Lektionstillfälle 12

Textbehandling

med Mikael Larsson

NACKADEMIN

Återblick

De två tidigare lektionerna gick vi igenom grunderna i shell-skriptande.

Förra lektionen läste vi in text-filer rad för rad och gjorde olika beräkningar.

Dagens lektion

Mål: Att kunna hitta, hantera och söka i loggfiler.
Förstå och skapa reguljära uttryck.

- Loggfiler, less och tail
- Textbehandling: grep, sed, cut, sort, uniq
- Reguljära uttryck
- **Inlämningsuppgift 2**

TLCL kapitel 19, 20

Termer och begrepp

- tail, att taila
- grep, att greppa
- regex
- ERE

Att arbeta med loggfiler och strukturerad text

Allt i Linux är filer, som vi sett.
Alla loggar är textfiler, som oftast hamnar i ”**/var/log**”.
Alla inställningar ligger i textfiler, många ligger någonstans under ”***/etc***”.

Förra lektionen läste vi in textfiler rad för rad och gjorde en del bearbetningar av inläst data.

Eftersom det här är så vanligt finns det ett antal kommandon som är till just för detta.

I kombination med skriptande är de verktyg vi tittar på idag de ni nog kommer använda mest när det gäller att bearbeta textfiler.

less och tail

Använd ”less” för att titta på en statisk fil, eller om du behöver blädda och söka för att hitta det du är ute efter.

Använd ”tail -f” för att bevaka en loggfil. Du får på så sätt rullande uppdateringar direkt i terminalen.

Att använda ”tail -f” kallas för att ”taila”.

Även less kan följa en loggfil men det aktiveras inne i less med ett stort F. [shift]+F
Då kan man följa en loggfil och samtidigt se sökresultat.

grep

Vi har hittills använt ”grep” för att söka efter ord och filtrera strömmar och filer.

Det är vanligt att man kombinerar ”tail -f” med ”grep” för att få en filtrerad ström av uppdateringar, eller för att tydligt se när något specifikt händer.

Men – ”grep” kan göra mycket mer!

Ordet ”grep” kommer från ett gammalt editor-kommando och står för ”Global Regular Expression Print”.

Sökuttrycket man anger till ”grep” kan alltså vara en **regex**, något som vi kommer titta på strax.

cat och cut

Som vi tidigare sett används ”cat” för att sätta ihop (con**cat**enate) filer. Många gånger anger man bara en fil, som då skrivs till **stdout**.

Förra lektionen använde vi ”**while read**”-konstruktionen i skript för att läsa in en rad i taget och dela upp den i fält.

Det är precis vad ”cut” gör, läser en fil, rad för rad och skriver sen ut de fält från varje rad som vi ber om:

```
$ cut –f 1,6 –d : /etc/passwd
```

Kommandot skriver ut fält 1 och 6 från password-filen, där fälten avgränsas med kolon.

TLCL går igenom fler verktyg än vi gör här i presentationer och labbar.

Ni behöver inte kunna alla verktyg som boken går igenom utantill!

cut med teckenindex

”cut” kan också klippa ut delar av rader baserat på kolumnindex i antal tecken med ”-c”:

```
frasse@frasse:~/scripting/lab10$ ip link show dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group default qlen 1000
    link/ether 08:00:27:c7:22:e5 brd ff:ff:ff:ff:ff:ff
frasse@frasse:~/scripting/lab10$ ip link show dev enp0s3 | tail -1 | cut -c 5- | cut -f 2 -d' '
08:00:27:c7:22:e5
```

sort

TLCL s277

Även ”sort” har vi använt tidigare.

Men ”sort” kan också dela upp rader i fält och sortera på angivna fält.

Här används ”-k” för att ange fält (**k**eys) och ”-t” för fältseparator.

Exempel:

```
$ sort -k6r -t: /etc/passwd
```

Kommandot sorterar password-filen på sjätte fältet, i fallande (reverse) ordning.

uniq

”uniq” eliminerar eller rapporterar upprepade rader i en fil eller ström.

För att hitta dubletter måste indatat vara sorterat, så kombinationer av ”sort” och ”uniq” är vanliga.

Vanligaste flaggorna till ”uniq”:

”uniq -d” – skriv bara ut de rader som har dubletter

”uniq -c” – skriv ut antalet upprepade rader som påträffats

diff

”diff” visar skillnaden mellan textfiler.

