DevOps School

Lesson 02. Bash Introduction

By Yuriy Bezgachnyuk, November 2021

softserve

Linux Shells Overview

- **Linux** has a variety of different **shells**:
  - Bourne Shell (sh)
  - C Shell (csh)
  - Korn Shell (ksh)
  - TC Shell (tcsh)
  - **Bourne Again Shell (bash)**
  - **Z Shell (zsh)**
  - …
- Certainly, the most popular shell is “bash”. Bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh)
- It is intended to conform to the *IEEE POSIX P1003.2/ISO 9945.2* Shell and Tools standard.
- It offers functional improvements over sh for both programming and interactive use

softserve

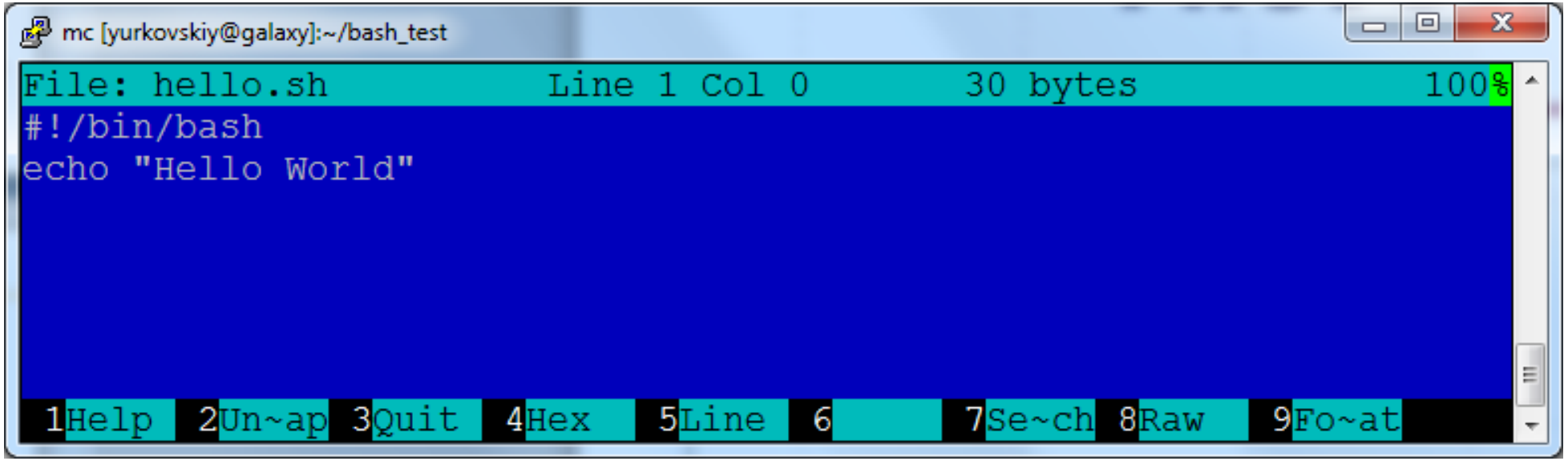
BASH Syntax

- Latin alphabet
- Arabic digits
- Punctuation symbols
- Some keywords
- …

softserve

BASH Syntax

- On the first line
- `#!/bin/bash`
- Tells the operating system that the following will be a script and not a regular text file



softserve

VARIABLES (1)

- We can use **variables** as in any programming languages. Their values are **always stored as strings**, but there are mathematical operators in the shell language that will *convert variables to numbers for calculations*.
- There is no needed to declare variables. Just assign a value to its reference will create it
- How to use variables

mc [yurkovskiy@galaxy]:~/bash_test

```
File: second.sh          Line 1 Col 0
#!/bin/bash
CDIR="BASH_TRASH"
echo "Creating directory...$CDIR"
mkdir $CDIR
cp * $CDIR
rm -rf $CDIR

 1Help  2Un~ap  3Quit  4Hex  5Line  6  7Se~ch  8Raw
```

Value assignment and variable declaration

Using Variable's value

softserve

VARIABLES (2)

