



Föreläsning III

DVA249/DVA267

Linux

Peter Backeman

IDT/NetCenter

DVA249/DVA267 HT2023



Föreläsningens innehåll

- Tar upp mycket av det som står i litteraturen
- Täcker inte allt
- Workshop på fredag för repetition och förtydligande

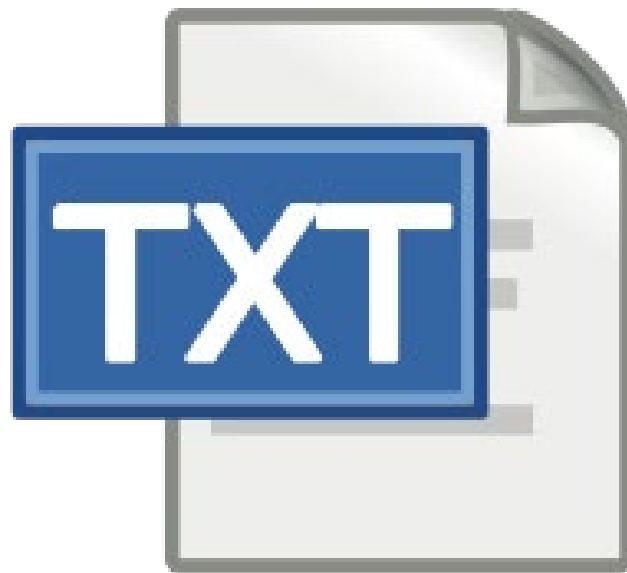
Agenda

- Text(filer)
- Reguljära uttryck (regular expressions)
- Redirection (>, <)
- Pipes (|)
- Mera skript (if, for, ...)

(Text(filer) **)**

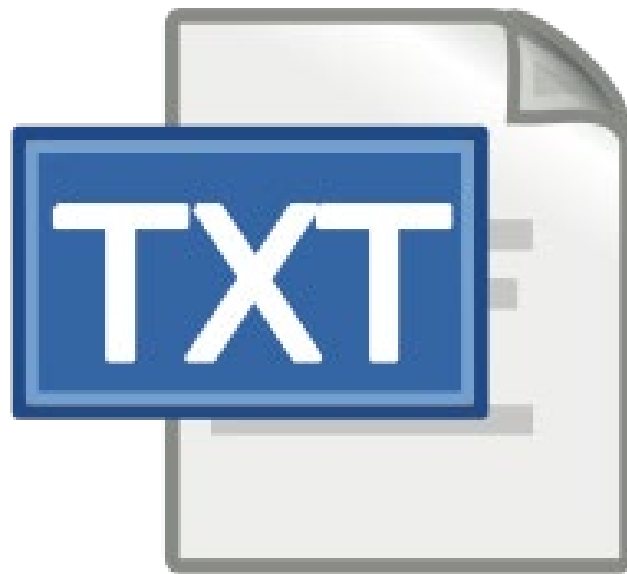
Läsa textfiler

- Två användbara kommandon
- `cat`
 - Skriver ut filen i terminalen
- `less`
 - Kan bläddra upp/ned, söka, ...
- Finns även `more`, men strunta i det ...



