SKAL-PROGRAMMERING

OLIKA SKAL

- csh-kompatibla
  - csh - C shell
  - tcsh
- Bourne-kompatibla
  - sh - Bourne shell.
  - Används ofta för script
  - bash - Bourne again shell. Standardskalet på institutionen (?)
  - ksh - Korn shell
  - pdksh - public domain-variant av ksh
  - zsh

PIPOR OCH OMDIRIGERING AV STRÖMMAR

- Med en pipa kan man låta ett kommando läsa från utdatat av ett annat
  - Åstadkoms genom att man skriver ett `|` mellan kommandona
  - Ex: `ls | wc`
- Stdin, stdout och stderr kan också omdirigeras till filer
  - I tcsh:
    - stdin `<`
    - stdout `>`
    - stderr och stdout `>&`
  - Ex: `cat *.c > fil.txt`

SKALPROGRAMMERING

- Shell-script i stället för ett C-program
  - Snabbare/enklare
    - Ingen kompilering
    - Kan använda redan färdiga program/kommandon
  - Finns oftast redan program som gör nästan det man vill göra
  - Kan se shell-scripts som ett snabbt sätt att utöka plattformens funktionalitet (utöka "systemprogramvaran")

SKALPROGRAMMERING

- I princip alla skal kan användas för att skriva shell-scripts (även om inte alla är lika lämpliga för detta), ett vanligt skal är bash (eller sh).
- Ett shell-script är en textfil som måste börja med "`#!`" följt av sökvägen till det skalprogram som ska köras, t.ex.

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

  - På den raden kan man också lägga in olika options till det skalprogram man använder.
- Om man ändrar textfilens rättigheter så kan man köra den som ett vanligt commando.

```
$ chmod +x scriptfil
$ ./scriptfil
```

SKALPROGRAMMERING – EXEKVERING

- Skapar ett nytt shell
  - Innebär t ex att alla variabler mm man skapar i ett script är lokala för scriptet

```
$ pwd
/Home/staff/mr
$ ./mycd
/etc
$ pwd
/Home/staff/mr
```

mycd

```
#!/bin/bash
cd /etc
pwd
```

BASH

- Kommentarer
  - `#` pågår till slutet av raden
- Exit-status
  - Alla program (och script) bör explicit returnera en exit-status
    - C-program via `exit`-funktionen (eller `return` från `main`)
    - Shell-scripts via `exit`
    - Används ofta i script för att få veta om ett program/kommando lyckades eller inte (`0` om allt OK, `≠0` om fel)

BASH – VARIABLER

- Man kan definiera variabler i shell-scripts

```
demoVariable=”Hi there”
echo $demoVariable
```

- Bara en datatyp, "strängar”
- Man kan också kolla om variabler och definierade genom att använda följande konstruktioner när man vill använda värdet

BASH – VARIABLER

- `${variabel:-värde}`
  - Om variabeln existerar och har ett värde så används detta värde i annat fall så används värdet 'värde'
- `${variabel:=värde}`
  - Om variabeln inte existerar eller inte har nåt värde skapas variabeln och sätts till värdet 'värde'
- `${variabel:+värde}`
  - Om variabeln existerar och har ett värde så används 'värde' annars blir det en tom sträng.
- `${variabel:?värde}`
  - Om variabeln existerar och har ett värde så använd det. I annat fall så skrivs 'värde' ut och avslutas.

BASH – CITATIONSTECKEN

- Dubbelsnutt (")
  - Om man börjar/slutar en sträng med " så kommer det att ske en variabelsubstitution när man evaluerar uttrycket
    `echo "$var1 $var2"`
- Enkelsnutt (')
  - Ingen variabelsubstitution
    `echo '$var1 $var2'`
- Bakåtsnutt (`)
  - Det som skrivs ut av ett kommando
    `example=`ls``

BASH – SUBSTITUTION EXEMPEL

```
var=ls
```

`echo $var`	Substitution: `ls`
`echo "$var"`	Substitution: `ls`
`echo '$var'`	Ingen substitution: `$var`
`echo `ls``	Substitution: `a.out fil.c text.doc`
`echo `$var``	Substitution: `a.out fil.c text.doc`
`echo '"$var"'`	Ingen substitution: `"$var"`
`echo "'$var'"`	Substitution: `'ls'`

