Lecture 8

Numeric calculation,

compression and archiving,

exit code.

Department of Computer Systems FIT, Czech Technical University in Prague

Integer arithmetic

- **Command:** `expr expression`
  - The expr utility evaluates the expression and writes the result to standard output.
  - Terms of the expression must be separated by blanks.
  - In front of special shell characters use character \.

Operator	Meaning	Example
`+`	adding	`N=`expr $N1 + 3``
`-`	substraction	`N=`expr $N1 - $N2``
`*`	multiplication	`N=`expr 10 * 21``
`/`	Integer dividing	`N=`expr $N1 / $N2``
`%`	Remainder of dividing	`N=`expr $N1 % 5``

Integer arithmetic

- **Expression is evaluated by priority** (like in mathematic):
  - First \(expression \)
  - After operations *, /, %
  - At the end operations + and -
- Operations of the same priority are evaluated from left to right.
- **Examples:**

```
$ A=`expr  5 + 3  \* 2`
$ echo $A
11
$ A=`expr \( 5 + 3 \) \* 2`
$ echo $A
16
```

Integer arithmetic

- **Built-in shell command `let expression` or `((expression))` (except of sh)**
  - Operands and operators needn't be separated by spaces.
  - Variables are automatically replaced by their values (don't use character $).

Operator	Meaning	Example
`+`	adding	`((N = N1 + 3))`
`-`	subtraction	`((N = N1 - N2))`
`*`	multiplication	`((N = 10 * 21))`
`/`	integer dividing	`((N = N1 / N2))`
`%`	remainder of dividing	`((N = N1 % 5))`
`#`	base	`((N=2#1011))`
`<<`	bit left shifting	`((N= 2#1011 << 3))`
`>>`	bit right shifting	`((N= 2#1011 >> 3))`

Integer arithmetic

Operator	Meaning	Example
&	AND	((N = 2#1011 & 2#1101))
|	OR	((N = 2#1011 | 2#1101))
^	XOR	((N = 2#1011 ^ 2#1101))

Flouting point arithmetic

- **Command `bc [-c] [-l] file]`**
  - Preprocessor of command **`dc`** (**`-c`** prints commands for **`dc`**)
  - **`-l`** load mathematic library and **`scale`**=20
  - Commands are read from file otherwise from stdin

Operators	Meaning	Examples
`+`	adding	`N=`echo "$N1 + $N2" | bc``
`-`	subtraction	`N=`echo "$N1 - $N2" | bc``
`*`	multiplication	`N=`echo "$N1 * $N2" | bc``
`/`	integer dividing	`N=`echo "$N1 / $N2" | bc``
`%`	remainder of dividing	`N=`echo "$N1 % $N2" | bc``
`^`	power of two	`N=`echo "$N1 ^ $N2" | bc``
`< <= > >=`	less than, …	
`== !=`	equal to,…	

Flouting point arithmetic

Keywords	Meaning	Examples
`ibase`	input base	`N=`echo "ibase=16; A + B" | bc``
`obase`	output base	`N=`echo "obase=2; 5 + 2" | bc``
`scale`	decimal places (default 0)	`N=`echo "scale 5; 10 / 3" | bc``

Identifiers	Meaning	Examples
`x`	variable (lower case letter)	`N=`echo "a=5;b=2; a + b" | bc``
`x[i]`	`i`-th element of array `x`	`N=`echo "a[1]=3; a[1]+1" | bc``
`x(y,z)`	function `x` with parameters `y` and `z`	`N=`echo “length(3.1415)" | bc``

Flouting point arithmetic

Functions	Meaning	Examples
`sqrt(x)`	square root of	`N=`echo "sqrt($A)" | bc``
`l(x)`	natural log	`N=`echo "l($A)" | bc``
`e(x)`	e^x	`N=`echo "e($A)" | bc``
`s(x)`	sin(x)	
`c(x)`	cos(x)	
`length(x)`	digit number of x	

- **Command `awk/nawk`**

Archiving and compression

- **Archive**
  - is file containing packed files and directories
- **File compression**
  - is the process of encoding information using fewer bits(or other information-bearing units) than an original representation would
  - Lossless data compression
- **Usage**
  - data transfer
  - backup (complete, incremental!!!)
- **Backup problems**
  - absolute/relative path
  - file attributes (owner, modification time, …)
  - hard link
  - soft link