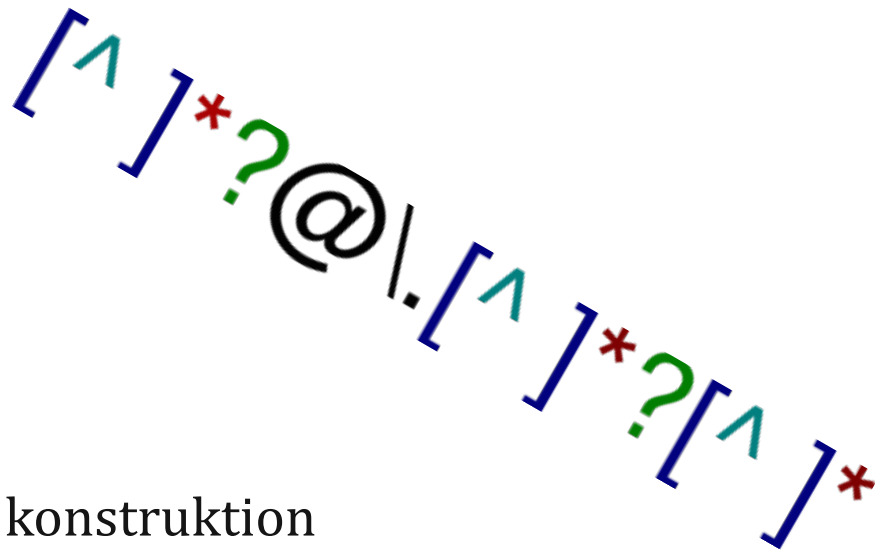
Leta i textfiler

- `grep`
- Ger alla rader som innehåller en sträng
- Går att använda *(utökade) reguljära uttryck*
- Mycket användbart!



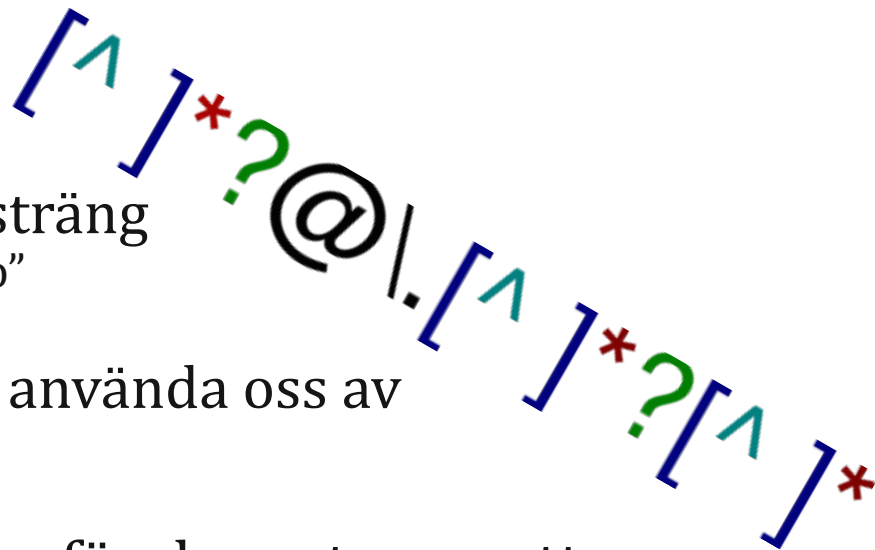
Reguljära Uttryck

- “Regular Expressions” (Regex)
- Finns på många ställen
 - `grep -E`
 - *Python, Java, Scala, ...*
- “Reguljära språk” en matematisk konstruktion
- I denna workshop använder vi med `grep`



En regex säger mer än tusen strängar

- Ett regex matchar strängar
- Enklaste uttrycket är bara en sträng
 - T.ex. "mango" matchar bara "mango"
- Finns sedan operatorer vi kan använda oss av
 - . , * , () , ^ , \$
- Kom ihåg att `grep` letar efter förekomster av ett regex
 - Ifall en sträng innehåller regex någonstans så matchar den



Punkt och Stjärna

- Påminner lite om globbing
- . fungerar som ?
- Matchar exakt ett tecken
- Till exempel så matchar "h.j":
 - hej, haj, hoj, ...