BASH – KOMMANDORADSARGUMENT

- `$0` scriptnamnet
- `$1` första argumentet, `$2` andra osv.
- `$#` antalet argument (scriptnamnet ej inkluderat)
- `$*` alla argument
  - `"$*"` tolkas som `"$1 $2 $3 …"`
- `$@` alla argument
  - `"$@"` tolkas som `"$1" "$2" "$3" …`
- `shift`
  - Flytta alla argument "ett steg till vänster" (ej `$0`), dvs `$1` får det värde `$2` hade, `$2` det `$3` hade osv.
  - `$1` försvinner och `$#` minskas med `1`

BASH – KOMMANDORADSARGUMENT

- Exempel:
- ```
  #!/bin/bash
  ls -l $* | grep "^.......rwx"
  cat $0 | grep ord | wc
  shift
  cat $0 | grep ord | wc
  ```

BASH – KONTROLLSTRUKTURER

- Det finns även ett antal olika kontrollstrukturer som man kan använda (annars vore det tämligen oanvändbart).
- **Sant/falskt**
  - Tvärtom från C, 0 är sant och icke-0 är falskt

BASH – IF

- `if list; then list; [elif list; then list;] … [else list;]`

BASH – IF EXEMPEL

```
if ls /finnsinte; then
  echo "... där var innehållet i /finnsinte"
else
  echo "Fel vid listning av /finnsinte"
fi
```

- En annan variant

```
if ls /hubba; then
  echo "... och där var innehållet i /hubba"
elif ls /bubba; then
  echo "...och där var innehållet i /bubba"
else
  echo "Fel vid listning av /hubba och /bubba"
fi
```
```

BASH – TEST

- Man kan använda kommandot `test` för att jämföra saker (OBS om man döper ett program till test och sedan försöker köra det genom att skriva `test` så är det det inbyggda `test` som körs)

```
if test "$a" = "$b"; then
    bla bla bla bla
```

- annat sätt att skriva det på

```
if [ "$a" = "$b" ]; then
    bla bla bla bla
```

  - OBS blanktecknen

- Kan också användas till att kolla filer (om de existerar är läsbara osv

BASH – TEST

- `test -w file` (har filen skrivrättigheter?)
  `-r` (läsrättigheter?)
  `-x` (exekveringsrättigheter?)
  `-s` (är filen icketom?)
  …
- `test nr1 -eq nr2` (är tal1 lika med tal2?)
  `-ne` (är tal1 skilt ifrån tal2?)
  `-gt` (är tal1 större än tal2?)
  …
- `test str1 = str2` (är sträng1 lika med sträng2?)
  `!=` (är sträng1 skilt ifrån sträng2?)

BASH

- `$?` Innehåller returvärdet från det senaste kommandot

```
anExampleCommand
myResult="$?"
if [ "$myResult" -eq 0 ] ; then
    bla bla bla bla
```

BASH – CASE

- Det finns en switchliknande sats också

```
maskin=`uname -n`
case $maskin in
    peppar)       echo "server" ;;
    jumjum)       echo "arbetsstation" ;;
    garnet)       echo "SGI-arbetsstation" ;;
    c[0-9] | d*) echo "labmaskiner" ;;
    *)            echo "Okänd"
                  exit
                  ;;
esac
```

BASH – FOR

- Man kan också loopa över innehållet i en lista

```
for n in This is a list
do
    echo $n
done
```

- Man kan använda "`break`" för att hoppa ur en for-sats

BASH – WHILE

- **`while`** `list-1;` **`do`** `list-2;` **`done`**

-
```
while [ ”$a” != ”$b” ]
do
    b=`kommando`
done
```

BASH – UNTIL

- **`until`** `list-1;` **`do`** `list-2;` **`done`**

-
```
until who | grep ”^username”
do
    sleep 60
done
echo ”Nu har User loggat in”
```

BASH – || OCH &&

- Används för att göra något om nåt annat lyckades/misslyckades. En sorts mini-if
  - `cat fil1 fil2 > fil3 || exit`
  -
```
cat fil1 fil2 > fil3 || {
    echo ”Hopslagningen funkade inte”
    exit
}
```
  - `lpr -Pprinternamn ”$fil” && rm ”$fil”`
- Om man använder `||` så betyder det att man gör det som kommer efter om det som var före returnerar ett värde skiljt från 0. `&&` gör tvärtom.

BASH – LÄSA IN EN RAD

- Man kan använda "`read`" för att läsa en rad, läser till radslutet.

