

Unix-like Operating Systems

Data compression and archiving

Jan Trdlička



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

1 Archiving

- tar

2 Compression

- compress, uncompress, zcat
- gzip, gunzip, gzcat
- bzip2, bunzip2, bzcac

3 Archiving and compression

- zip, unzip
- fastjar/jar

4 Integer arithmetic

- expr, let, (()), awk

5 Floating point arithmetic

- bc, awk

Archiving and compression

- Archive

- File containing packed files and directories.

- File compression

- Process of encoding information using fewer bits (or other information-bearing units) than an original representation.
- Lossless data compression.

- Usage

- Data transfer.
- Backup (complete, incremental).

- Backup problems

- Absolute/relative path.
- File attributes (owner, modification time, ...).
- Hard link.
- Soft link.
- Deleted items.

- **tar** (Tape **AR**chive)
 - **Create archive**(default suffix is .tar)

```
cd directory ; tar -cvf archive.tar *
```

```
find . > list.txt  
tar -cvf archive.tar -I list.txt      # Solaris  
tar -cvf archive.tar -T list.txt      # GNU
```

- **Test archive** (list content of archive)

```
tar -tvf archive.tar
```

- **Extract archive**

```
tar -xvf ../archive.tar
```

```
tar -xvpf ../archive.tar      // preserve permissions
```

Compression – compress

- `compress`, `uncompress`, `zcat`

- Data compression algorithm is LZW (Lempel-Ziv-Welch code).
- `Compression` (suffix is `.Z`)

```
compress file
```

```
cat file | compress > file.Z
```

```
tar cvf - -T list.txt | compress > archive.tar.Z
```

```
tar -cvZf archive.tar.Z -T list.txt # only GNU
```

- `Decompression`

```
uncompress file.Z
```

```
zcat file.tar.Z | tar xvf -
```

```
tar -xvf file.tar.Z # only GNU
```

Compression – gzip

- `gzip`, `gunzip`, `gzcat`

- Data compression algorithm is LZ77 (Lempel-Ziv code)
- `Compression` (default suffix is `.gz`)

```
gzip file
```

```
cat file | gzip > file.gz
```

```
tar cvf - -I list.txt | gzip > archive.tar.gz      # Solaris
```

```
tar cvf - -T list.txt | gzip > archive.tar.gz      # GNU
```

```
tar -cvzf archive.tar.gz -T list.txt              # only GNU
```

- `Decompression`

```
gunzip file.gz
```

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

```
tar -xvf file.tar.gz                             # only GNU
```

Compression – bzip2

- **bzip2, bunzip2, bzcac**

- Data compression use combination of algorithms BWT (Burrows-Wheeler transformation), MTF (Move-to-Front) transformation and Huffman code.

- **Compression** (default suffix is .bz2)

```
gzip file
```

```
cat file | bzip2 > file.bz2
```

```
tar cvf - -I list.txt | bzip2 > archive.tar.bz2 # Solaris
```

```
tar cvf - -T list.txt | bzip2 > archive.tar.bz2 # GNU
```

```
tar -cvJf archive.tar.bz2 -T list.txt # only GNU
```

- **Decompression**

```
bunzip file.bz2
```

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

```
tar -xvf file.tar.bz2 # only GNU
```

Archiving and compression – zip

- zip, unzip

- Use format created by Phil Katz (program PKZIP).

- Creation of compress archive (default suffix is .zip)

```
zip archive.zip files
```

```
zip archive.zip -@ < list.txt
```

- Listing of content

```
unzip -l archive.zip
```

- Extraction of archive

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


Archiving and compression – fastjar/jar

- **fastjar/jar** (Java ARchive tool)
 - Use formats ZIP and ZLIB.
 - Originally developed for archiving of JAVA packages.
 - Syntax similar like command tar.
 - **Creation of compress archive** (default suffix is .jar)

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

```
fastjar cvf elf.jar -@ < list.txt
```

