Exit code.

Command test.

Flow Control.

Loops.

Department of Computer Systems FIT, Czech Technical University in Prague

Exit code

- Every process returns exit code during termination.
- **Exit code = integer 0, 1, ... ,255**

  0 success,

  1,...,255 error.

- **Exit code of the last foreground command** is saved in variable **?**

- **Shell script can be terminated with exit code n** by command

```
exit [n]
```

- **Shell function can be terminated with exit code n** by command

```
return [n]
```

Exit code - examples

```
$ grep 'root' /etc/passwd
root:x:0:1:Super-User:/root:/sbin/sh
$ echo $?
0
$ grep 'XXX' /etc/passwd
$ echo $?
1
$ grep 'root' /XXX
grep: can't open /XXX
$ echo $?
2
$ XXXgrep 'root' /etc/passwd
-bash: XXXgrep: command not found
$ echo $?
127
```

Command test

- Test exits with the exit status determined by expression.
- Arguments are analyzed by shell, therefore
  - they must be separated by spaces,
  - use symbol '`\`' before special character.

- **Compound expressions**

`\( A \)`	Define priority of the expression `A`.
`A –a B`	Logical conjunction: the expression is true if both `A` and `B` are true.
`A –o B`	Logical disjunction: the expression is true if either `A`, `B`, or both, are true.
`! A`	Logical negation: the expression is true if `A` is false and false if `A` is true.

Command test and its synonyms

`test expression`

- The test utility evaluates the condition and indicates the result of the evaluation by its exit status.
- The command `test` is built-in command in ksh and bash (faster).

`[ condition ]`

- synonym of command `test výraz`

`[[ condition ]]`

- only in ksh and bash, built-in command
- `-a` is replaced by `&&` and `-o` is replaced by `||`

`(( expression))`

- only in ksh and bash
- returns true if expression is not equals to zero

File attributes

Option	Example	Meaning
`-f`	`[ -f file ]`	True if `file` exists and is a regular file.
`-d`	`[ -d file ]`	True if `file` exists and is a directory.
`-s`	`[ -s file ]`	True if `file` exists and has a size greater than zero.
`-e`	`[ -e file ]`	True if `file` exists.
`-L`	`[ -L file ]`	True if `file` exists and is a symbolic link.
`-r`	`[ -r file ]`	True if `file` exists and is readable.
`-w`	`[ -w file ]`	True if `file` exists and is writable.
`-x`	`[ -x file ]`	True if `file` exists and is executable.

- See: `man sh`, `man ksh`, `man bash`, `man test`.

File attributes - examples

```
$ test -f /etc/passwd ; echo $?
0
$  [ -f /etc/passwd ] ; echo $?
0

$ name="/etc/passwd"
$ if [ -f "$name" ]; then echo "$name is file"; fi
/etc/passwd is file

$ name="/etc/xxx"
$ if [ -f "$name" ]; then echo "$name is file"; fi

$ if [ -f "$name" ]; then echo "$name is file"; \
> else echo "$name is not file" ; fi
/etc/xxx is not file
```

File attributes - examples

```
$ name="/etc/passwd"
$ if [ -f "$name" -a -r "$name" ]; then cat "$name"; fi
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
avahi:x:497:496:User for Avahi:/var/run/avahi-daemon:/bin/false
…

$ name="$HOME/muj file.txt"
$ if [ -d "$HOME" -a -w "$HOME" ]; then echo "Hello" > "$name"; fi
$ if [ -f $name -a -r $name ]; then cat $name; fi
bash: [: too many arguments

$ if [-f "$name" -a -r "$name"]; then cat "$name"; fi
If '[-f' is not a typo you can use command-not-found to lookup ...

$ if [ -f "$name" -a -r "$name" ]; then cat "$name"; fi
Hello
```

String comparison

Test	Meaning
`[ s1 = s2 ]`	Is string **s1** equal to string **s2** ?
`[ s1 != s2 ]`	Isn't string **s1** equal to string **s2** ?
`[ s1 \< s2 ]`	Is string **s1** less than string **s2** ? (in alphabetical order)
`[ s1 \> s2 ]`	Is string **s1** greater than string **s2** ? (in alphabetical order)
`[ -z s1 ]`	Has string **s1** zero length?
`[ -n s1 ]`	Hasn't string **s1** zero length?

- Note
  - If shell variable is used as argument, then put quotes around the variable to avoid error.

String comparison - examples

```
$ test "John" \< "Petr" ; echo $?
0
$ [ "John" \< "Petr" ] ; echo $?
0

$ A=Alex ; B=John ; C="Good Morning"
$ test $A \< $B ; echo $?
0
$ test $A = $B ; echo $?
1
$ test $B \< $C ; echo $?
-bash: test: too many arguments
2
$ test "$B" \< "$C" ; echo $?
1
```

Integer comparison

Test	Meaning
`[ n1 –eq n2 ]`	Is number **`n1`** equal to number **`n2`**?
`[ n1 –ne n2 ]`	Isn't number **`n1`** equal to number **`n2`**?
`[ n1 –lt n2 ]`	Is number **`n1`** less than number **`n2`**?
`[ n1 –gt n2 ]`	Is number **`n1`** greater than number **`n2`**?
`[ n1 –le n2 ]`	Is number **`n1`** less than or equal to number **`n2`**?
`[ n1 –ge n2 ]`	Is number **`n1`** greater than or equal to number **`n2`**?