Det finns många flaggor och formatinställningar till ”diff”, men det viktigaste är att känna till verktyget.

Det används kanske mest under utveckling när man vill jämföra versioner av källkodsfiler.

Man kan jämföra filer och/eller mappar.

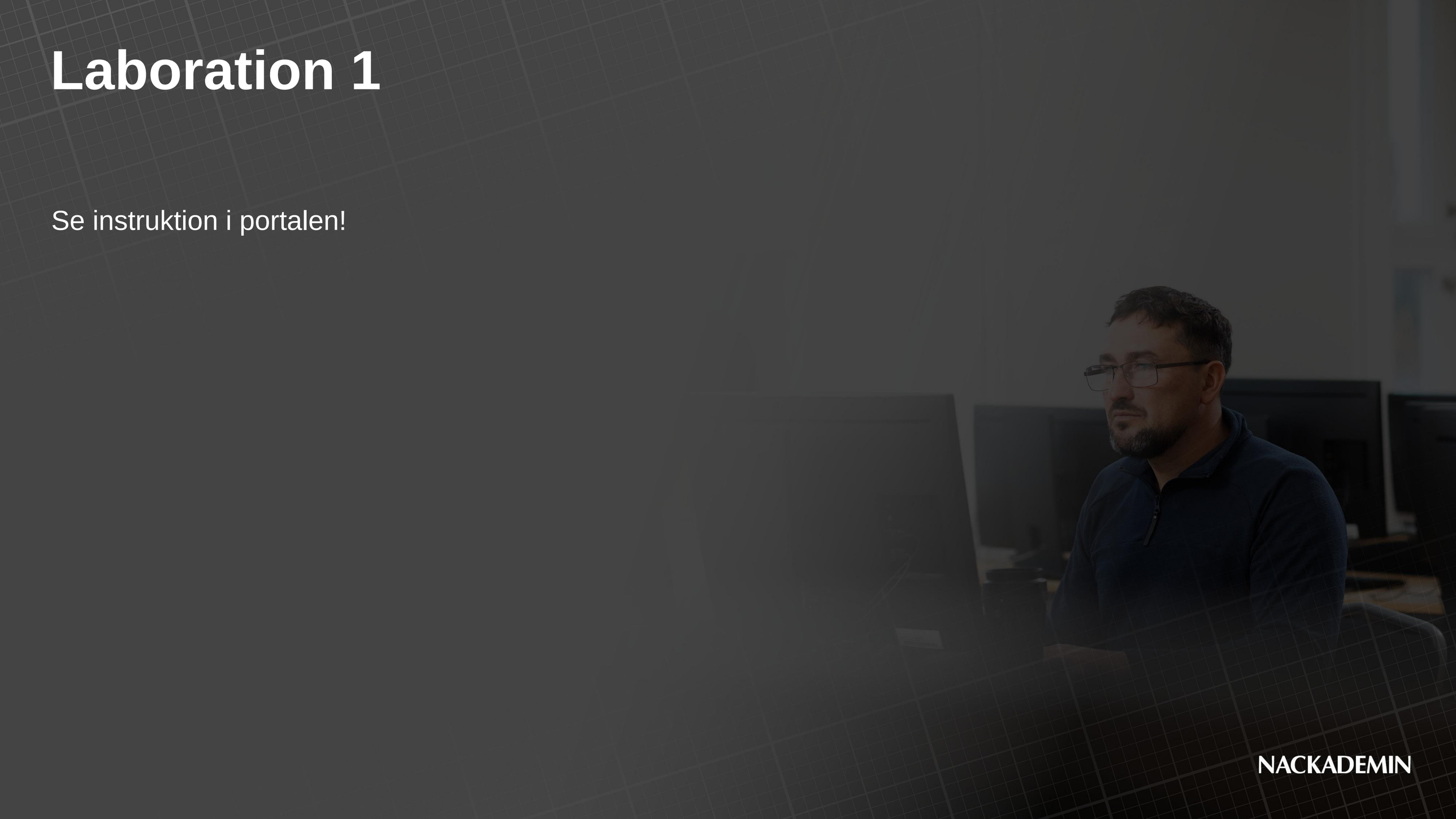
```
$ diff mycopyofpasswd /etc/passwd
$ diff mycopyofpasswd /etc
```

Med flaggan ”-r” (recursive) kan man jämföra filträd.