$[^?@|.[^?[^?]$

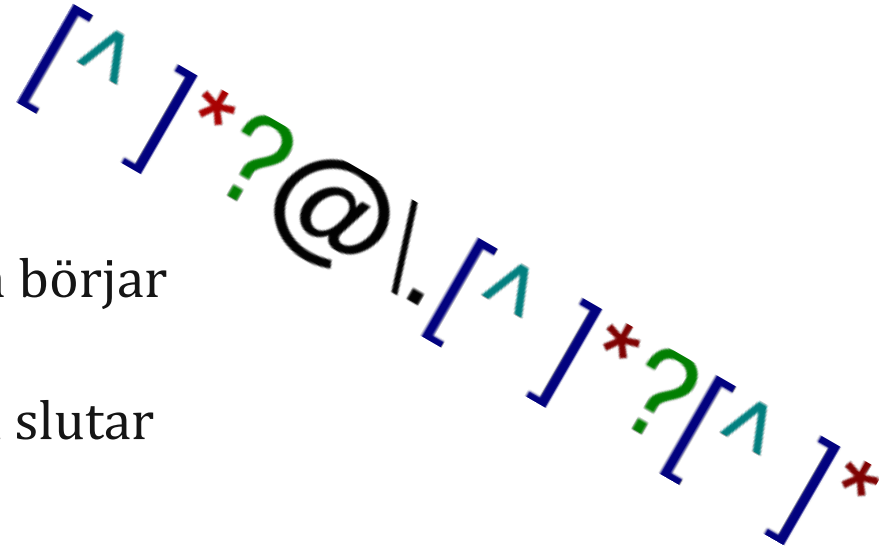
Punkt och Stjärna

- Påminner lite om globbing
- * fungerar inte som * i globbing!
- I regex betyder * upprepa föregående tecken noll eller flera gånger.
- Till exempel så matchar "h*j":
 - "hj", "hhhhj", "hhhhhhhhhhhhj", "j", ...

[^]*?@|.[^]*?[^]*?

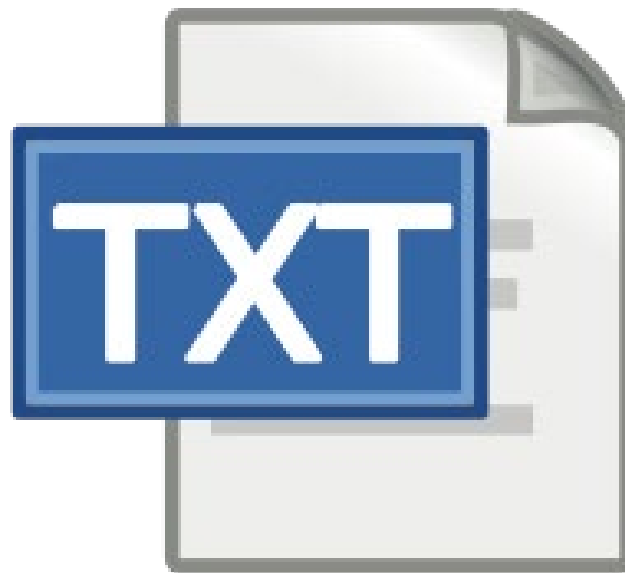
Ankare (regex i grep)