- The shell programming language does **not type-cast** its variables.
  - This means that a variable can hold number data or character data
- Switching the TYPE of a variable can lead to confusion for the writer of the script or someone trying to modify it, so **it is recommended to use a variable for only a single TYPE of data** in a script
- **** is the bash **escape** character and it preserves the literal value of the next character that follows

```
count=0
count=Sunday
```

softserve

VARIABLES (3)

- When assigning character data containing ***spaces*** *or special characters*, the data must be enclosed in either **single or double quotes**.
- Using *double quotes* to show a string of characters will allow any variables in the quotes to be *resolved*.
- Using *single quotes* to show a string of characters will *not allow variable resolution*.

```
var="test string"
newvar="Value of var is $var"
echo $newvar
```

```
var='test string'
newvar='Value of var is $var'
echo $newvar
```

softserve

ENVIRONMENTAL VARIABLES (1)

- There are two types of variables:
  - Local Variables
  - Environmental Variables
- **Environmental variables** are set by the system and can usually be found by using the `env` command. Environmental variables hold special values
- Environmental variables are defined in */etc/profile*, */etc/profile.d/* and *~/.bash_profile*. These files are the initialization files, and they are read when bash shell is invoked.
- When a login shell exits, bash reads *~/.bash_logout*.
- The startup is more complex; for example, if bash is used interactively, then `/etc/bashrc` or `~/.bashrc` are read.
  - See the man page for more details.

softserve

ENVIRONMENTAL VARIABLES (2)

Name	Description
PATH	The search path for commands. It is a colon-separated list of directories that are searched when you type a command
HOSTNAME	Name of the host (computer)
USER, LOGNAME	Current logged in user
PS1	Sequence of characters shown before the prompt
PWD	Current working directory
SHELL	The path to the current command shell
…	

softserve

ENVIRONMENTAL VARIABLES (1)

- PS1: sequence of characters shown before the prompt
  - `\t` – hour
  - `\d` – date
  - `\w` – current directory
  - `\W` – last part of current directory
  - `\u` – user name
  - `\$` - prompt character

softserve

VARIABLES

- The read command allows you to prompt for input and store it in a variable

```
mc [yurkovskiy@galaxy]:~/bash_test
read_test.sh        [----] 18 L:[  1+ 3   4/  4] *(75  /  75b
#!/bin/bash
echo -n "Please enter your name: "
read name
echo "Hello $name"
 1Help  2Save  3Mark  4Re~ac  5Copy  6Move  7Se~ch  8De~te
```

softserve

SPECIAL VARIABLES

Parameter	Description
$0	Name of the current shell script
$1-$9	Positional parameters 1 through 9
${10}	Positional parameter 10
$#	The number of positional parameters
$*	All positional parameters, “$*” is one string
$@	All positional parameters, “$@” is a set of strings
$?	Return status of most recently executed command
$$	Process id of current process

softserve

ENVIRONMENTAL VARIABLES

- We have a next bash script
  - `#!/bin/bash`
  - `echo $#`
  - `echo $0`
  - `echo $1`

```
yurkovskiy@galaxy: ~/bash_test
yurkovskiy@galaxy:~/bash_test$ ./spec_var.sh var_1
1
./spec_var.sh
var_1
yurkovskiy@galaxy:~/bash_test$
```

softserve

COMMAND SUBSTITUTION

- The backquote “`” is different from the single quote “`”.
- It is used for command substitution: `command`

```
#!/bin/bash
list=`ls -l`
echo $list
```

softserve

ARITHMETIC EVALUATION

- The **let** statement can be used to do mathematical functions:
- An arithmetic expression can be evaluated by **$[expression]** or **$((expression))**
- ATTENTION, PLEASE BASH DON'T KNOW HOW TO WORK WITH **floating** point units ☺

```
mc [yurkovskiy@galaxy]:~/bash_test
ari.sh              [----] 17 L:[  1+ 5   6/  6] *(81  /  81b) <EOF>
#!/bin/bash
let X=10+2*4
echo $X
echo "$((10+40))"
val=$[20+40]
echo "$[10*$val]"

 1Help  2Save  3Mark  4Re~ac 5Copy  6Move  7Se~ch 8De~te 9Pu~Dn10Quit
```

