Administration of Operating Systems

DO2003

http://www.hh.se/do2003

Programming the Bourne Again Shell

A wife asks her husband, a computer programmer, "Would you please go to the store for me and buy a bottle of milk? And if they have eggs, get six."

A short time later the husband comes back with 6 bottles of milk.

The wife asks him, "Why the hell did you buy 6 bottles of milk?"

He replied, "They had eggs."

Bourne Again Shell (bash)

- Executes commands from the command line and from shell scripts files
- A shell script is a text file containing
  - Shell commands, control flow structures, parameters, variables, functions, …
- Running scripts
  1. `$ Filename [arguments]`
  2. `$ ./Filename [arguments]`
  3. `$ bash Filename [arguments]`

HÖGSKOLAN HALMSTAD

Very simple script

```
$ PATH=$PATH:/home/ide
$ chmod 700 *.sh
$ cat > srm
#!/bin/bash
# Safe Remove
backup_path=/tmp
cp $1 $backup_path
echo $1 is stored in $backup_path
rm $1
$ chmod u+x srm
$ srm report.txt
report.txt is stored in /tmp
$
```

working directory in the PATH

rwx permission for all scripts

HÖGSKOLAN HALMSTAD

Parameters and variables

- A parameter stores values and a variable is a parameter denoted by a name
- Users can define *User-created variables*

```
VARIABLE=value
```

- The shell has *keyword variables*
- Array variables

```
VARIABLE=(element1 element2 …)
```

```
$ STUDENTS=(Wagner Slawomir Mattias)
$ echo ${STUDENTS[1]}
Slawomir
$ echo ${#STUDENTS[*]}
3
$ echo ${#STUDENTS[1]}
8
```

Special parameters

- $? Exit status or exit code
  - 0 is success (true)
- $# Number of Command-Line Arguments
- $0 Name of the Calling Program
- $1–$n Command-Line Arguments
- shift Promotes Command-Line Arguments
- set Initializes Command-Line Arguments
- $* and $@ Represent All Command-Line Arguments

Special characters

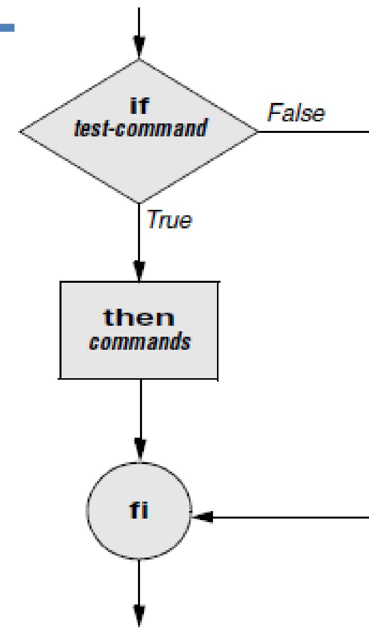
- #   comment; terminated by new line
- | connects two commands
- ;   command separator
- &   run process in background
- &&   only run following command if previous command completed successfully
- || only run following command if previous command failed
- ´   enclose string to be taken literally
- "   enclose string to have variable, command and arithmetic substitution only
- $( )   in-line command substitution (new style)
- ‘   in-line command substitution (old style)
- ((...)) arithmetic evaluation, like let "..."
- $((...)) in-line arithmetic evaluation
- \   treat following character literally
- \newline line continuation

HÖGSKOLAN HALMSTAD

Conditional construct: if...then

```
$ cat compIN1.sh
#!/bin/bash
if test $1 = $2
then
   echo "Inputs are equal"
   exit 0
fi
echo "Inputs are different"
$ compIN1.sh 1 2
```

- test -d file
- test -e file
- test -f file
- test -w file
- test -x file
- test -s file



- [ ] is a synonym for test

```
$ cat compIN2.sh
#!/bin/bash
if [ $1 == $2 ]
then
   echo "Inputs are equal"
   exit 0
fi
echo "Inputs are different"
$ compIN2.sh 1 2
```

