

# Programming in Shell 1

Exit status. Command test. Flow control. Loops

Jan Trdlička



Czech Technical University in Prague, Faculty of Information Technology  
Department of Computer Systems

## 1 Exit status

## 2 Command test

- File attributes
- String comparison
- Integer comparison

## 3 Flow control

- Command `if/then/else`
- Command `case`

## 4 Loops

- While
- Until
- For

- Every process returns exit status during termination.
- **Exit status** = integer 0, 1, 2, ...
  - Success: 0.
  - Error: 1, 2, ...
- **Built-in shell variable ?** contains the exit status of the last foreground command.
- Shell script can be terminated with exit status `n` by command  
`exit [n]`
- Shell function can be terminated with exit status `n` by command  
`return [n]`

# Exit status – examples

- See manual page of the command `grep`.
  - A line with the pattern was found by `grep`.

```
~> grep 'root' /etc/passwd
root:x:0:0:root:/root:/bin/bash
~> echo $?
0
```

- A line with the pattern was not found by `grep`.

```
~> grep 'XXX' /etc/passwd
~> echo $?
1
```

- Input file was not found by `grep`.

```
~> grep 'root' /XXX
grep: /XXX: No such file or directory
~> echo $?
130
```

- Command `XXXgrep` was not found by shell.

```
~> XXXgrep 'root' /etc/passwd
If '['-f' is not a typo you can use command-not-found ...
~> echo $?
127
```

# Command test

- Command `test` exits with the exit status determined by expression.
- Arguments are analysed by `test`, therefore
  - they must be separated by `space` or `TAB`,
  - use `symbol \` before meta-character.
- To get more info about the command use `help test` (built-in command) or `man test` (for binary program).
- Compound expressions
  - ( A )      Brackets 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).

- `[ expression ]`

- Synonym of the command test.

- `[[ expression ]]`

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

- `(( arithmetic_expression ))`

- Only in ksh and bash.
- The arithmetic expression is evaluated.
- Command returns true if expression is not equals to zero.

# File attributes

- Command `test` enable to check some attributes of files/directories.
- For more info use `help test`.

Option	Example	Meaning
<code>-f</code>	<code>[ -f file ]</code>	True if file exists and is a regular file.
<code>-d</code>	<code>[ -d file ]</code>	True if file is a directory.
<code>-e</code>	<code>[ -e file ]</code>	True if file exists.
<code>-s</code>	<code>[ -s file ]</code>	True if file exists and is not empty.
<code>-L</code>	<code>[ -L file ]</code>	True if file is a symbolic link.
<code>-r</code>	<code>[ -r file ]</code>	True if file is readable by you.
<code>-w</code>	<code>[ -w file ]</code>	True if the file is writable by you.
<code>-x</code>	<code>[ -x file ]</code>	True if the file is executable by you.

# File attributes – examples

- Test if `/etc/passwd` is a regular file.

```
~> test -f /etc/passwd ; echo $?  
0
```

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

- Test if `/etc/passwd` is a regular file and do something in dependence on the test result.

```
~> 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

- If the you can modify the content of directory, then create new file in the directory.

```
~> name="my_file.txt"
~> if [ -d "$HOME" -a -w "$HOME" ] ; then\
    echo "Hello" > "$HOME/$name" ; fi
```

- Badly written command.

```
~> cd
~> if [ -f $name -a -r $name ] ; then wc -l < $name ; fi
bash: [: too many arguments
```

- Badly written command.

```
~> if [-f "$name" -a -r "$name" ] ; then wc -l < "$name" ; fi
If '['-f' is not a typo you can use command-not-found ...
```

- Badly written command.

```
~> if [ -f "$name" -a -r "$name"] ; then wc -l < "$name" ; fi
bash: [: missing ']'
```

# String comparison

- Shell can compare strings.
- If shell variable is used as argument, then put quotes around the variable to avoid error.
- For more info use `help test`.

Expression	Meaning
<code>[ str1 = str2 ]</code>	True if the strings are equal.
<code>[ str1 != str2 ]</code>	True if the strings are not equal.
<code>[ str1 \&lt; str2 ]</code>	True if str1 sorts before str2 lexicographically.
<code>[ str1 \&gt;str2 ]</code>	True if str1 sorts after str2 lexicographically.
<code>[ -z str ]</code>	True if string is empty..
<code>[ -n str ]</code>	True if string is not empty..

# String comparison – examples

- Check lexicographical order of two strings.

```
~> test "John" \< "Peter" ; echo $?  
0
```

```
~> [ "John" \< "Peter" ] ; echo $?  
0
```

```
~> A="Alex" B="John" C="Good morning"  
~> test "$A" \< "$B" ; echo $?  
0
```

- Comparison of two strings

```
~> test "$A" = "$B" ; echo $?  
1
```

- Badly written command.

```
test $B = $C ; echo $?  
bash: test: too many arguments  
2
```

# Integer comparison

- Shell can compare integer numbers.
- For more info use `help test`.

Expression	Meaning
<code>[ num1 -eq num2 ]</code>	True if the numbers are equal.
<code>[ num1 -ne num2 ]</code>	True if the numbers are not equal.
<code>[ num1 -lt num2 ]</code>	True if the num1 is less than num2.
<code>[ num1 -le num2 ]</code>	True if the num1 is less than or equal to num2.
<code>[ num1 -gt num2 ]</code>	True if the num1 is greater than num2.
<code>[ num1 -ge num2 ]</code>	True if the num1 is greater than or equal to num2.

# Integer comparison – examples

- Comparison of two numbers.

```
~> test 2 -lt 7 ; echo $?  
0
```

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

- Compare two numbers and do something in dependence on the test result.