softserve

ARITHMETIC EVALUATION

```
#!/bin/bash
echo $(( 100 / 3 ))
myvar="56"
echo $(( $myvar + 12 ))
echo $(( $myvar - $myvar ))
myvar=$(( $myvar + 1 ))
echo $myvar
```

softserve

CONDITIONAL STATEMENTS

- Conditionals let us decide whether to perform an action or not, this decision is taken by evaluating an expression. The most basic form is:

```
if [ expression ];
  then
       statements
  elif [ expression ];
  then
       statements
  else
       statements
  fi
```

- the elif (`else if`) and else sections are optional
- Put **spaces** after [and before], and around the operators and operands.

softserve

EXPRESSION (1)

- An expression can be:
  - String comparison
  - Numeric comparison
  - File operators and Logical operators and it is represented by [expression]:
- String Comparisons:

Expression	Description
=	compare if two strings are equal
!=	compare if two strings are not equal
-n	evaluate if string length is greater than zero
-z	evaluate if string length is equal to zero

softserve

EXPRESSION (2)

- Number Comparisons

Expression	Description
`-eq`	compare if two numbers are equal
`-ge`	compare if one number is greater than or equal to a number
`-le`	compare if one number is less than or equal to a number
`-ne`	compare if two numbers are not equal
`-gt`	compare if one number is greater than another number
`-lt`	compare if one number is less than another number

softserve

EXPRESSION

```
if [ "$myvar" -eq 3 ]
  then echo "myvar eq 3 as num"
fi

if [ "$myvar" = "3" ]
  then echo "myvar eq 3 as srt"
fi

if [ -z $1 ]; then echo "Empty Parameter"; fi
```

softserve

RELATIONAL OPERATORS

Meaning	Numeric	String
Greater than	-gt	
Greater than or equal	-ge	
Less than	-lt	
Less than or equal	-le	
Equal	-eg	= or ==
Not equal	-ne	!=
str1 is less than str2		str1 < str2
str1 is greater str2		str1 > str2
String length is greater than zero		-n str
String length is zero		-z str