Integer comparison - examples

```
$ test 2 -lt 7 ; echo $?
0
$ [ 2 -lt 7 ] ; echo $?
0

$ test 2 -gt 7 ; echo $?
1
$ [ 2 -gt 7 ] ; echo $?
1

$ A=10 ; B=7
$ test $A -eq $B || echo "$A is not equal to $B"
10 is not equal to 7
$ [ $A -gt $B ] &&  echo "$A > $B"
10 > 7
```

Command if / then / else

```
if list1; then list2; [ else list3; ]  fi
```

```
if list1
  then
     list2
  [ else
     list3 ]
fi
```

- The `list1` is executed.
- If its exit status is zero, the `list2` is executed.
- Otherwise, the `list3` is executed, if present.
- Line with `fi` must end by newline character!!!

Command if / then / else - example

```
#! /bin/sh

name="$1"

if [ -f "$name" ]
  then
    echo "$name is file"
  else
    if [ -d "$name" ]
      then
        echo "$name is directory"
      else
        if [ -b "$name" -o -c "$name" ]
          then
            echo "$name is special file"
          else
            echo "$name is not file/directory/special file"
        fi
    fi
fi
```

if-cmd-01.sh

Command if / then / else - example

```
#! /bin/sh

name="$1"

if [ -f "$name" ]
   then
      echo "$name is file"
elif [ -d "$name" ]
   then
      echo "$name is directory"
elif [ -b "$name" -o -c "$name" ]
   then
      echo "$name is special file"
else
      echo "$name is not file, nor directory"
fi
```

if-cmd-02.sh

Command if / then / else - example

```
#! /bin/sh

# Check the input parameters
if [ $# -ne 2 ]
  then
    echo "Usage: $0 number1 number2" >&2
    exit 2
fi

# Find the maximum
if [ "$1" -gt "$2" ]
  then
    echo $1
  else
    echo $2
fi
```

if-cmd-03.sh

Command case - example

```
#!/sbin/sh

case "$1" in
'start')
        [ -x /usr/lib/lpsched ] && /usr/lib/lpsched
        ;;

'stop')
        [ -x /usr/lib/lpshut ]  && /usr/lib/lpshut
        ;;

*)
        echo "Usage: $0 { start | stop }" >&2
        exit 1
        ;;
esac
```

Command case

```
case word in [ [(] pattern [ | pattern ] ... )list ;; ] ... esac
```

```
case word in
    [ [(] pattern [ | pattern ] ... ) list ;; ]
     ...
esac
```

- A case command first expands **word**.
- Then it tries to match **word** against each **pattern** in turn, using the same matching rules as for pathname expansion.
- If the operator **;;** is used, no subsequent matches are attempted after the first pattern match.

Command case - example

```
#!/bin/sh

export LANG="en_US.utf8"
export LC_ALL="en_US.utf8"

case "$(date '+%a')" in
'Mon' | 'Tue' | 'Wed' | 'Thu' | 'Fri' )
        echo "Today is working day."
        ;;

'Sat' | 'Sun' )
        echo "Today is weekend."
        ;;

*)
        echo "Something is wrong." >&2
        exit 2
        ;;
esac
```

case-cmd-01.sh

Loop while

```
while list1; do list2; done
```

```
while list1
do
    list2
done
```

- The while command continuously executes the list `list2` as long as the last command in the list `list1` returns an exit status of zero.

Loop while - example

```
#!/bin/bash

MAX="$1"
I="1"

while [ "$I" -le "$MAX" ]
do
  echo "Value of I is $I"
  I=$((I + 1))
done
```

while-cmd-01.sh

Loop until

```
until list1; do list2; done
```

```
until list1
do
    list2
done
```

- The until command is identical to the while command, except that the test is negated.
- List `list2` is executed as long as the last command in `list1` returns a non-zero exit status.

Loop until - example

```
#!/bin/bash

MAX="$1"
I="1"

until [ "$I" -gt "$MAX" ]
do
  echo "Value of I is $I"
  I=$((I + 1))
done
```

until-cmd-01.sh

Loop for

```
for name [ [ in [ word ... ] ] ; ] do list ; done
```

```
for name [ [ in [ word ... ] ] ; ]
do
    list
done
```

- The list of words following `in` is expanded, generating a list of items.
- The variable name is set to each element of this list of items in turn, and list `list` is executed each time.
- If the `in word ...` is omitted, the for command executes list `list` once for each positional parameter that is set.

Loop for - example

```
#!/bin/sh

for I in 1 2 3 4 5
do
  echo "Value of I is $I"
done
```

for-cmd-01.sh

Loop for - example

```
#!/bin/sh

for (( I=1; I<=5; I++))
do
  echo "Value of I is $I"
done
```

for-cmd-02.sh

Loop for - example

```
#!/bin/sh

for name in "$@"
do

   if [ -f "$name" ]
      then
         echo "$name is file"
      else
         if [ -d "$name" ]
            then
               echo "$name is directory"
            else
               echo "$name is not file, nor directory"
         fi
   fi
done
```

for-cmd-03.sh

Loop for - example

```
#!/bin/sh

# For all files in the current directory
for name in *
do
  echo "File:  $name"
done
```

for-cmd-04.sh