```
~> A="10" B="7"  
~> if test \! "$A" -eq "$B" \  
    then echo "$A is not equal to $B." ; fi  
10 is not equal to 7.
```

```
~> [ "$A" -eq "$B" ] || echo "$A is not equal to $B."  
10 is not equal to 7.
```

# Command if/then/else

- Single-line syntax

```
if LIST1 ; then LIST2 ; [ else LIST3 ; ] fi
```

- Multiple line syntax

```
if LIST1
then
    LIST2
[ else
    LIST3 ]
fi
```

- The LIST1 of commands is executed.
- If its exit status is 0, the LIST2 of commands is executed.
- Otherwise, the LIST3 of commands is executed, if present.
- Line with **fi** must end by newline character!!!
- For more info use `help case` or `man bash`.

# Command if/then/else – example

- The shell script that tries to find the file type.

```
#!/bin/bash

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
```

# Command if/then/else – example

- The shell script that tries to find the file type.
- Better syntax in this case: `else if` is replaced by `elif`.

```
#!/bin/bash

name="$1"

# --- regular file ---
if [ -f "$name" ] ; then
    echo "$name is file"

# --- directory ---
elif [ -d "$name" ] ; then
    echo "$name is directory"

# --- special file ---
elif [ -b "$name" -o -c "$name" ] ; then
    echo "$name is special file"

# --- the others ---
else
    echo "$name is not file/directory/special file"
fi
```



# Command if/then/else – example

- The shell script checks the number of arguments and then prints a greater i number.

```
#!/bin/bash

name="$1"

# --- check number of arguments ---
if [ $# -ne 2 ] ; then

    echo "Usage: $0 interger1 integer2" >&2
    exit 1
fi

# --- print greater integer ---
if [ "$1" -gt "$2" ] ; then

    echo "$1"

else

    echo "$2"

fi
```

# Command case

- Single-line syntax

```
case WORD in [ PATTERN1 [ | PATTERN2 ] ... ) LIST ;; ] ... esac
```

- Multiple line syntax

```
case WORD in  
    [ PATTERN1 [ | PATTERN2 ] ... ) 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.
- Line with esac must end by newline character!!!
- For more info use [help case](#) or [man bash](#).

# Command case – example

- The shell script will check the argument and try to start or stop some system service.

```
#!/bin/bash

case "$1" in
    "start" )
        [ -x /usr/bin/lib/lpsched ] && /usr/bin/lib/lpsched
        ;;
    "stop" )
        [ -x /usr/bin/lib/lpshut ] && /usr/bin/lib/lpshut
        ;;
    *)
        echo "Usage: $0 [ start | stop ]" >&2
        exit 1
        ;;
esac
```

# Command case – example

- The shell script will set the environment to en\_US.utf8.
- The shell script will try to determine the type of the day.

```
#!/bin/bash

# --- set environment ---
export LANG="en_US.utf8"
export LC_ALL="en_US.utf8"

# --- determine type of day ---
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 1
        ;;
esac
```

# While loop

- Single-line syntax

```
while LIST1 ; do LIST2; done
```

- Multiple line syntax

```
while LIST1
do
    LIST2
done
```

- The while command continuously executes the list of commands LIST2 as long as the last command in the list LIST1 returns an exit status of zero.
- Line with do must end by newline character!!!
- For more info use `help case` or `man bash`.

# While loop – example

- The shell script will check the argument and prints numbers 1,...,MAX.

```
#!/bin/bash

# --- check arguments ---
if [ $# != 1 ] ; then
    echo "Usage: $0 integer" >&2
    exit 1
fi

# --- print numbers ---
I="1"          # from
MAX="$1"       # to

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

# Loop until

- Single-line syntax

```
until LIST1 ; do LIST2; done
```

- Multiple line syntax

```
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.
- Line with do must end by newline character!!!
- For more info use `help case` or `man bash`.

# Loop until – example

- The shell script will check the argument and prints numbers 1,..., *MAX*.

```
#!/bin/bash

# --- check arguments ---
if [ $# != 1 ] ; then
    echo "Usage: $0 integer" >&2
    exit 1
fi

# --- print numbers ---
I="1"          # from
MAX="$1"       # to

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



# For loop

- Single-line syntax

```
for NAME [ in WORDS ... ] do LIST; done
```

- Multiple line syntax

```
for NAME [ in WORDS ... ]  
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 is executed each time.
- If the in WORDS ... is omitted, the for command executes LIST once for each positional parameter that is set.
- Line with do must end by newline character!!!

# For loop – example

- The shell script will repeat steps 1, 2, ..., 5.

```
#!/bin/bash

for I in 1 2 3 4 5
do
    echo "Step $I"
    I=$(( I + 1 ))
done
```

- Same behaviour, but different syntax.

```
#!/bin/bash

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

# For loop – example

- The shell script will check argument and print all script arguments.

```
#!/bin/bash

# --- check arguments ---
if [ $# -eq 0 ] ; then
    echo "Usage: $0 arg1 [ arg2 ... ]" >&2
    exit 1
fi

# --- print script arguments ---
I="1"

for ARG
do
    echo "arg$I = \"$ARG\""
    I=$((I+1))
done
```

# For loop – example

- The shell script will check argument and print all regular files from the given directory.

```
#!/bin/bash

# --- check arguments ---
if [ \! \ ( $# -eq 1 -a -d "$1" -a -r "$1" \ ) ] ; then
    echo "Usage: $0 directory" >&2
    exit 1
fi

# --- print all items of given directory ---
I="1"

for NAME in "$1/"* "$1/".*
do
    if [ -f "$NAME" ] ; then
        echo "  file$I = \"$NAME\""
        I=$((I+1))
    fi
done
```