softserve

RELATIONAL OPERATORS

- Script

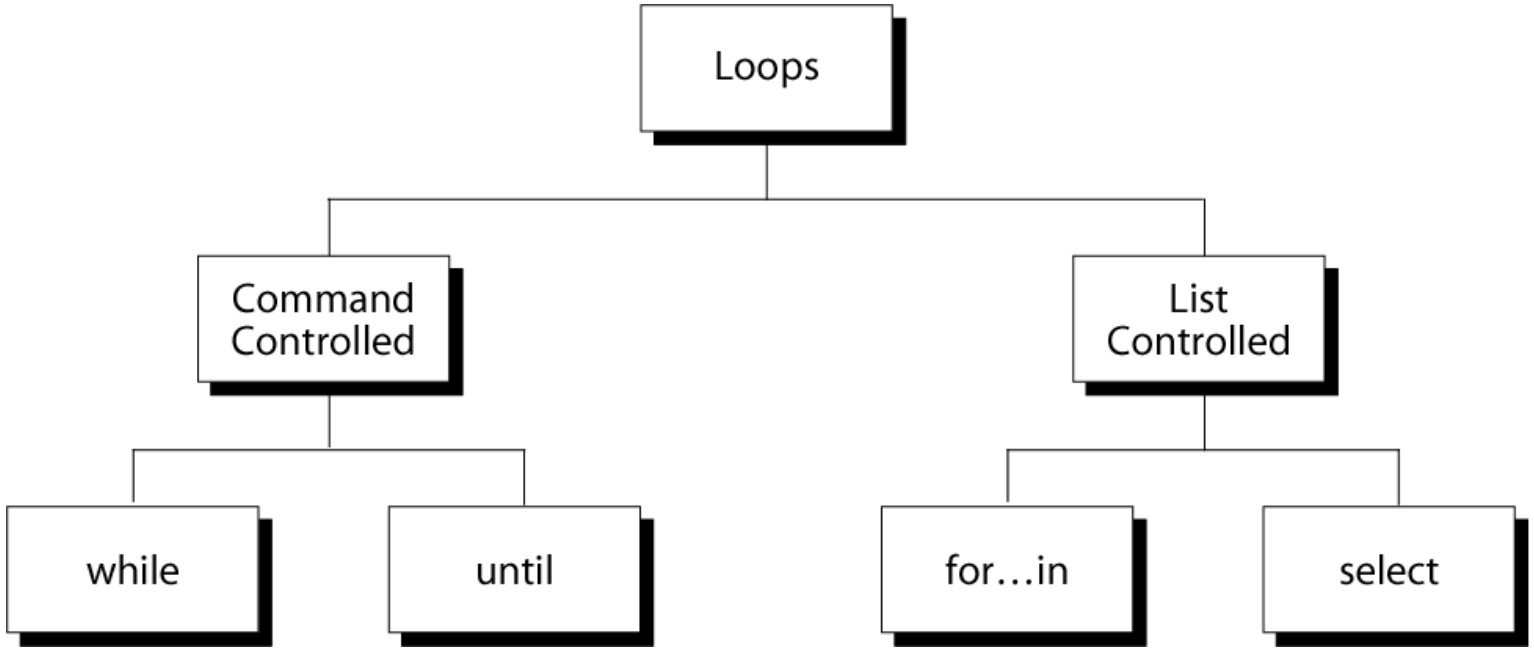
```
mc [yurkovskiy@galaxy]:~/bash_test
ifels.sh            [----]  0 L:[  1+ 9  10/ 10] *(164 / 164b) <EOF>
#!/bin/bash
echo -n "Enter your login name: "
read name
if [ "$name" = "$USER" ];
then
    echo "Hello $name"
else
    echo "You are not $USER, so who are you?"
fi

 1Help  2Save  3Mark  4Re~ac 5Copy  6Move  7Se~ch 8De~te 9PullDn10Quit
```

softserve

LANGUAGE ELEMENTS

- Control structures
  - Repetition
    - do-while, repeat-until
    - for … in
    - Select
  - Functions
  - Trapping signals



softserve

LANGUAGE ELEMENTS

- Purpose:
- To execute commands in “command-list” as long as “expression” evaluates to true

- Syntax:

```
while [ expression ]
do
        command-list
done
```

softserve

LANGUAGE ELEMENTS

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
    echo “The counter is $COUNTER”
    let COUNTER=$COUNTER+1
done
```

```
yurkovskiy@galaxy: ~/bash_test
yurkovskiy@galaxy:~/bash_test$ ./lupo_test.sh
The counter is 0
The counter is 1
The counter is 2
The counter is 3
The counter is 4
The counter is 5
The counter is 6
The counter is 7
The counter is 8
The counter is 9
yurkovskiy@galaxy:~/bash_test$
```

softserve

LANGUAGE ELEMENTS

```
#!/bin/bash
for x in one two three three four; do
  echo "number $x"
done

#!/bin/bash
for myfile in /etc/r*; do
  if [ -d "$myfile" ]
    then echo "$myfile (dir)"
  else
    echo "$myfile"
  fi
done
```

softserve

LANGUAGE ELEMENTS

- Repeat until true

```
myvar=0
while [ $myvar -ne 10 ]; do
  echo "$myvar"
  myvar=$(( $myvar + 1 ))
done
```

- Repeat until the value is false

```
myvar=0
until [ $myvar -eq 10 ]
  do
    echo $myvar
    myvar=$(( $myvar + 1 ))
done
```

softserve

LANGUAGE ELEMENTS

- Variable in function

