BASH

Bourne-Again SHell

Eric Elfving

Institutionen för datavetenskap (IDA)

BASH

- sh-compatible command language interpreter
- Kan startas på två sätt:
  - *Login shell*
    - Kör init-filer i följande ordning:
      - /etc/profile, ~/.bash_profile, ~/.bash_login, ~/.profile
    - Kör filen ”~/.bash_logout” när bash avslutas
  - *Interactive*
    - Kör filen ”~/.bashrc” vid uppstart (kan skippas med flaggan **–norc**)
    - Inläsning och utskrift sker mot en terminal

SYNTAX (OCH SEMANTIK)

Syntax i BASH

- Ett kommando består av en lista av en sekvens av en eller flera ”pipelines”
- Varje pipeline separeras av en av operatorerna **;**, **&**, **&&** eller **||** och kan avslutas av **;**, **&** eller **<newline>**
- Om en pipeline avslutas med **&** körs det i bakgrunden i ett subshell, med **;** väntar bash istället tills körningen är klar.
- **&&** och **||** betyder logiskt och respektive logiskt eller.
- En pipeline skrivs som

  kommando1 [| *kommando2* …]

  där utskriften från *kommando1* skickas som inmatning till *kommando2*.

Sammansatta uttryck

- (*list)* *list* körs i ett eget subshell
- { *list; }* Gruppering, *list* körs i nuvarande miljö. { och } måste separeras från *list* med whitespace
- ((*uttryck*)) *uttryck* beräknas, vid ett nollskilt värde ges resultatet 1, annars 0
- [[*villkor*]] Strängar kan jämföras med operatorerna **==**, **!=**, **<** och **>**. Den högra strängen kan vara ett mönster. Med operatorn **~=** tolkas den högra operanden som ett reguljärt uttryck

Villkor

- Det finns flera test att göra på filer:
  - -a *file* — Sann om *file* finns
  - -d *file* — Sann om *file* finns och är en mapp
  - -f *file* — Sann om *file* finns och är en vanlig fil
  - -r, -w, -x *file* — Sann om *file* finns och är läs-, skriv- respektive körbar
- Andra bra tester:
  - -n *string* — Sann om *string* har längd > 0
  - *Val1* **OP** *val2* — **OP** kan vara en av **–eq, -ne, -lt, -le, -gt, -ge.** Ger sant om *val1* är lika, inte lika, mindre än, mindre eller lika med, större respektive större eller lika med *val2* för heltalsvärden.

Upprepning

- **for** *name* [**in** *word*] ; **do** *list* ; **done**

  Fungerar likt for-loopen i Python, intererar över orden i *word* *name* tar nuvarande värde och *list* utförs för varje värde.

- ***for*** ((*uttryck1* ; *uttryck2* ; *uttryck3*)) ; **do** *list* ; **done**

  Inledningsvis beräknas *uttryck1*. *uttryck2* beräknas och om det är nollskilt utförs *list* och därefter beräknas *uttryck3.* Om *uttryck2* blir noll avbryts upprepningen

- **while** *list1*; **do** *list2*; **done**

  Upprepa *list2* tills den sista pipen i *list1* returnerar 0

Selektion

- **if** *list*; **then** *list*; [**elif** *list*; **then** *list*;] … [**else** *list*;] **fi**

```
if [[ -f ~/my_file ]]
then
echo 'my_file finns!'
else
echo 'my_file finns inte!'
fi
```

```
if [[ 4 -eq 5 ]]
then
echo "Något är väldigt fel"
fi
```

```
for file  in $(ls)
do
echo $file
done
```

Variabler

- Skapas likt python: *var* = *värde*
- Åtkomst med $: $var eller ${var}
- Det finns fler inbyggda variabler, t.ex.
  - $HOME — Path till hemkatalogen
  - $EDITOR — Default-editor för systemet (sätt till emacs…)
  - $LANG — Systemets språk (locale)
  - $PATH — En lista på mappar som används vid sökning av program
  - $PS1 — Bestämmer hur promten ser ut (senare)
  - $PWD — Nuvarande mapp
  - $RANDOM — Ger ett slumptal
  - $TMPDIR — Systemets tempkatalog (ofta /tmp)

Fält (array)

- BASH har endimensionella fält. Skapas enligt

  var[*index*] = *värde*

  - *Index* ska vara ett heltalsvärde (0 och uppåt)
  - Tilldelning kan också ske enligt *var*=(*värde1, värde2, … , värdeN*)
  - Åtkomst sker med ${*var*[*index*]}
  - För att komma åt alla värden som en lista kan man använda * eller @ som index

Utvidgning (expansion)

- Inmatade ord utvidgas på sju olika sätt;
  - *Brace expansion*

    Ett sätt att generera strängar.

    a{b,d,f}e => ’abe ade afe’

    a{b..d}e => ’abe ace ade’

    Kan nästlas

```
cp {orginal,kopia}.py

cp min_fil.{py,py_bak}
```

```
echo ~ => /home/eriel
echo ~torjo => /home/torjo
```

  - *Tilde expansion*

    Om ett ord inleds med tilde (~) anses alla tecken fram till första / vara ett *tilde-prefix*. Om prefixet är noll tecken långt byts tilde-tecknet ut mot **$HOME** annars byts det ut mot hemkatalogen för användaren som motsvarar prefixet. Om det inte finns någon användare med det namnet lämnas ordet oförändrat

Utvidgning (expansion)

- *Parameter expansion*

  $*parameter* byts ut mot dess värde. Kan skrivas ${*parameter*} för att göra det tydligare

  Det finns flera varianter, t.ex;

  - ${*par:-word}*

    Använd *word* om *par* inte är satt

  - ${*par:offset*[:*length*]}

    Ta fram en delsträng av $par från tecken *offset* till slutet av strängen, dock max *length* tecken.