Om det inte finns några skillnader får man ingen utskrift från ”diff”.

Laboration 1

Se instruktion i portalen!



NACKADEMIN

Reguljära uttryck, regexar, regexp...

TLCL kapitel 19

Reguljära uttryck är ett slags språk för att söka i text.

Förutom att bara hitta en sökträff kan uttrycket även dela in text i olika grupper.

Grupperna kan sen behandlas separat.

Mer generellt kan man säga att regex **hittar mönster** i text.

Det finns många dialekter av **regex**, men grunden är densamma.

*Reguljära uttryck liknar jokertecken (wildcards), och fungerar ibland snarlikt, men är **inte** samma sak.*

Vi kommer använda oss av Extended Regular Expressions, ERE.

regex: några grundregler

Följande metatecken har speciell betydelse:

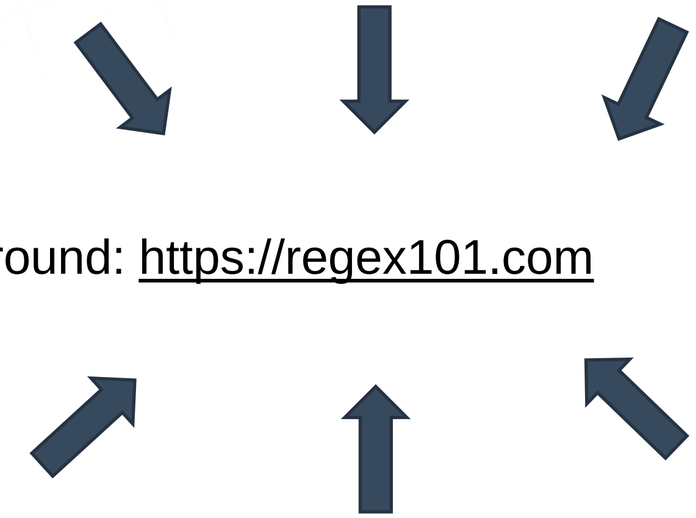
**^ $. [ ] { } - ? * + ( ) | **

Alla andra tecken är ”vanliga” och matchar sig själva.

Regex playground: https://regex101.com

Metatecken	Matchar
^	Början på text
$	Slutet på text
.	Vilket tecken som helst
*	Noll eller flera av uttrycket till vänster
+	En eller flera av uttrycket till vänster

regex: några grundregler



Följande metatecken har speciell betydelse:

**^ $. [ ] { } - ? * + ( ) | **

Alla andra tecken är ”vanliga” och matchar sig själva.

Regex playground: https://regex101.com

Metatecken	Matchar
^	Början på text
$	Slutet på text
.	Vilket tecken som helst
*	Noll eller flera av uttrycket till vänster
+	En eller flera av uttrycket till vänster

regex: några exempel

Regex	Teststräng	Match?	Resultat
abba	abba	Ja	**abba**
ab+a	abba	Ja	**abba**
ab.a	abca	Ja	**abca**
ab.a	aba	Nej	
bc	abca	Ja	**bc**
^z.o$	zorro	Nej	
^z.o$	zro	Ja	**zro**
^z.*o$	zorro	Ja	**zorro**
^z.*o$	zabadabadooo	Ja	**zabadabadooo**
^z.*o$	zo	Ja	**zo**
^z.+o$	zo	Nej	
^z.+o$	zro	Ja	**zro**
or	sorry, zorro!	Ja	***Det beror på!***

grep med regex

För att kunna använda full regex-syntax måste man istället för ”grep” använda ”grep -E”, alternativt ”egrep” som gör samma sak.

Filer i /bin som börjar på ”e”:

```
$ ls /bin | egrep ”^e”
```

Filer i /bin som slutar på ”e”:

```
$ ls /bin | egrep ”e$”
```

Med parenteser kan man gruppera matchningar, och med ”|” kan man skapa ”eller” – matchningar.

Filer i /bin som börjar på ”e” eller slutar på ”e”:

```
$ ls /bin | egrep ”(^e|e$)”
```

Fler regler

Uttryck som reglerar hur många gånger något ska matchas kallas kvantifierare (quantifiers).