- **Test (list) of archive**

```
fastjar -tvf archive.jar
```

- **Extraction of archive**

```
fastjar -xvf archive.jar
```

Integer arithmetic – exp

- *expr expression*

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

Operator	Meaning	Example
+	addition	<code>N=\$(expr \$N1 + 3)</code>
-	subtraction	<code>N=\$(expr \$N1 - 3)</code>
*	multiplication	<code>N=\$(expr \$N1 * 3)</code>
/	integer division	<code>N=\$(expr \$N1 / 3)</code>
%	remainder of integer division	<code>N=\$(expr \$N1 % 3)</code>

Integer arithmetic – exp

- Expression is evaluated by priority (like in mathematics)
 - 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 – let or (())

- `let expression` or `((expression))`
 - Built-in commands (see `help let`).
 - Operands and operators may not be separated by spaces.
 - Variables are automatically replaced by their values (don't use character `$`).

Operator	Meaning	Example
<code>+</code>	addition	<code>((N = N1 + 3))</code>
<code>-</code>	subtraction	<code>((N = N1 - 3))</code>
<code>*</code>	multiplication	<code>((N = N1 * 3))</code>
<code>/</code>	integer division	<code>((N = N1 / 3))</code>
<code>%</code>	remainder of integer division	<code>((N = N1 % 3))</code>
<code>#</code>	base	<code>((N = 2#1011))</code>
<code><<</code>	bit left shifting	<code>((N = 2#1011 << 3))</code>
<code>>></code>	bit right shifting	<code>((N = 2#1011 >> 3))</code>

Flouting point arithmetic – bc

- `bc [-1] [file]`
 - `-1` ... load mathematic library.
 - Commands are read from the *file* otherwise from the standard input.
 - Result is printed to the standard output.

Operator	Meaning	Example
<code>+</code>	addition	<code>N=\$(echo "\$N1 + 3" bc -l)</code>
<code>-</code>	subtraction	<code>N=\$(echo "\$N1 - 3" bc -l)</code>
<code>*</code>	multiplication	<code>N=\$(echo "\$N1 * 3" bc -l)</code>
<code>/</code>	integer division	<code>N=\$(echo "\$N1 / 3" bc -l)</code>
<code>%</code>	remainder of division	<code>N=\$(echo "\$N1 % 3" bc -l)</code>
<code>^</code>	power	<code>N=\$(echo "2^3" bc -l)</code>
<code><, <=, >, >=</code>	less than, ...	
<code>==, !=</code>	equal to, ...	

Flouting point arithmetic – bc

Keyword	Meaning	Example
<code>scale</code>	decimal paces (default 0)	<code>N=\$(echo "scale=2 ; 10/3 " bc -l)</code>
<code>ibase</code>	input base	<code>N=\$(echo "ibase=16 ; A + B " bc -l)</code>
<code>obase</code>	output base	<code>N=\$(echo "obase=2 ; 5 + 2 " bc -l)</code>

Identifiers	Meaning	Example
<code>x</code>	variable (only lower case letters)	<code>N=\$(echo "x=2 ; y=5 ; x + y" bc -l)</code>
<code>x[i]</code>	<i>i</i> -th element of array <code>x</code>	<code>N=\$(echo "x[1]=2 ; x[1] + 3" bc -l)</code>
<code>x(y,z)</code>	function <code>x</code> with parameters <code>y</code> and <code>z</code>	<code>N=\$(echo "length(3.1415)" bc -l)</code>

Flouting point arithmetic – bc, awk

Function	Meaning	Example
<code>sqrt(x)</code>	square root of	<code>N=\$(echo "sqrt(\$A)" bc -l)</code>
<code>l(x)</code>	natural log	<code>N=\$(echo "l(\$A)" bc -l)</code>
<code>e(x)</code>	e^x	<code>N=\$(echo "e(\$A)" bc -l)</code>
<code>s(x)</code>	$\sin(x)$	<code>N=\$(echo "s(\$A)" bc -l)</code>
<code>c(x)</code>	$\cos(x)$	<code>N=\$(echo "c(\$A)" bc -l)</code>
<code>length(x)</code>	digits of number x	<code>N=\$(echo "length(\$A)" bc -l)</code>

- **awk**

- Inside the program of commad `awk` we can make integer/flouting point arithmetic.