- *Command substitution*

  $(*kommando*) eller ´*kommando*´ kör *kommando* och byts ut mot resultatet av körningen.

```
var=”Eric”
echo $vars       =>
echo ${var}s     => Erics
echo ${var:2}    => ic
echo ${var:0:2} => Er
```

```
for file  in $(ls)
do
echo $file
done
```

Utvidgning (expansion)

- *Arithmetic expansion*

  $((*uttryck*)) beräknar *uttryck* och byts ut mot resultatet av beräkningen. Går att använda samma artimetiska och logiska operatorer som i C.

- *Process substitution*

  <(*list*) och >(*list*) skapar temporära filer med utskrift respektive indata till kommandot *list*

- *Word splitting*

  De ord som finns kvar efter ovanstående steg delas vid förekomst av vita tecken

- *Pathname expansion*

  Varje ord som innehåller tecknen *****,**?** och **[** anses vara ett mönster och byts ut mot en alfabetiskt sorterad lista av ord som matchar

```
var=5
echo $(($var+6)) => 11
echo $((var++))  => 5
echo ${var}      => 6
echo $((++var))  => 7
```

```
echo <(echo "hej")
/dev/fd/63
cat <(echo "hej") <(echo "du")
hej
du
```

Mönstermatchning

- Om ett mönster hittats matchas det enligt följande regler
  - ***** => Noll eller flera tecken
  - **?** => Ett tecken
  - **[…]** => Ett av de tecken som innesluts av [och]
  - Sammansatta mönster
    - Flera mönster kan slås samman med **|**
    - *(*pattern*) Noll eller flera förekomster av *pattern*
    - ?(*pattern*) Noll eller en förekomst av *pattern*
    - +(*pattern*) En eller flera förekomster av *pattern*
    - @(*pattern-list*) Endast ett av de givna mönstren
    - !(*pattern*) Allt utom *pattern*
  - Övriga tecken matchas endast mot sig själv

Omdirigering

- Istället för att läsa och skriva från terminalen kan det ske med hjälp av filer. Med operatorn **>** skickas utskriften till en fil och med **<** läses en fil istället för användarinmatning

```
ls > files.txt
my_program < indata
```

- Man kan även använda omdirigering för att få felmeddelanden (stderr) på fil

```
my_program 2> errors > output
```

- > skriver över gammalt innehåll, **>>** lägger till i slutet av filen

```
my_program >> output.log
```

Alias

- Med hjälp av alias kan man få BASH att byta ut ett ord mot ett annat ord

```
alias ls='ls -l'
unalias ls
```

- Bra att ha för jobbiga kommandon men oftast bättre med en funktion

Funktioner

- **[function]** *name*() *sammansatt-uttryck*

  Definierar funktionen *name.* Om man vill kan antingen det reserverade ordet function eller parenteserna utelämnas
- Funktioner anropas som vanliga kommandon
- Variabler delas helt mellan funktioner och anroparen. Detta kan ändras genom användning av **local**
- Endast positionella parametrar,
  - åtkomst via $*parameternummer* med start från 1.
  - I $# lagras antal parametrar.
  - Alla parametrar kan kommas åt med ”$@”

```
function fun() { a=4; }
fun
echo $a

b=6
function fun2()
{
local b
b=5
}
fun2
echo $b
```

Funktioner

```
function fun()
{
for p in "$@"
do
echo $p
done
}
```

```
fun hej du
hej
du

fun 'hej du'
hej du
```

Promten

- Bash kan visa användaren lite information om den körs interaktivt
- Om bash är redo att ta emot kommandon visas $PS1. $PS2 visas när mer inmatning krävs
- Variablerna kan innehålla ett flertal specialtecken, tex.
  - \h Datornamn
  - \w Namnet på nuvarande mapp (full path)
  - \u Användarnamn
  - \! Kommandots historiknummer

Job

- Med *job control* menas möjligheten att stoppa en process för att senare återuppta det.
- Bash kopplar ett job med varje pipeline
- Med hjälp av Ctrl-Z sätts en process i vila och bash fortsätter. Ctrl-Y sätter processen i vila vid nästa inläsningsoperation
- Bra kommandon i samband med job control:
  - **jobs** Listar aktuella jobs (åtkomst till job med %n)
  - **bg** [*%n*] Kör jobb nummer n i bakgrunden (egen process)
  - **fg** [*%n*] Återta job n i denna process (”låser” bash)
  - **kill** [*pid*] [*jobid*] Stäng ner process *pid* eller job *jobid*

Historik

- Som standard sparas alla kommandon till filen $HISTFILE (~/.bash_history) vid avslutad session
- För att återupprepa ett tidigare kommando kan man använda !;
  - !*n* Kör kommando nummer *n*
  - !-*n* Kör det n:e föregående kommandot
  - !! Kör föregående kommando (samma som !-1)
  - !*string* Kör det senast förekommande kommandot som börjar med *string*
  - !?*string*[?] Kör det senast förekommande kommandot som innehåller *string*

Inbyggda kommandon

- **source** *filnamn*

  Kör varje rad i filen *filnamn*
- **cd, pwd, mkdir …** (se stone!)
- **echo** Skriv ut värden
- **export** [*name*]

  Se till att variabeln *name* finns tillgänglig utanför nuvarande miljö
- **fc** [*first*] [*last*]

  Kopiera kommando [*first, last*] från historiken till din editor. När du avslutar körs de kommandona
- **pushd** [*dir*]

  Gå till mapp *dir* men spara nuvarande position, **popd** går tillbaka

Linköpings universitet

expanding reality

www.liu.se