Conditional construct: `if...then`

- $# special parameter

```
$ cat compIN3.sh
#!/bin/bash
if [ $# -ne 2 ]
then
  echo "You must supply two arguments"
  exit 1
fi
if test $1 = $2
then
   echo "Inputs are equal“
   exit 0
fi
echo "Inputs are different"
$ compIN3.sh wagner slawomir
```

Integer or binary comparison

- -eq
- -ne
- -gt
- -ge
- -lt
- -le

Integer and string comparison

- == or =
- !=
- >
- <

Boolean comparison

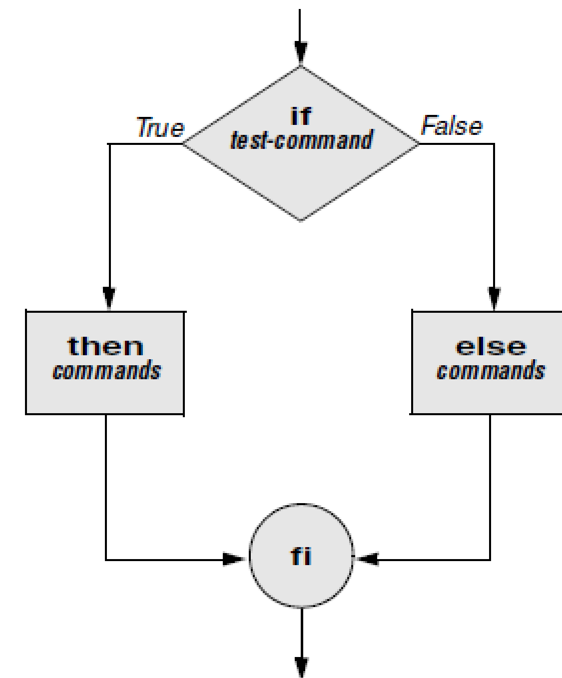
- -a
- -o

HÖGSKOLAN HALMSTAD

Conditional construct: `if...then…else`

- ; ends a command

```
$ cat compIN4.sh
#!/bin/bash
if [ $# -ne 2 ] ; then
  echo "Usage: comIN4.sh arg1 arg2"
  exit 1
fi
if test $1 = $2
then
   echo "Inputs are equal"
else
   echo "Inputs are different"
fi
$ compIN4 1 2
```

