Shell script Programming

Practical Class 2

Systems and Storage Laboratory
Department of Computer Science and Engineering
Chung-Ang University

Shell script programming

- **Basic Concepts and characteristic**
- **Variable**
- **If condition statement**
  - if – else – then
  - case – esac
- **Iteration statement**
  - for – in
  - while, until
  - break, continue, exit, return
- **Function**

Concept of shell

- **Shell in Linux**
  - **Interface** to execute commands and programs.
  - The role of **communication between kernel and user**
    - interpreting commands entered by a user to forward to the kernel
    - forward the result of processing by the kernel to the user
  - **Bash**(Bourne Again Shell) is the popular shell in many Linux systems.

Characteristics of Bash

- Characteristics of **Bash**
  - **Alias** feature: nick name, an abbreviation of long command, kind of keyboard shortcut
  - **History** feature
    e.g. **`$ vim ~/.bash_history`**
  - **Calculation** feature
    e.g. **`$ expr 23 + 2`**
  - **Job Control** feature: selectively control jobs (process)
    e.g. **`$ jobs, $ fg, $ bg`**
  - **Autocorrection** feature: (Tab key)
  - **Command line edit** feature

Form of simple shell script

- **Form of simple shell statement**

```
(prompt) command [option…] [argument…]
```

- **Example**

```
$ ls -l
$ rm -rf ./mydir
$ sudo find . / -name “*.conf”
```

Environment variable

- Characteristics of **Bash**
  - Shell can recall multiple environment variables
  - The set environment variable can be checked by the command

```
$ echo $environment_variable
```

    e.g. **`$ echo $HOSTNAME`**

  - To change value of environment variable

```
$ export environment_variable=value
```

  - To verify the value of environment variable

```
$ printenv
```

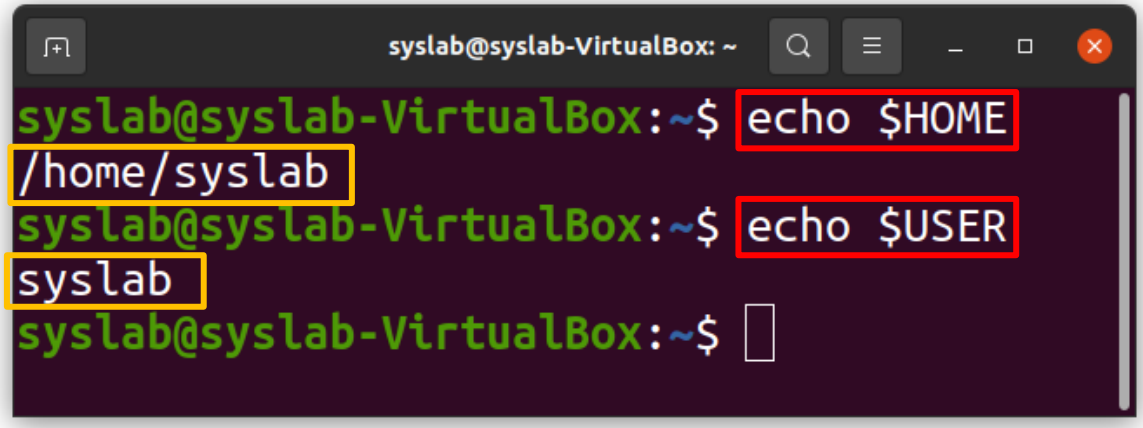
Common environment variables

variable	description	variable	description
HOME	Home directory of present user	PATH	search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
TERM	login terminal type	SHELL	Set path to login shell.
USER	The name of present user	PWD	present working directory
COLUMNS	column size of	LINES	total lines of present working terminal
PS1	your first prompt setting	PS2	your second prompt setting
BASH	full path to BASH shell	BASH_VERSION	version of current BASH shell
HISTFILE	name of the file in which command history is saved	HISTSIZE	maximum number of lines contained in the history file.
HOSTNAME	name of the your computer.	USERNAME	The name of present user

Common environment variables

❖ Example

variable	description	variable	description
HOME	Home directory of present user	USER	The name of present user



Shell script writing

Writing

- Alike other programming languages, shell script also support variables and various control statements(such as iteration, if else statement).
- Shell script runs directly in the shell, in the form of a text file without compiling separately.