```
read filnamn
echo $filnamn
```

BASH – EXEMPEL 1

```
#!/bin/sh
if [ $# -lt 1 ]; then
   echo 1>&2 ”Usage: $0 [-l] file [file...]”
   exit 1
fi
if [ $1 = ”-l” ]; then
   command=”ls -l”
   shift
else
   command=”ls”
fi
while [ $# -ge 1 ]; do
   $command $1
   shift
done
exit 0;
```

BASH – EXEMPEL 1: TESTKÖRNING

```
boule[48] ~ % ./myls
Usage: ./myls [-l] file [file...]
boule[49] ~ % ./myls fil
ls: cannot access fil: No such file or directory
boule[50] ~ % ./myls a.out tt.c myls
a.out
tt.c
myls
boule[51] ~ % ./myls -l a.out tt.c myls
-rwx------ 1 mr tdb 6456 2010-03-02 13:31 a.out
-rw------- 1 mr tdb  152 2010-03-02 13:36 tt.c
-rwx------ 1 mr tdb  241 2010-04-09 10:50 myls
boule[52] ~ %
```

BASH – EXEMPEL 1: TESTKÖRNING

- Ändra till: `#!/bin/sh -x`

```
boule[53] ~ % ./myls -l a.out tt.c myls
+ '[' 4 -lt 1 ']'
+ '[' -l = -l ']'
+ command='ls -l'
+ shift
+ '[' 3 -ge 1 ']'
+ ls -l a.out
-rwx------ 1 mr tdb 6456 2010-03-02 13:31 a.out
+ shift
+ '[' 2 -ge 1 ']'
+ ls -l tt.c
-rw------- 1 mr tdb 152 2010-03-02 13:36 tt.c
...
```

BASH – EXEMPEL 2

```
#!/bin/bash
if [ $# -lt 2 ]; then
  echo 1>&2 ”Usage: $0 försättsblad.pdf tenta.pdf”
  exit 1
fi
tot_pages=`pdfinfo $1 | grep Pages | cut -d: -f2`
nr_pages=2
fblad=”/tmp/$(basename $1)_$$.ps”
tenta=”/tmp/$(basename $2)_$$.ps”
exemplar=”/tmp/tmp_$$.ps”
if ! (pdf2ps $1 $fblad); then
  echo 1>&2 ”Error using pdf2ps on $1”
  exit 1;
fi
if ! (pdf2ps $2 $tenta); then
  rm -rf $fblad
  echo 1>&2 ”Error using pdf2ps on $2”
  exit 1;
fi
# Skriv ut tentavaktsbladet
psselect -p1-1 $fblad | lpr –Pmypr
```

BASH – EXEMPEL 2 (FORTS)

```
# Skriv ut försättsblad ihophäftat med tentan
# (en utskrift per kodnummer)
i=2
c=0
while [ $i -lt $tot_pages ]; do
  let c=$c+1
  let j=$i+$nr_pages-1
  psselect -p$i-$j $fblad > $exemplar
  psmerge $exemplar $tenta | lpr -Pmypr -o Duplex=Simplex -o Stapling=UL
  let i=$i+2
  rm -rf $exemplar
  if [ $c -eq 10 ]; then
    echo ”Press ENTER to continue”
    read
    let c=0
  fi
done
rm -rf $fblad $tenta $exemplar
```

BASH – EXEMPEL 3

```
#!/bin/bash
if [ $# -gt 1 ]; then
    echo 1>&2 ”Usage: ”$0” [file]”
    exit 1;
fi
if [ $# -eq 1 ]; then
    file=$1;
else
    file=-;
fi
#Wrong order in the output
#cat $file | cut -d: -f1,3 | sort -n -t: -k2
#Correct order using sed
cat $file | cut -d: -f1,3 | sed 'p' | paste -d: - - | cut -d: -f2,3
| sort -n
#Correct order using awk
#cat $file | cut -d: -f1,3 | sort -n -t: -k2 | awk -F: '{printf ”%s:
%s\n”, $2, $1}'
```

EXEMPEL 3 – ANNAT SCRIPTSPRÅK

```
#!/usr/local/bin/perl

@document = <>;
sub mysort { (split /:/, $a)[2] <=> (split /:/, $b)[2] }

@sorteddoc = sort mysort @document;
foreach $ent (@sorteddoc) {
   @tmp = split /:/, $ent;
   print $tmp[2], ”:”, $tmp[0], ”\n”;
}
```