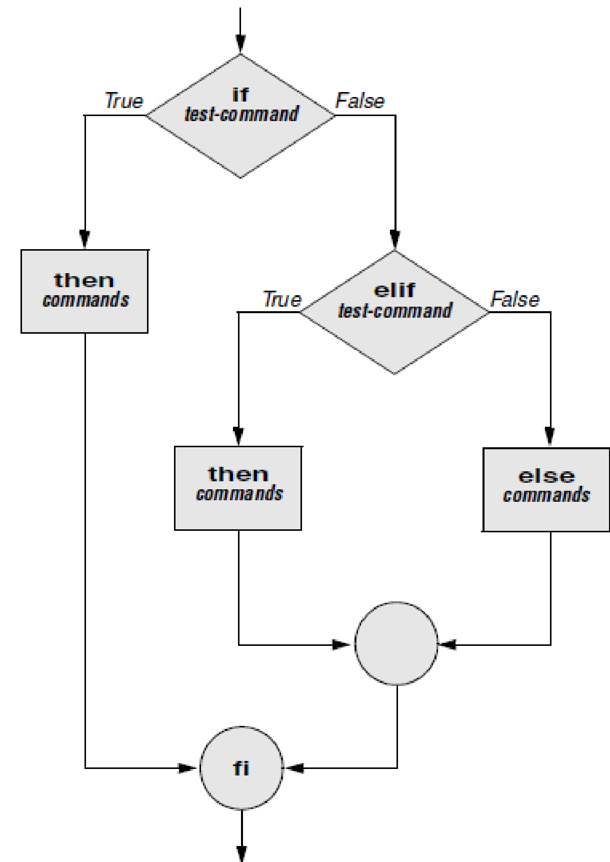


HÖGSKOLAN HALMSTAD

Conditional construct: `if...then…elif`

- Nested set of if…then…else

```
$ cat if3
#!/bin/bash
echo -n "word 1: "
read word1
echo -n "word 2: "
read word2
echo -n "word 3: "
read word3
if [ "$word1" = "$word2" -a "$word2" = "$word3" ]
   then
      echo "Match: words 1, 2, & 3"
   elif [ "$word1" = "$word2" ]
   then
      echo "Match: words 1 & 2"
   elif [ "$word1" = "$word3" ]
   then
      echo "Match: words 1 & 3"
   elif [ "$word2" = "$word3" ]
   then
      echo "Match: words 2 & 3"
   else
      echo "No match"
fi
```

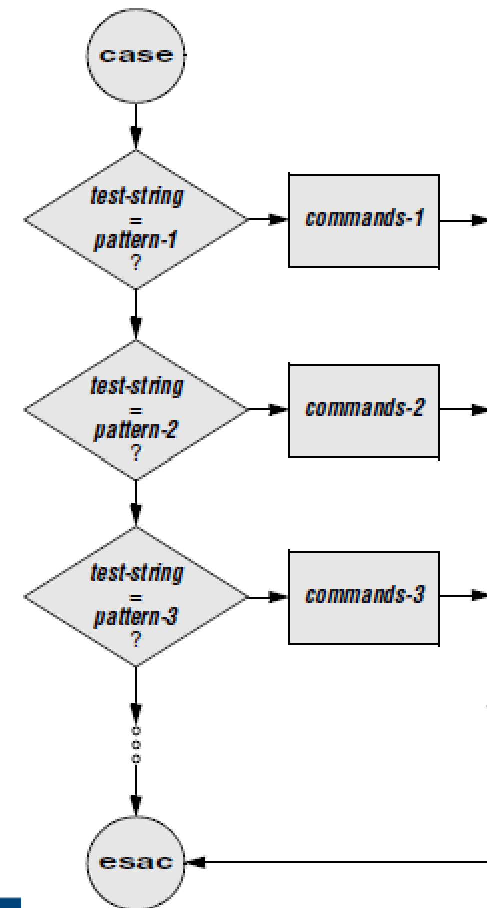


HÖGSKOLAN HALMSTAD

Conditional construct: `case`

- A nested set of `if…then…else` can be confusing
- The case control structure provides a multiway branch

```
$ cat signal.sh
#! /bin/bash
if [ $# -lt 2 ]
then
   echo "Usage : $0 Signalnumber PID"; exit
fi
case "$1" in
1)  echo "Sending SIGHUP signal"; kill -SIGHUP $2
   ;;
2)  echo  "Sending SIGINT signal"; kill -SIGINT $2
   ;;
3)  echo  "Sending SIGQUIT signal" ; kill -SIGQUIT $2
   ;;
9) echo  "Sending SIGKILL signal"; kill -SIGKILL $2
   ;;
*) echo "Signal number $1 is not processed"
   ;;
esac
```