```
#!/bin/bash
myvar="hello"
myfunc() {
  myvar="one two three"
  for x in $myvar; do
    echo $x
  done
}
myfunc
echo  "\$myvar = $myvar  \$x = $x"
```

- Result `one two three three`

softserve

LANGUAGE ELEMENTS

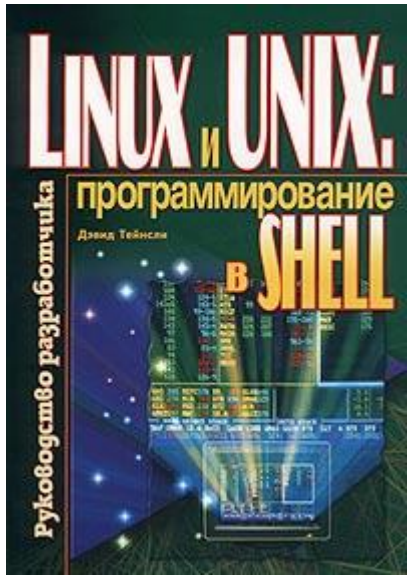
- A local variable in a function

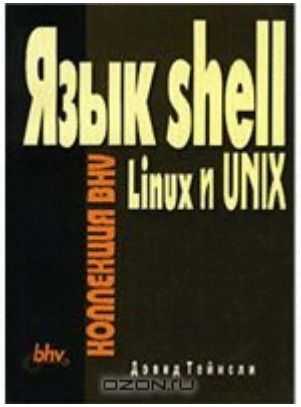
```
#!/bin/bash
myvar="hello"
myfunc() {
  local x
  local myvar="one two three"
  for x in $myvar; do
    echo $x
  done
}
myfunc
echo "\$myvar = $myvar  \$x = $x"
```

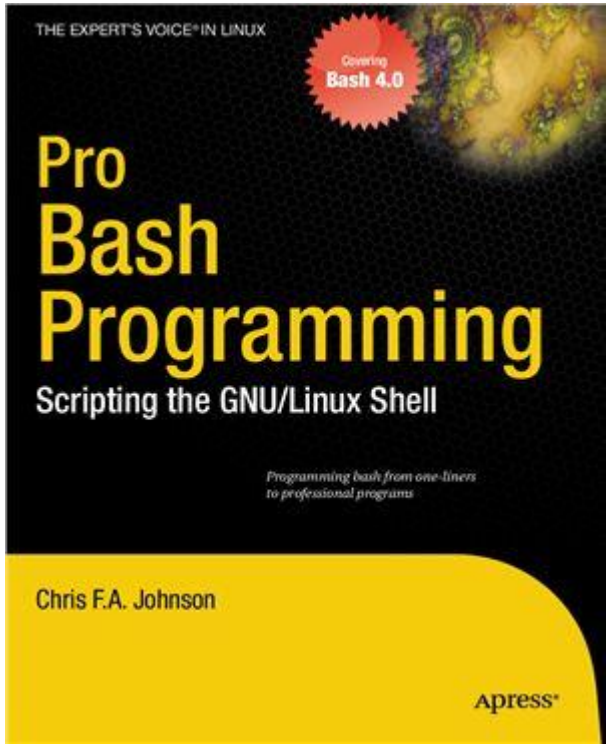
- Result `hello`

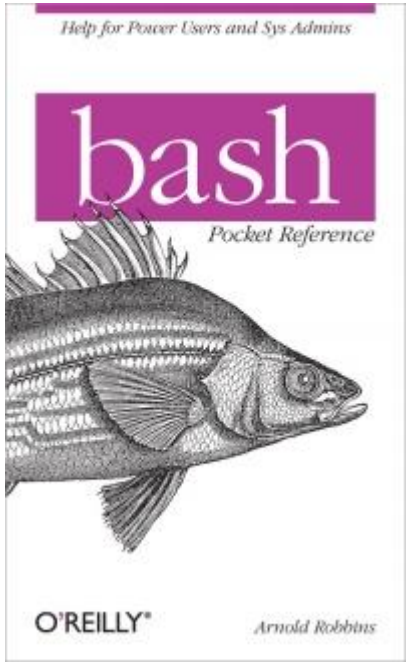
softserve

REFERENCES & SOURCES









https://www.gnu.org/software/bash/manual/

softserve


FOR
THE
FUTURE
softserve