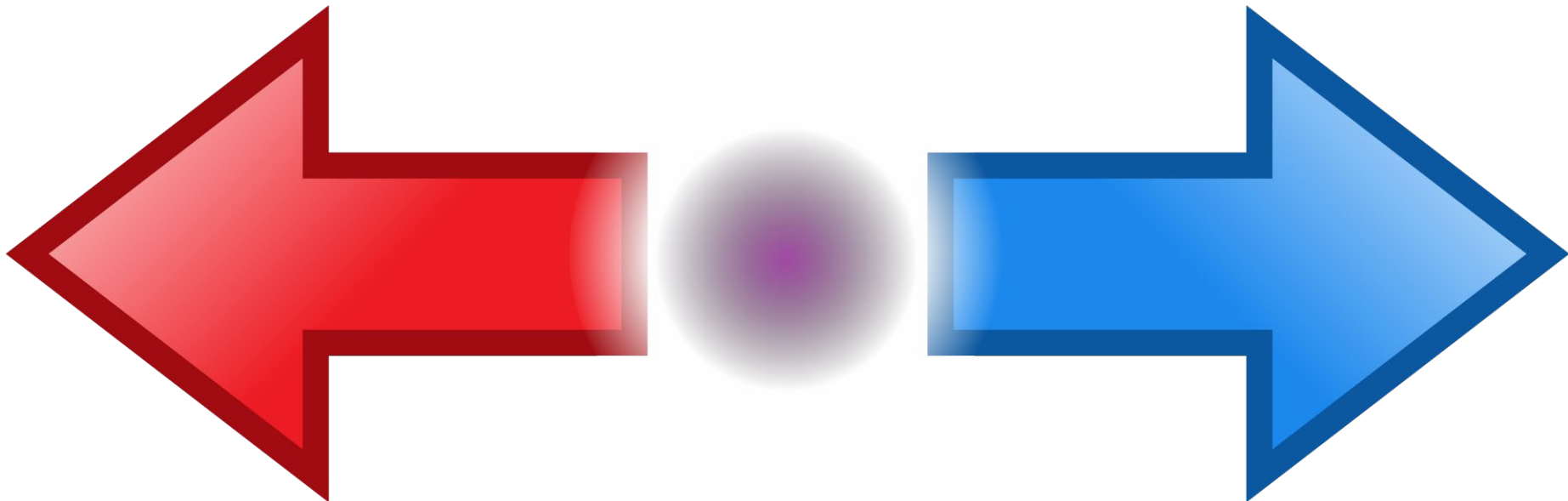
- Markerar början/slutet av raden
 - `^` markerar början av raden
 - `$` markerar slutet av raden
- `"^pl.*"` matchar alla strängar som börjar med "pl"
- `".*as$"` matchar alla strängar som slutar med "as"
- `"^pl.*as$"` matchar alla strängar som börjar med "pl" och slutar med "as".



Ändra textfiler

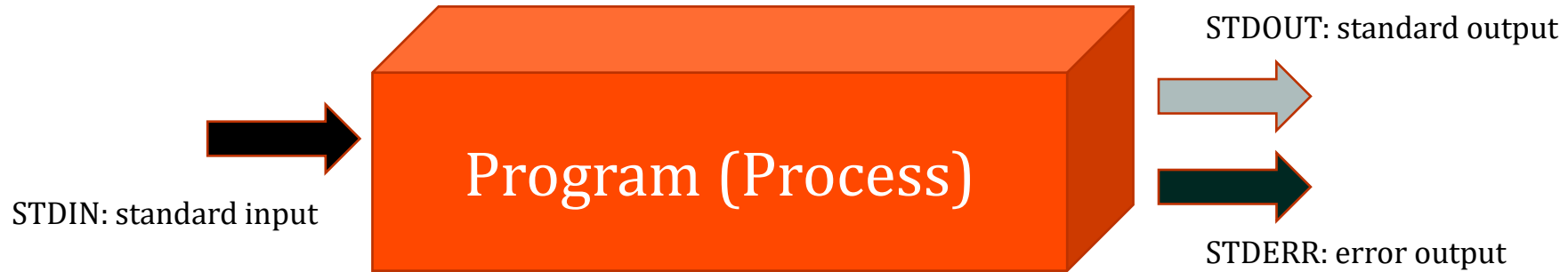
- nano
 - Väldigt simpel texteditor
- *vi/vim*
 - Mer avancerad texteditor
- emacs, pico, ...
- Senare i presentationen: många små program





Pilar och Rör

Input & Output



Pilar: omdirigering av input/output

- Normalt
 - STDOUT skrivs ut i terminalen
 - STDERR skrivs ut i terminalen
 - (STDIN läses in från terminalen)