De skrivs alltid direkt **efter** uttrycket som ska matchas.

Exempelvis ”a{,4}” betyder max fyra ”a” efter varandra.

Kvantifierare	Matchar
?	Noll eller en - även som lazy-switch
*****	Noll eller flera
+	En eller flera
{n}	Exakt n stycken
{n,}	Minst n stycken
{,m}	Max m stycken - kan behöva skrivas {0,m}
{n,m}	Minst n, max m stycken

Teckenklasser

Tecken som skrivs inom hakparenteser definierar teckenklasser (character classes) och liknar de uttryck som skalet använder.

Inleds en teckenklass med ”^” inverteras matchningen.

Exempel:

Teckenklass	Matchar
[abc]	En av ”a”, ”b” eller ”c”
[a-c]	Samma som ovan, ”a” till ”c”
[a-cd]	”a” till ”c” och ”d”
[^0-9]	Alla tecken som inte är 0 till 9
[[:word:]]	”word characters”
[[:space:]]	”whitespace”

*OBS – även om en teckenklass definierar ett intervall eller en lista av tecken, så matchar själva uttrycket med hakparenteser bara **ett** tecken.*

För att matcha fler måste kvantifierare användas!

sed

”sed” står för ”Stream EDitor” och redigerar strömmar med text med hjälp av ett eget skriptspråk och **regex**.

”sed” är ett djupt kommando med alltför mycket funktioner. Ni behöver bara känna till ”sök och ersätt”, som är mycket användbart:

```
$ sed -e 's/:/;/gi' /etc/passwd
```

Syntaxen
s/<pattern>/<replacement>/<modifiers>
kommer igen i andra verktyg.

”s” står för ”substitute”
”/” är separator, kan vara vilket tecken som helst
”g” är den vanligaste modifieraren, som aktiverar ”global” matching
”i” står för ”ignore case”

”replacement” – delen kan referera till grupper i ”pattern” – delen med \1, \2, et.c.

Även ”sed” behöver köras med ”-E” – flagga för att acceptera Extended Regular Expressions.

Några sed-exempel

sed-kommando	resultat
`sed -e 's/alldrig/aldrig/'`	Fixa en felstavning av ”aldrig” (den första per rad)
`sed -e 's/alldrig/aldrig/g'`	Fixa alla felstavningar av ”aldrig”
`sed -e 's/^#/# COMMENT:/'`	Byt ut det första tecket på alla rader som börjar med ”#” till ”# COMMENT:”
`sed -E -e 's/\s+/ /g'`	Ta bort dubbletter av whitespace
`sed -E -e 's/[0-9]+/(\0)/g'`	Sätt alla tal (grupper med decimala siffror) mellan parenteser

Ett lite krångligare exempel:

```
$ sed -Ee 's/([0-9]+)\/(\w+)/Port: [\1], Proto: [\2]/' < /etc/services
```

Konverterar rader i “/etc/services” som ser ut såhär:

```
echo            7/tcp
```

till:

```
echo            Port: [7], Proto: [tcp]
```

tr

”tr” , translate, översätter en lista med tecken till en annan lista med tecken.

Kan också användas för att rensa bort tecken.

Går att göra med ”sed” också, men ”tr” är enklare.

Exempel:

```
$ cut -d: -f1 /etc/passwd | tr ’a-z’ ’A-Z’
```

Fler verktyg som kan regex

Förutom grep och sed kan man använda regex i flera av de kommandon vi använt redan:

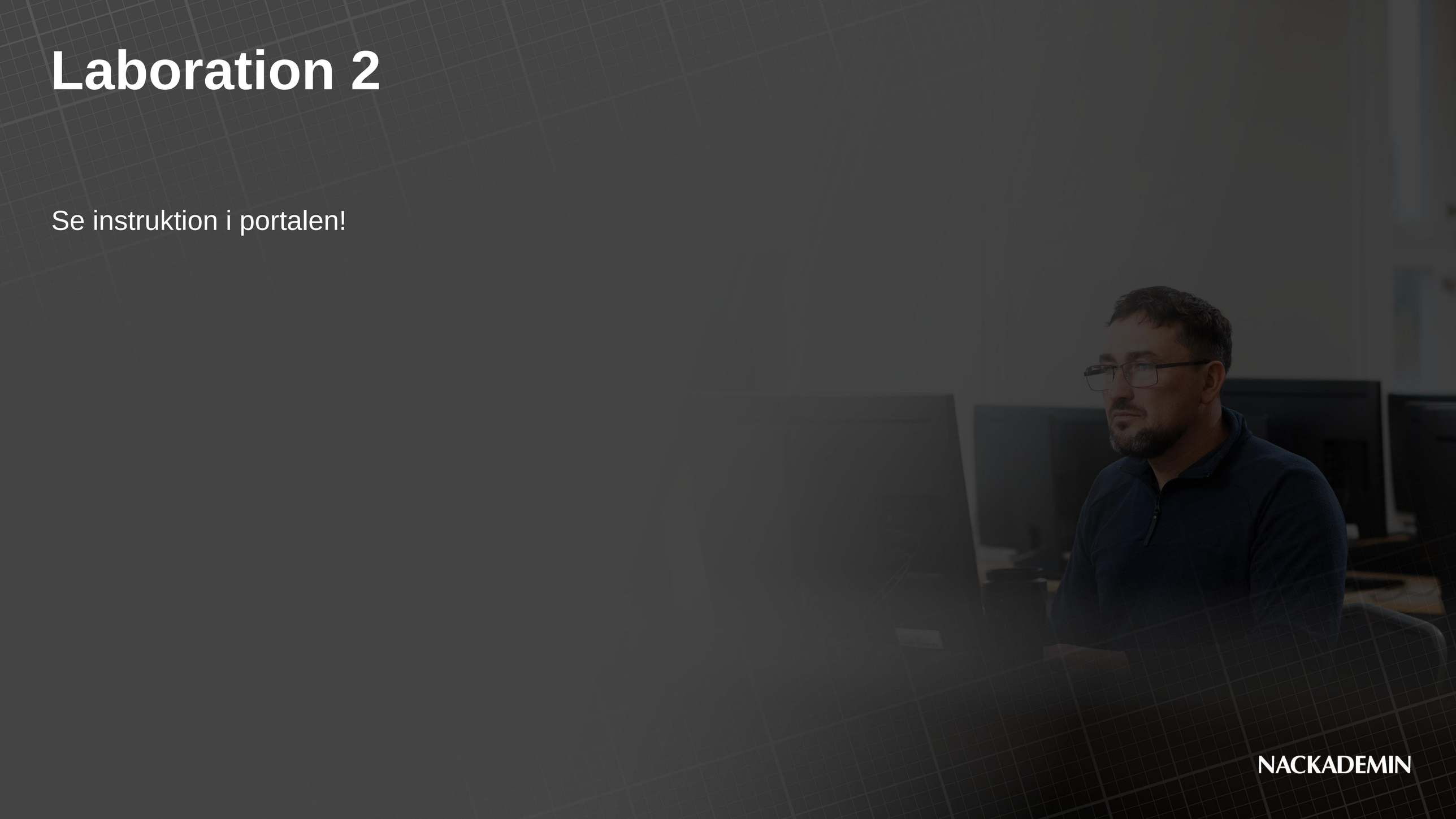
- find
- locate
- less
- vi
- nano

De flesta programmeringsspråk har någon form av regex inbyggt eller som bibliotek.

Javascript har anammat ”/” – syntaxen för att definiera regex.

Laboration 2

Se instruktion i portalen!



NACKADEMIN

Inlämningsuppgift 2

- Uppgiften ligger i portalen
- Uppgiften utförs och lämnas in på samma dator som tidigare
- Uppgiften är individuell, man får inte lämna in i grupp
- Läs igenom uppgiften **helt** innan ni börjar
- Fråga om något är oklart!

Summering

Idag har vi utökat vår verktygslåda med text-kommandon.

Vi har jobbat med ERE, Extended Regular Expressions i verktygen ”grep” och ”sed”.

Vi kan skilja ”cut” från ”cat”, och vi kan både taila och greppa.

Nästa gång

Mål: Att kunna starta och stoppa tjänster samt ändra schemaläggning och förstå uppstart.

- Bootstrapping
- Systemtjänster
  - systemctl, service
- Schemalagda aktiviteter
  - crontab, systemd
- Runlevels

Stort tack!

NACKADEMIN