Simple example of shell script

- Create a shell script file named '01name.sh' by vim:
  `$ vim 01name.sh`

```
1 #!/bin/sh
2 echo "printout user name: " $USER
3 echo "printout home directory " $HOME
4 exit 0
```

- row1: mandatory, a special form of a comment (**`#!`**), runs with bash shell
- row2: environment variable **`$USER`** will be printed out by **`echo`** command
- row4: return termination code. 0 means successful termination

First line in shell script

- **#!/bin/sh**
  - First line of shell script '**`#!bin/sh`**' means that the script file should always be run with bash
  - Usually **`/bin/sh`** points to the executable Bash file
    - Implemented as a symbolic link

```
1 #!/bin/sh
2 echo "printout user name: " $USER
3 echo "printout home directory " $HOME
4 exit 0
```

Running shell script: two methods

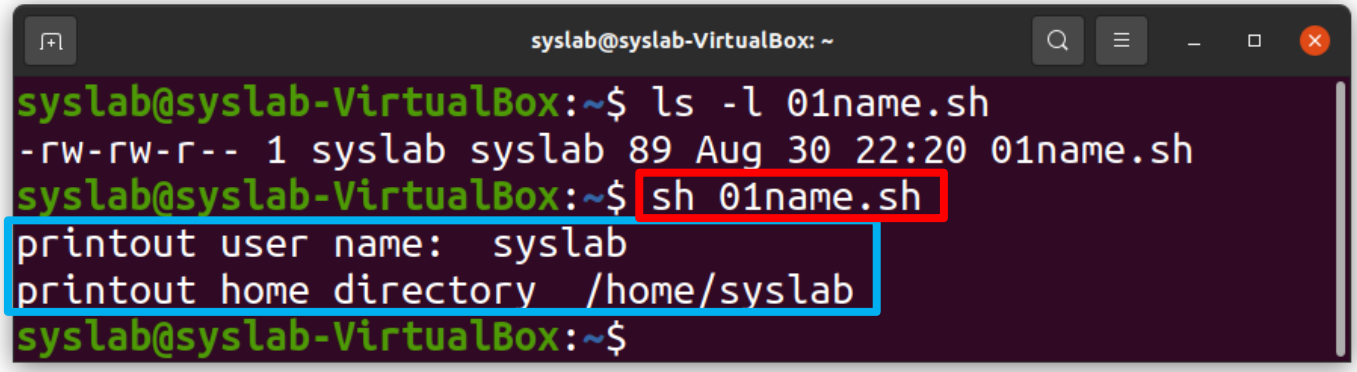
First method: sh

- use '**sh**' command to execute shell script
  - **No need to change property** of shell script when using 'sh' command

```
$ sh <file_name>
```

contents of '**01name.sh**'

```
1 #!/bin/sh
2 echo "printout user name: " $USER
3 echo "printout home directory " $HOME
4 exit 0
```



Running shell script: two methods

❖ Second method: ./

- use '**./**' to execute shell script (./ refers to current directory)
  - First, change **file property** to '**executable**'

```
$ chmod +x <file_name>.sh
```

  - Then execute file

```
$ ./<file_name>.sh
```

contents of '**01name.sh**'

```
1 #!/bin/sh
2 echo "printout user name: " $USER
3 echo "printout home directory " $HOME
4 exit 0
```