- Vill spara output?
 - Styr om med hjälp av *omdiregering (redirection)*



Omdiregering

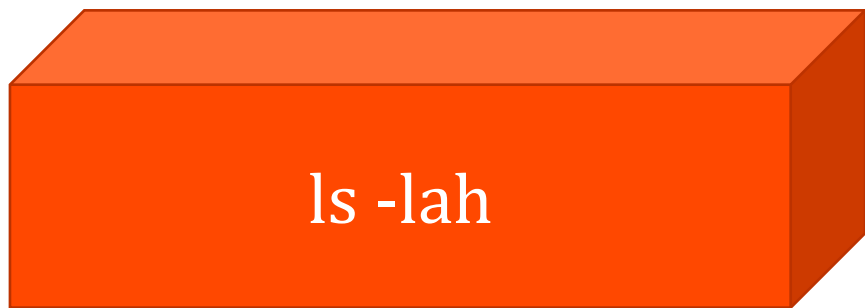
- Egentligen: *Redirection operators*

> <

- Här: pilar



Från Program till Fil



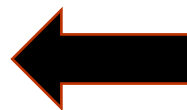
lista.txt

ls -lah

>

lista.txt

Från Fil till Program



lista.txt

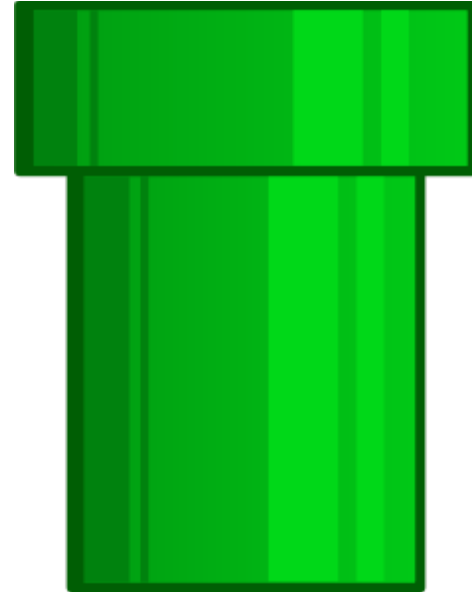
`wc -l`

<

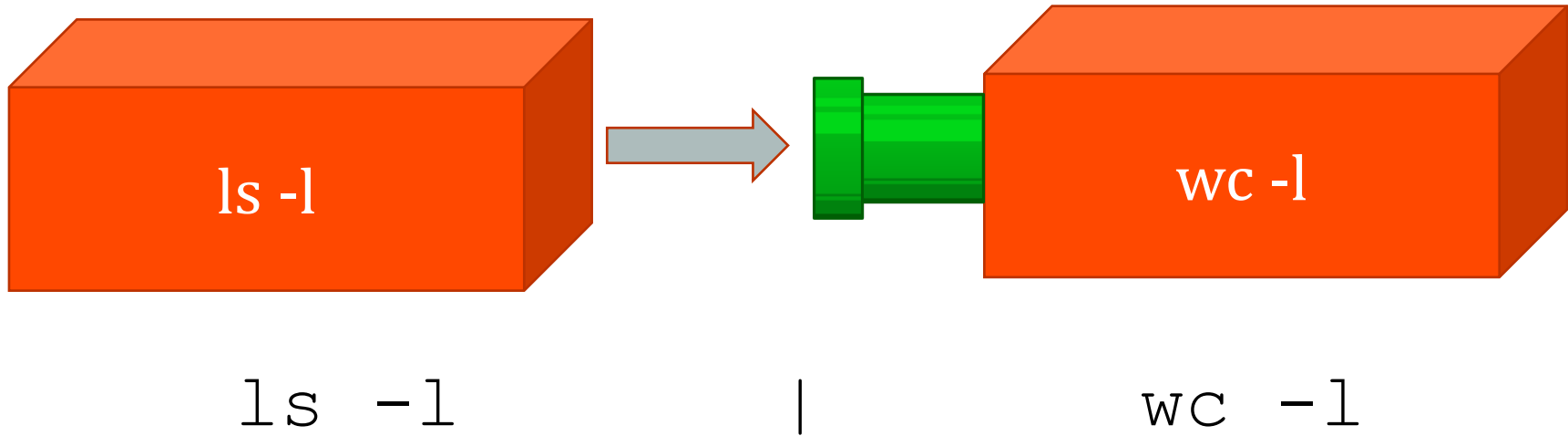
`lista.txt`

Rör (pipes)

- Vi har sett hur pilar styr om från program till filer
- Rör tillåter oss att styra om från program till *andra program*
- Ett rör skrivs som tecknet | .