Archiving

- **Command `tar`** (Tape ARchive)
- **C**reate archive (default suffix is .tar)

```
cd directory
tar cvf archiv.tar directory/files
find . > list.txt ; tar cvf archive.tar –I list.txt
```

- **U**pdate archive

```
cd directory
tar uvf archive.tar directory/files
```

- **T**est archive (list contet of archive)

```
tar tvf archive.tar
```

- **E**xtract archive

```
tar xv[op]f archive.tar
```

- **R**ecover files from archive

```
tar rv[op]f archive.tar files
```

Compression

- **Commands `compress, uncompress, zcat`**
  - Data compression algorithm is LZW (Lampel-Ziv-Welch code)
  - Compression (suffix is .Z)

```
compress file
cat file| compress > file.Z
tar cv files | compress > archive.tar.Z
```

  - Decompression

```
uncompress [-f] file.Z
zcat file.tar.Z | tar xvf -
```

Compression

- **Commands `gzip, gunzip, gzcat`**
  - Data compression algorithm is LZ77 (Lempel-Ziv code)
  - Compression (default suffix is .gz)

```
gzip [-9] file
cat file | gzip > file.gz
tar cv files | gzip > archive.tar.gz
```

  - Decompression

```
gunzip file.gz
gzcat file.tar.gz | tar xvf -
```

Compression

- **Commands `bzip2, bunzip2, bzcat`**
  - Data compression use combination of algorithms BWT (Burrows-Wheelerova transformation), MTF (Move-to-Front) transformation and Huffman code
  - Compression (default suffix is .bz2)

```
bzip2 [-9] file
cat file | bzip2 > file.bz2
tar cv files | bzip2 > archive.tar.bz2
```

  - Decompression

```
bunzip file.bz2
bzcat file.tar.bz2 | tar xvf -
```

Archiving and compression

- **Commands `zip, unzip`**
  - Use format created by Philem Katzem (program PKZIP).
  - Creation of compress archive (default suffix is .zip)

```
zip archive.zip files
zip –r[9] archive.zip directories
```

  - Listing of content

```
unzip –l archive.zip
```

  - Extraction of archive

```
unzip archive.zip [directories/files]
```

Archiving and compression

- **Command `jar`** (Java ARchive tool)
  - Use formats ZIP and ZLIB.
  - Originally developed for archiving of JAVA packages.
  - Syntax similar like command tar.

- **C**reation of compress archive (default suffix is .jar)

```
jar cvf archive.jar directories/files
```

- **T**est of archive

```
jar tvf archive.jar
```

- **E**xtraction of archive

```
jar xv[op]f archive.jar
```

Archiving and compression

- **Command `gtar`** (GNU tar)
- GNU implementation of command tar
- More clever (e.g. it can call commands for data compression)

- Creation of compress archive

```
gtar cvZf archiv soubory      (calling compress)
gtar cvzf archiv soubory      (calling gzip)
gtar cvJf archiv soubory      (calling bzip2)
```

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]`

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 and its synonyms

`test condition`

- The test utility evaluates the condition and indicates the result of the evaluation by its exit status.

`[ 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

- The command **`test`** is built-in command in ksh and bash (faster).

Examples

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

$ test -d /etc/passwd ; echo $?
1
$ [ -d /etc/passwd ] ; echo $?
1

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

Examples

```
$ P="/etc/group"
$ [ -r $P ] ; echo $?
0

$ [ -r $P ] && echo “file $P is readable"
file /etc/group is readable

$ P="/etc/shadow"
$ [ -r $P ] ; echo $?
1

$ [ -r $P ] || echo "file $P is not readable"
file  /etc/shadow is not readable

$ ! [ -r $P ] && echo "file $P is not readable"
file /etc/shadow is not readable
```

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 not equal to $B"
10 not equal to 7
$ [ $A -gt $B ] &&  echo "$A > $B"
10 > 7
```

Example – exit code

```
#!/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 }"
        exit 1
        ;;
esac
```

Example – exit code

- **Skript while1.sh:**

```
#!/bin/sh

MAX=5
I=1

while [ $I -le 10 ]
do
  echo "Value I is $I"
  I=`expr $I + 1`
done
```