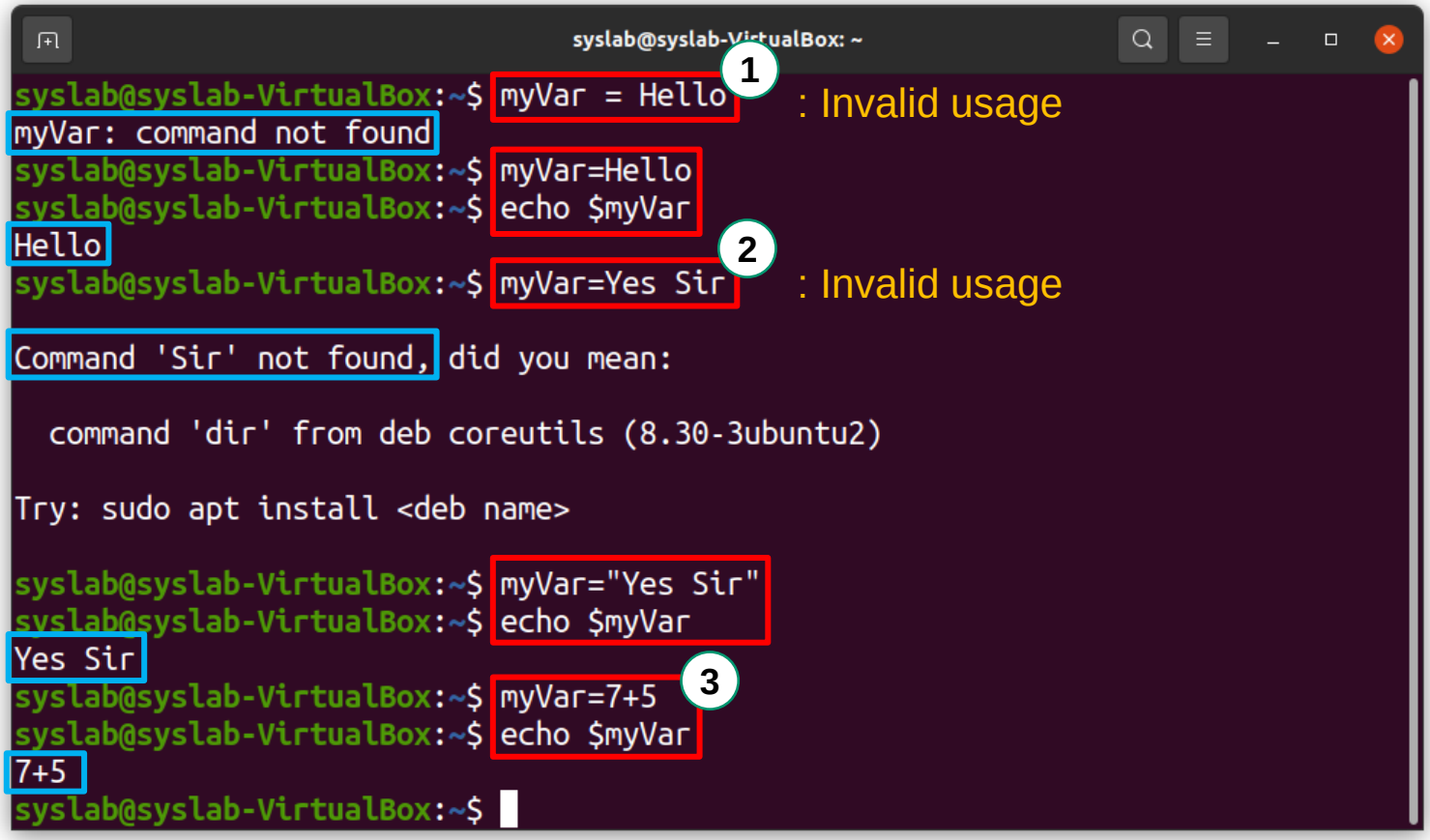
```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ ls -l 01name.sh
-rw-rw-r-- 1 syslab syslab 89 Aug 30 22:20 01name.sh
syslab@syslab-VirtualBox:~$ chmod +x 01name.sh
syslab@syslab-VirtualBox:~$ ls -l 01name.sh
-rwxrwxr-x 1 syslab syslab 89 Aug 30 22:20 01name.sh
syslab@syslab-VirtualBox:~$ ./01name.sh
printout user name:  syslab
printout home directory  /home/syslab
syslab@syslab-VirtualBox:~$
```