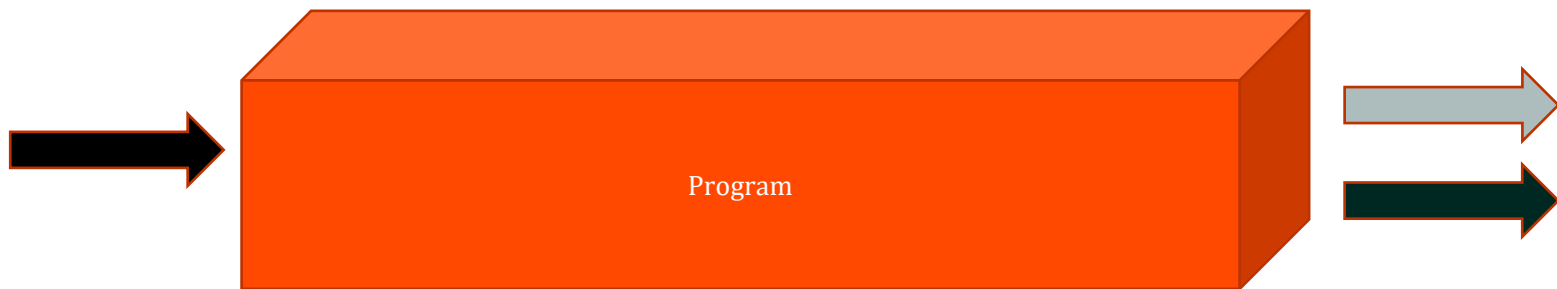


Från Program till Program



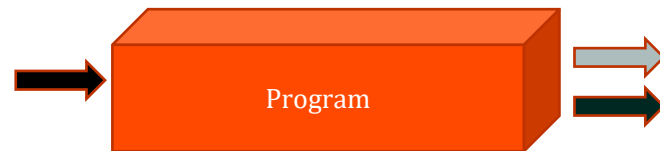
Strömmar

- Hittills bara jobbat med STDIN/STDOUT
- Ett program har två outputströmmar, STDOUT och STDERR
- STDOUT = "vanlig" output, STDERR = felmeddelande



Strömmar

- Hittills bara jobbat med STDIN/STDOUT
- Ett program har två outputströmmar, STDOUT och STDERR
- Vi kan även dirigera om strömmarna
 - 1 = STDOUT
 - 2 = STDERR
 - 1> fil - styr om STDOUT till fil (samma som > fil)
 - 2> fil - styr om STDERR till fil
 - &>fil - styr om STDOUT och STDERR till fil



(Mer) Text

Textmanipulation via *cut* och *awk*

- Några kommandon som är användbara för att plocka ut text från output
- `cut` klipper ut en textsträng
- `tr` "översätter" bokstav-för-bokstav
- `sed` byter ut strängar
- `awk` är väldigt kraftfullt, men här kollar vi bara på enkel användning



cut

- Klipper ut en bit av strängen
- Går att ange start- och slutposition på olika sätt
 - $-c$ *Characters* (tecken)
 - $-f$ *Fields* (fält)
- När man använder fält så kan man också ange *delimiter* (avgränsare) med hjälp av flaggan $-d$ (TAB är standard).



tr

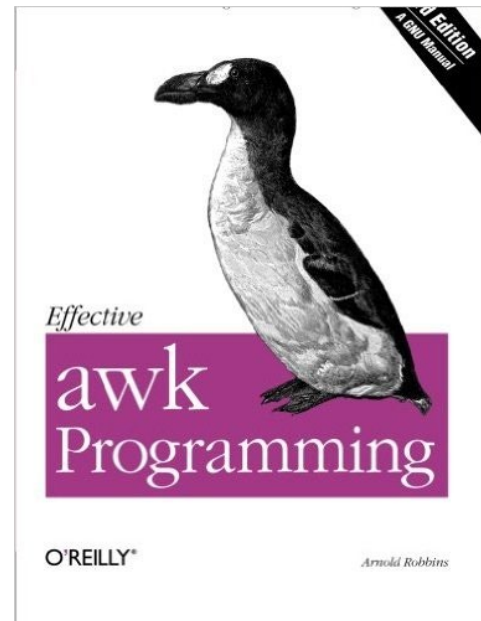
- Enkelt verktyg
- *translate*
- Byter ut bokstav-för-bokstav
- Kan också "squeeze" bort dubletter

