

Den moderna syntaxen är inbyggd i Bash, det är inte ett externt kommando.

Modern heltals-syntax

Ännu enklare är det för heltals-testerna. Då skriver man:

```
(( uttryck ))
```

Uttrycket tolkas då som ”truthy”, mao är noll falskt, icke-noll är sant.

```
[me@linuxbox ~]$ if ((1)); then echo "It is true."; fi
It is true.
[me@linuxbox ~]$ if ((0)); then echo "It is true."; fi
[me@linuxbox ~]$
```

Man kan också använda vanliga jämförelseoperatorerna med heltal här:

```
$ if ((2 >= 1)); then echo hurra; fi
```

Logiska operatorer

För att kombinera test-uttryck kan man använda följande logiska operatorer:

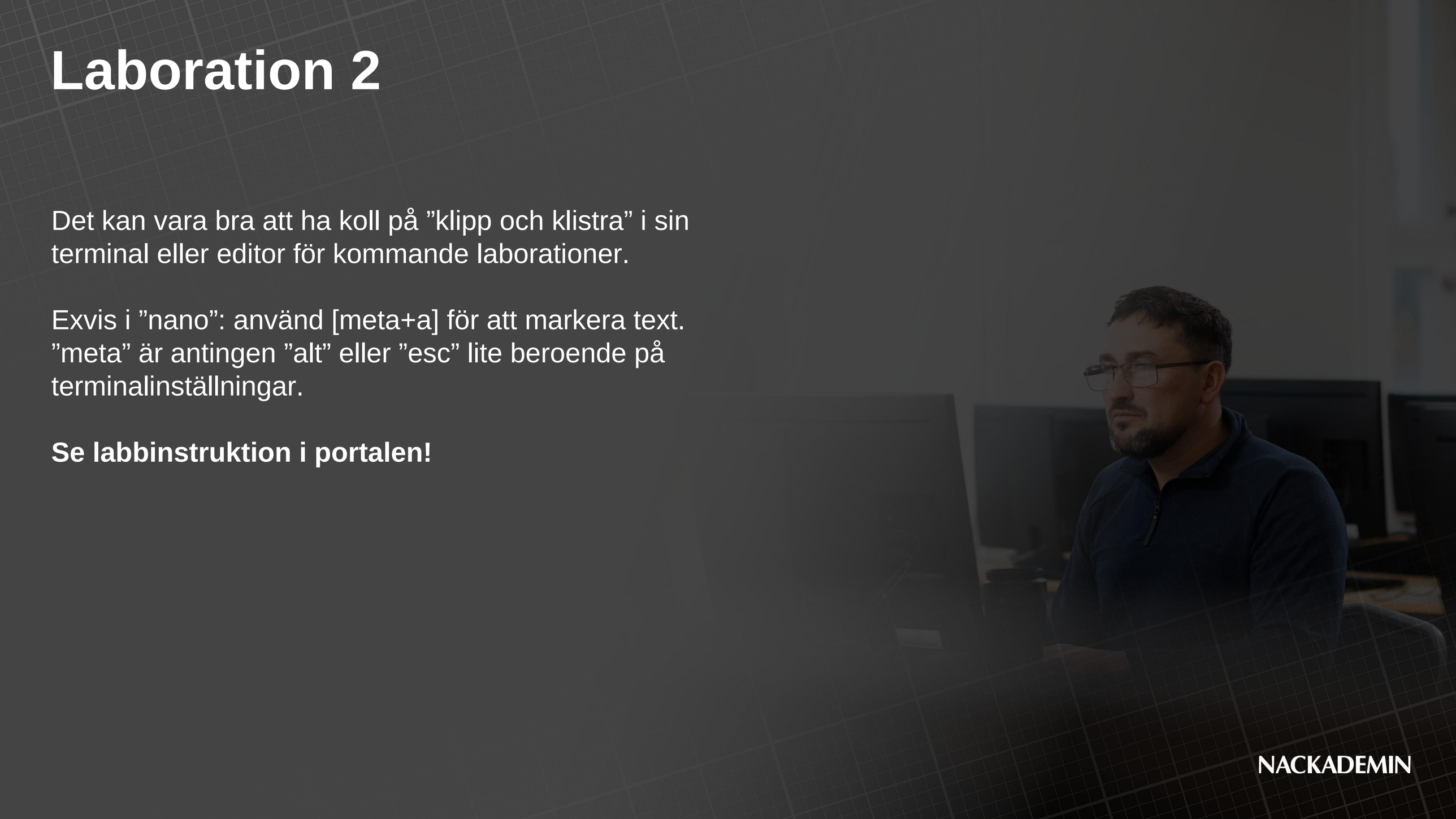
Operation	`test`	`[[ ]]` and `(( ))`
AND	`-a`	`&&`
OR	`-o`	`||`
NOT	`!`	`!`

Laboration 2

Det kan vara bra att ha koll på ”klipp och klistra” i sin terminal eller editor för kommande laborationer.

Exvis i ”nano”: använd [meta+a] för att markera text. ”meta” är antingen ”alt” eller ”esc” lite beroende på terminalinställningar.

Se labbinstruktion i portalen!



NACKADEMIN

Summering

Idag har vi äntligen börjat skripta lite på riktigt.

Vi har utforskat argument, variabelexpansion, tester och ”if”-satser.

Vad är lika och olika jämfört med tidigare programmeringskurser / andra språk?

Nästa gång

Mål: Kunna skapa skript med iteration och funktioner.

- Funktioner och lokala variabler
- Loopar med for och while
- Beräkningar
- Inläsning med while read

Stort tack!

NACKADEMIN