Variable

General outline of variable

- Variable stores value which can be changed
- Structure of shell script can be reused by only changing values stored in variable. With less effort or even no change of structure
- Variable is automatically created when it is first assigned
  - In the shell script, a variable doesn't have to be declared before its usage
- All values are considered as a 'string', even when the number is assigned to a variable
- Variable names are case sensitive
  - e.g. **`$aa $AA`** is different variable name
- No blank space between the '=' operator
  - e.g. **`myvar=Hello`** (valid usage)
    `myvar`✓**`=`**✓**`Hello`** (Invalid usage)

Variable example



1. First invalid usage: **No blank space** between the '=' operator
2. Second invalid usage: Value with blank space must be nested with " "
3. Third usage: 7+5 itself is assigned to variable (not calculated result of 7+5)

Variable

❖ Input and output of variable

- ‘ ’ or \ needed to print out a string with $.
- “ ” is optional for printing out a string, except for the case of the string with blank space

prints out `Hi sys`

prints out `myvar`

```
 1 #!/bin/sh
 2 myvar="Hi sys"
 3 echo $myvar
 4 echo "$myvar"
 5 echo '$myvar'
 6 echo \$myvar
 7 echo type somethind you want :
 8 read myvar
 9 echo '$myvar' = $myvar
10 exit 0
```

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 02var1.sh
Hi sys
Hi sys
$myvar
$myvar
type somethind you want :
:)
$myvar = :)
syslab@syslab-VirtualBox:~$
```

Variable

❖ Number calculation

- Use **expr** keyword to **calculate** variable values with operators such as +, -, *, /
- **expr** keyword must be tied with the formula and the ‘`’ (backquart)
  - ‘`’ (**backquart) is usally located on left side of [1/!] key on the keyboard**
- \ (backward slash) must be preceded to use parentheses in calculation
  - e.g. \(\)
- As an exception, to use * (the multiplication symbol) as a multiplication operator, \ must be preceded

Variable

❖ Number calculation example

```
1 #!/bin/sh
2 num1=100
3 num2=$num1+200
4 echo $num2
5 num3=`expr $num1 + 200`
6 echo $num3
7 num4=`expr \( $num1 + 200 \) / 10 \* 2`
8 echo $num4
9 exit 0
```

1. Expression **$num+200** is considered as a string
   - There must be **no blank space** e.g. **num2=$num1+200**
2. Expression **expr $num1 + 200** is considered as a calculation
   - **Blank space** is mandatory e.g. **num3=`expr $num1✓+✓200`**
3. For calculating * and using parentheses for calculation, \ must be preceded

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 03numcalc.sh
100+200
300
60
```

Variable

❖ Parameter variable

- Each part of the command being executed is specified as a variable in the form of **$0**, **$1**, **$2**, …
- Each of **$0**, **$1**, **$2**, … is a parameter variable
  - e.g. `# apt-get -y install gftp`
    `$0 $1 $2 $3`
- All parameter variables of command are expressed as **$***

```
1 #!/bin/sh
2 echo "name of this shell script is <$0>"
3 echo "first parameter is <$1> and second parameter is <$2>"
4 echo "all parameters are <$*>"
5 exit 0
```