sed

- Lite mer komplicerat verktyg
- *stream editor*
- Vi kommer främst använda på följande form
 - `sed "s/ord1/ord/g"`
 - byter ut alla ord1 mot ord 2 (substitute, global)

awk

- Kraftfull programmeringsspråk
- `awk '{script}' input`
- I skript så använder vi `print`
 - Vi kan använda `$1` för att skriva ut första fältet
- Flaggan `-F` ställer in vad som är avgränsare



Skript

Skript



- Vi har sett enkla skript (kommandon på följd)
- If-satser
 - Låt returvärdet (exit code) av ett program avgöra vad som ska hända
- Loopar
 - Finns två varianter: BASH-stil och C-stil
 - Iterera över en lista av värden

If-satser

- Ni har sett detta i programmering tidigare
- Om ett villkor är sant, så ska något göras
 - Ofta använder man returvärden från program



```
if villkor; then
    statement
fi
```

```
if villkor; then
    statement
else
    statement
fi
```

Command	Description
<code>test -f /dev/ttyS0</code>	0 if the file exists
<code>test ! -f /dev/ttyS0</code>	0 if the file doesn't exist
<code>test -d /tmp</code>	0 if the directory exists
<code>test -x `which ls`</code>	substitute the location of <code>ls</code> then <code>test</code> if the user can execute
<code>test 1 -eq 1</code>	0 if numeric comparison succeeds
<code>test ! 1 -eq 1</code>	NOT - 0 if the comparison fails
<code>test 1 -ne 1</code>	Easier, <code>test</code> for numeric inequality
<code>test "a" = "a"</code>	0 if the string comparison succeeds
<code>test "a" != "a"</code>	0 if the strings are different
<code>test 1 -eq 1 -o 2 -eq 2</code>	<code>-o</code> is OR: either can be the same
<code>test 1 -eq 1 -a 2 -eq 2</code>	<code>-a</code> is AND: both must be the same

For-loopar



- Ni har sett detta i programmering tidigare

- **BASH-stil**

```
fruits="Mango Pineapple Papaya Watermelon"
for fruit in $fruits; do
    echo "I like $fruit"
done
```

- **C-stil**

```
for ((i = 0 ; i < 5 ; i++)); do
    echo "Remember to backup important files!"
done
```

Felmeddelande i bash-skript

- Tyvärr inte de bästa...
- Vanligt exempel, man glömmer något i en for-loop.

Specialvariabler



Variabel	Beskriving
\$0	Filnamnet
\$1-\$9	Argument 1-9
\$#	Antalet argument
\$@	Alla argumenten som en array
(\$*	Alla argumenten som en sträng)
\$?	Returvärdet av senaste kommandot

Exempel:



- Ett skript som kollar första argumentet och:
 - Ifall det är lika med "second" så ska skriptet skriva ut det andra argumentet
 - Ifall det är lika med "third" så ska skriptet skriva ut det tredje argumentet

Variabel

\$0

\$1-\$9

\$#

\$@

(\$*)

\$?

Beskriving

Filnamnet

Argument 1-9

Antalet argument

Alla argumenten som en array

Alla argumenten som en sträng

Returvärde av senaste kommandot

~/ .bashrc

- Skript som körs vid varje ny terminal
- Bra för att konfigurera!
- (Kom ihåg: ~ är din hemkatalog)

“command substitution”

- Ibland vill man i ett kommando ha resultatet av ett annat kommando
- Två operatörer för att åstadkomma detta
 - `$(date)` `<--` Använd denna
 - ``date``
- Vanlig användning att spara output i variabel:
 - `filer=$(ls)`