Conditional construct: case

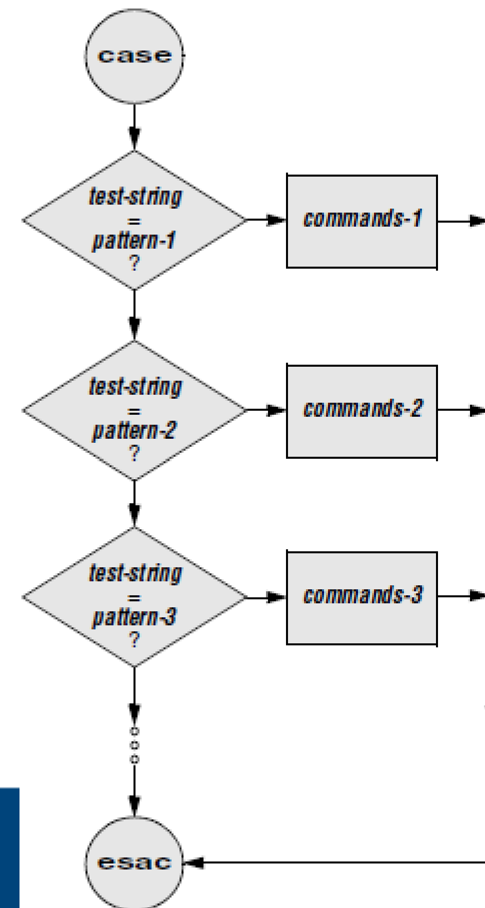
```
#!/bin/bash
extract () {
    if [ -f $1 ] ; then
        case $1 in
            *.tar.bz2)   tar xvjf $1        ;;
            *.tar.gz)    tar xvzf $1     ;;
            *.bz2)       bunzip2 $1       ;;
            *.rar)       unrar x $1     ;;
            *.gz)        gunzip $1     ;;
            *.tar)       tar xvf $1        ;;
            *.tbz2)      tar xvjf $1      ;;
            *.tgz)       tar xvzf $1       ;;
            *.zip)       unzip $1     ;;
            *.Z)         uncompress $1  ;;
            *.7z)        7z x $1    ;;
            *)           echo "'$1' cannot be extracted via >extract<" ;;
        esac
    else
        echo "'$1' is not a valid file"
    fi
}
extract "$@“
```

http://www.shell-fu.org/lister.php?tag=bash

Conditional construct: case

- A nested set of `if…then…else` can be confusing
- The case control structure provides a multiway branch
  - The path depends on a match or lack of a match between the test-string and one of the patterns

```
$ cat caseex.sh
echo -n "Enter A, B, or C: “; read letter
case "$letter" in
   a|A)
      echo "You entered A"
      ;;
   b|B)
      echo "You entered B"
      ;;
   c|C)
      echo "You entered C"
      ;;
   *)
      echo "You did not enter A, B, or C"
      ;;
esac
```

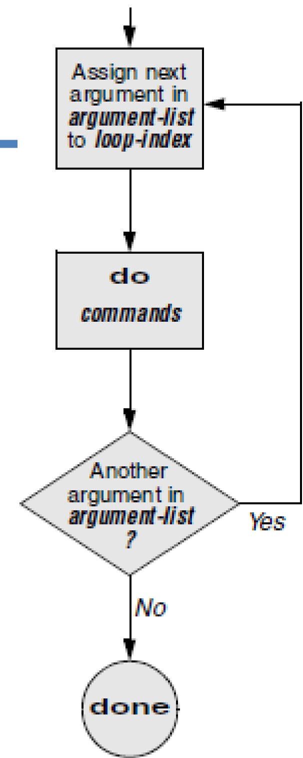


Looping Constructs: **for…in**

- Perform a loop over the argument list

```
$ cat shtolower.sh
#!/bin/bash
for i in *.sh
do
   echo mv \"$i\" \"`echo $i| tr [A-Z] [a-z]`\"
done
$ shtolower
mv "compIN1.sh" "compin1.sh"
...
$
```

```
$ cat dirdir.sh
#!/bin/bash
for i in *; do if [ -d "$i" ]; then echo "$i"; fi; done
$
```



Looping Constructs: for

- Perform a loop over the argument list

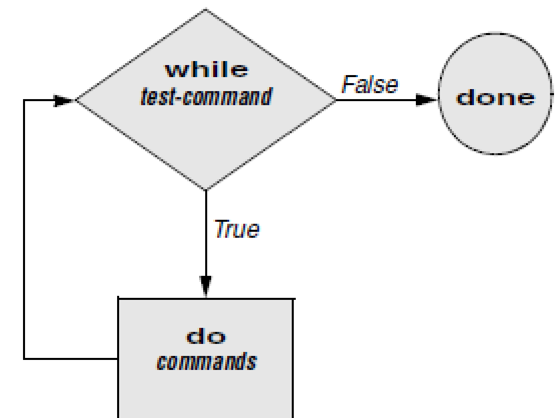
```
$ cat count.sh
#!/bin/bash
for (( i=1; i<=5; i++ ))
do
   echo "Counting $i times..."
done
$
```

```
$ cat count2.sh
#!/bin/bash
for i in 1 2 3 4 5
do
   echo "Counting $i times..."
done
$
```