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 04paravar.sh 12 3456
name of this shell script is <04paravar.sh>
first parameter is <12> and second parameter is <3456>
all parameters are <12 3456>
```

if statement, case statement

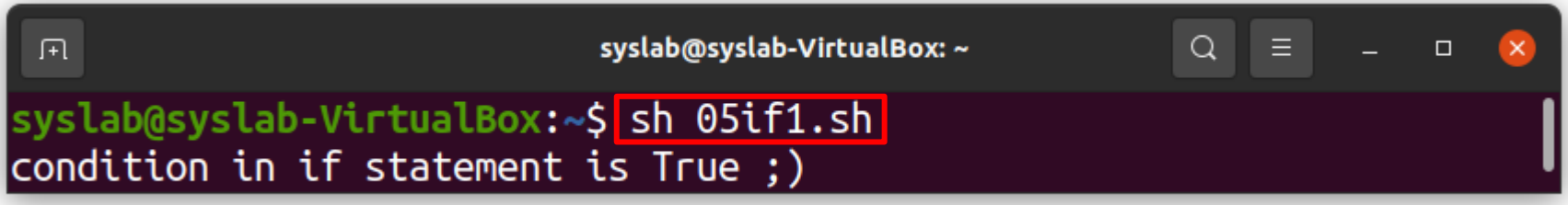
❖ Simple if statement

- Within the conditional statement, **blank spaces are needed**
  - e.g. `if`✓`[`✓`condition`✓`]`

- When the condition is true, if statement executes commands between **`then`** and **`fi`**

```
if [ condition ]
then
        -execute when TRUE-
fi
```

```
1 #!/bin/sh
2 if [ "sys" = "sys" ]
3 then
4         echo "condition in if statement is True ;)"
5 fi
6 exit 0
```



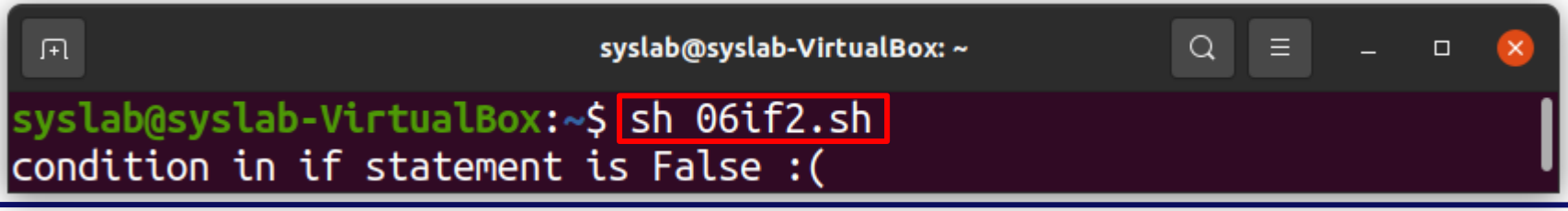
if statement, case statement

❖ if - else statement

- Executes by distinguishing between TRUE and FALSE

```
if [ condition ]
then
        -execute when TRUE-
else
        -execute when FALSE-
fi
```

```
1 #!/bin/sh
2 if [ "sys" != "sys" ]
3 then
4         echo "condition in if statement is True :)"
5 else
6         echo "condition in if statement is False :("
7 fi
8 exit 0
```



if statement, case statement

❖ Conditional statement: comparison operators

- Conditional statements allow string comparison and arithmetic comparison etc...

Integer Comparison operators	description
-eq	equal
-ne	not equal
-gt	greater than
-ge	greater than or equal to
-lt	less than
-le	less than or equal to

String Comparison operators	description
=	test for equality of strings
!=	test for inequality of strings
-n	test for empty string (if not empty than True)
-z	test for empty string (if empty than True)

07if3.sh

```
1 #!/bin/sh
2 if [ 100 -eq 200 ]
3 then
4         echo "100 and 200 got equal value \\\(^o^)/ "
5 else
6         echo "100 and 200 got different value \\\(^o^)/ "
7 fi
8 exit 0
```

```
if [ condition ]
then
        -execute when TRUE-
else
        -execute when FALSE-
fi
```

if statement, case statement

❖ File-related operators and Logical operators

File-related operators	description
-d	true when filename is a directory name
-e	true when file exist
-f	true when file is regular(ordinary) file
-w	true when file is writable
-r	true when file is readable
-x	true when file is executable
-s	true when file is not empty

Logical operators	description
-a or **&&**	logical AND
-o or **||**	logical OR
!	logical not

- There are a lot more Bash script-supported operators, but to keep this simple, the above operators would be sufficient

if statement, case statement

```
1 #!/bin/sh
2 fname=./rbtree.c
3 if [ -f $fname ]
4 then
5         echo "look! I found this! \(^.^)/ "
6         head -5 $fname
7 else
8         echo "there is no file named as '$fname' /(T.T)\ "
9 fi
10 exit 0
```

(2) (3)

(1) Copy a C file into current directory (./rbtree.c) and assign it into variable `fname`

(2) If the file stored in variable `fname` is an **ordinary file**(= true) then execute row 5

- if not(=false) execute row 7

(3) `head -5 $fname` **prints out first five lines in file**, which stored in variable `fname`

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 08if4.sh
there is no file named as './rbtree.c' /(T.T)\
syslab@syslab-VirtualBox:~$ cp linux-5.4.212/lib/rbtree.c .
syslab@syslab-VirtualBox:~$ sh 08if4.sh
look! I found this! \(^.^)/
// SPDX-License-Identifier: GPL-2.0-or-later
/*
  Red Black Trees
  (C) 1999  Andrea Arcangeli <andrea@suse.de>
  (C) 2002  David Woodhouse <dwmw2@infradead.org>
```

(1)

First 5 lines of rbtree.c

if statement, case statement

❖ case~esac statement

- Single if-else statement can only be used for distinguishing between T/F cases
- Script can become complicated if the number of cases is greater than three.
- In that case, case~esac statement will be useful

```
case expression in
        pattern1)
                statements;;
        pattern2)
                statements;;
        ...
esac
```

if statement, case statement

❖ case - esac statement example

```
1  #!/bin/sh
2  case "$1" in
3          start)
4                  echo "starting";;
5          stop)
6                  echo "halt";;
7          restart)
8                  echo "restarting";;
9          *)
10                 echo "idk... :(";;
11 esac
12 exit 0
```

1. According to the first parameter variable($1) added in execution time, the execution branches into 3, 5, 7, 9 rows
2. In the end of each cases, **double semicolon(;;)** is needed to **stop case statement**
3. Termination of case statement

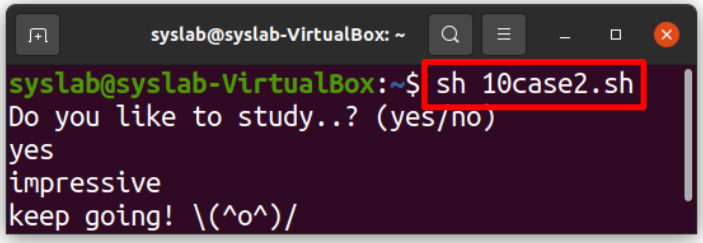
```
syslab@syslab-VirtualBox:~$ sh 09case1.sh start
starting
syslab@syslab-VirtualBox:~$ sh 09case1.sh stop
halt
syslab@syslab-VirtualBox:~$ sh 09case1.sh restart
restarting
syslab@syslab-VirtualBox:~$ sh 09case1.sh 123
idk... :(
syslab@syslab-VirtualBox:~$ sh 09case1.sh
idk... :(
```

```
case expression in
        pattern1)
                statements;;
        pattern2)
                statements;;
        ...
esac
```

if statement, case statement

```
1  #!/bin/sh
2  echo "Do you like to study..? (yes/no)"
3  read answer
4  case $answer in
5       yes | y | Y | Yes | YES)
6               echo "impressive"
7               echo "keep going! \(^o^)/ ";;
8       [nN]*)
9               echo "hope you like this too.. /(ToT)\ ";;
10         *)
11              echo "answer with yes or no please"
12              exit 1;;
13 esac
14 exit 0
```