Looping Constructs: `while`

- Perform a loop as long as the test-command returns a true exit status

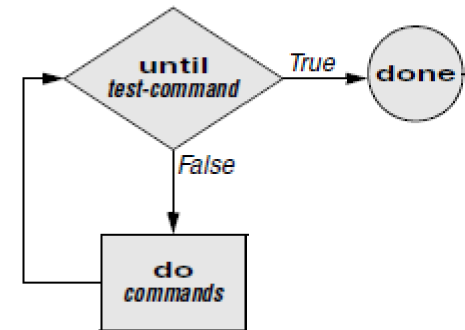
```
$ cat count3.sh
#!/bin/bash
number=0
while [ "$number" -lt 10 ]
   do
      echo -n "$number"
      ((number +=1))
   done
echo
$ count3.sh
0123456789
$
```



Looping Constructs: `until`

- Perform a loop as long as the test-command returns a false exit status

```
$ cat guesswho.sh
#!/bin/bash
secretname=wagner
name=noname
echo "Try to guess the secret name!"
echo
until [ "$name" = "$secretname" ]
do
   echo -n "Your guess: "
   read name
done
echo "Very good."
$
```



HÖGSKOLAN HALMSTAD

Break and continue control structures

- Both alter control within loops
  - break transfers control out of a loop
    - exit the current loop before its normal ending
  - continue transfers control immediately to the top of a loop

```
$ cat brkcont.sh
#!/bin/bash
for index in 1 2 3 4 5 6 7 8 9 10
do
   if [ $index -le 3 ] ; then
      echo "continue"
      continue
   fi
   echo $index
   if [ $index -ge 8 ] ; then
      echo "break"
      break
   fi
done
$
```

File descriptors

- Known file descriptors
  - stdin (0), stdout (1) and stderr (2)
- When a process opens a file, Linux associates a number, the file descriptor, with the file

```
exec n> outfile
exec m< infile
```

- Read and write operations use the file descriptor
- At the, close the file descriptor

Operator	Meaning
`<&m`	Duplicates stdin from file descriptor m
`[n]>&m`	Duplicates stdout or file descriptor n if specified from file descriptor m
`[n]<&-`	Closes standard input or file descriptor n if specified
`[n]>&-`	Closes standard output or file descriptor n if specified

File descriptors

```
$ cat FileDesc.sh
#!/bin/bash
usage ()
{
if [ $# -ne 2 ]; then
   echo "Usage: $0 file1 file2" 2>&1
   exit 1
fi
}
usage "$@“
exec 3<$1
exec 4<$2
read line1 <&3
echo "$line1“
read line2 <&4
echo "$line2“
read -u3 lineword1 lineword2 lineremaining
echo "$lineword1, $lineword2, $lineremaining“
exec 3<$- 4<$-
$
```

Debugging shell scripts

- Programming mistakes are common
- The –x option helps to debug a script, i.e, traces a script's execution
- The shell displays each command before it runs the command

```
$ bash -x compIN1.sh wagner wagner
+ test wagner = wagner
+ echo 'Inputs are equal'
+ exit 0
$
```

Builtins

- `type` Displays how each argument would be interpreted as a command
- `read` Reads a line from standard input
- `exec` Executes a shell script or program in place of the current process
- `trap` Traps a signal
- `kill` Sends a signal to a process or job
- `getopts` Parses arguments to a shell script
- `let` allows arithmetic to be performed on shell variables

Help with BASH programming

- http://tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf
- http://linuxconfig.org/Bash_scripting_Tutorial
- http://www.gnu.org/software/bash/manual/
- http://www.digilife.be/quickreferences/QRC/Bash%20Quick%20Reference.pdf

HÖGSKOLAN HALMSTAD