(1) (2) (3) (4) (5)



1. **`read answer`**: receive value and store into variable '**`answer`**'
2. If the input value (which is stored in '**`answer`**') is **`yes`**, **`y`**, **`Y`**, **`Yes`**, **`YES`** then execute row 6, 7
3. **More rows to be executed** in this case, so there is **no need for the double semicolon**
4. **`[nN]*)`** means that all input values starting with character n or N are allowed for this case
5. Abnormal termination. exit with code 1

if statement, case statement

- **if statement with relational operator example**
  - Single semicolon which lies between (**`if [~] && [~]`**) and (**`then`**) separates the statement into independent rows

```
1 #!/bin/sh
2 echo "type the file name, which you want to check"
3 read fname
4 if [ -f $fname ] && [ -s $fname ] ; then
5         head -5 $fname
6 else
7         echo "there is no such file or file is empty."
8 fi
9 exit 0
```

```
syslab@syslab-VirtualBox:~$ sh 11andor.sh
type the file name, which you want to check
rbtree.c
// SPDX-License-Identifier: GPL-2.0-or-later
/*
  Red Black Trees
  (C) 1999  Andrea Arcangeli <andrea@suse.de>
  (C) 2002  David Woodhouse <dwmw2@infradead.org>
syslab@syslab-VirtualBox:~$ touch 11zero_size
syslab@syslab-VirtualBox:~$ sh 11andor.sh
type the file name, which you want to check
11zero_size
there is no such file or file is empty.
```

Iteration statement

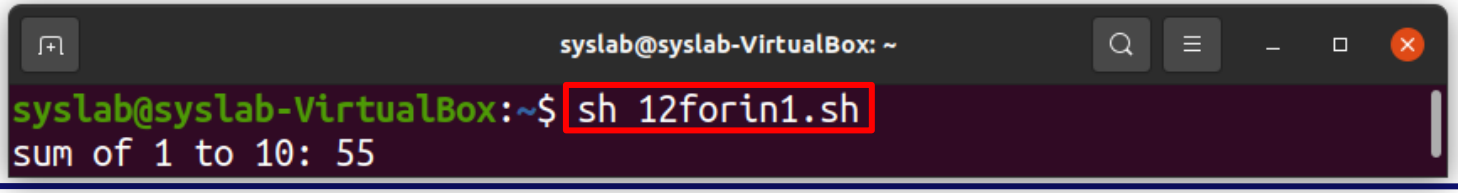
❖ for - in statement

- Put each value into the variable and run the statement to repeat after the keyword '***do***'
  - Repeats as many times as the number of values

```
for variable in value1 value2 value3 ...
do
        -statement to repeat-
done
```

```
1 #!/bin/sh
2 sum=0
3 for i in 1 2 3 4 5 6 7 8 9 10
4 do
5         sum=`expr $sum + $i`
6 done
7 echo "sum of 1 to 10: "$sum
8 exit 0
```

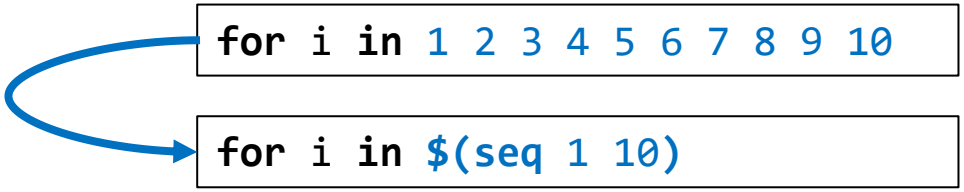
1. Execute row5 for 10 times with variable **i** (1 ~ 10)
2. Cumulate value of variable '**i**' to variable '**sum**'



Iteration statement

❖ for - in statement

- More simple way to iterate.



  - See **`$ man seq`** for detailed info
  - Test with sample script file '12forin1.sh' commented line.

```
1 #!/bin/sh
2 sum=0
3 for i in $(seq 1 10)
4 do
5         sum=`expr $sum + $i`
6 done
7 echo "sum of 1 to 10: "$sum
8 exit 0
```

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 12forin1.sh
sum of 1 to 10: 55
```

Iteration statement

- **Print out shell script file name in the current working directory and first three lines of each file**

```
1 #!/bin/sh
2 for fname in $(ls *.sh)
3 do
4         echo "-------$fname-------"
5         head -3 $fname
6 done
7 exit 0
```

1. Run rows 4 to 5 with the execution results of the **`ls *.sh`** command one by one in the `fname` variable
2. Print out file name
3. Print out first three lines of file

```
syslab@syslab-VirtualBox:~$ sh 13forin2.sh
-------01name.sh-------
#!/bin/sh
echo "printout user name: " $USER
echo "printout home directory " $HOME
-------02var1.sh-------
#!/bin/sh
myvar="Hi sys"
echo $myvar
-------03numcalc.sh-------
#!/bin/sh
num1=100
num2=$num1+200
```

Iteration statement

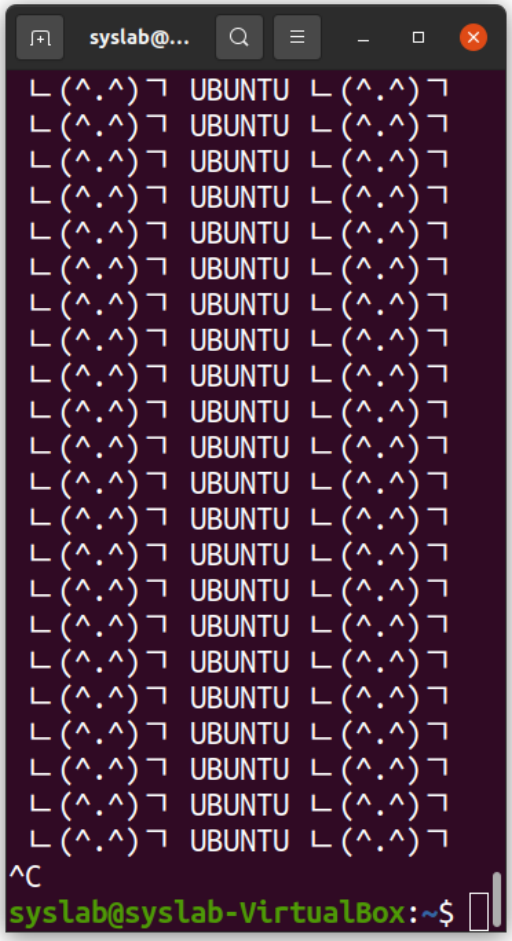
❖ while statement

- The while statement repeats execution until the conditional expression becomes false

14while1.sh

```
1 #!/bin/sh
2 while [ 1 ]
3 do
4         echo " ㄴ(^.^)ㄱ UBUNTU ㄴ(^.^)ㄱ "
5 done
6 exit 0
```

① **Always true** when **[1]** or **[:]** comes in the conditional position, so repeat row4 **infinitely**



- **ctrl + c** to stop the process

Iteration statement

❖ Example 1)

- Print out cumulative total from 1 to 10

```
1 #!/bin/sh
2 sum=0
3 i=1
4 while [ $i -le 10 ]
5 do
6         sum=`expr $sum + $i`
7         i=`expr $i + 1`
8 done
9 echo "sum of 1 to 10: "$sum
10 exit 0
```

(1) If **i** is less or equal than 10, execute rows6~7

(2) Cumulate values of **i** and store it into variable **sum**

(3) Increase the value of variable **i** by 1

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 15while2.sh
sum of 1 to 10: 55
```

Iteration statement

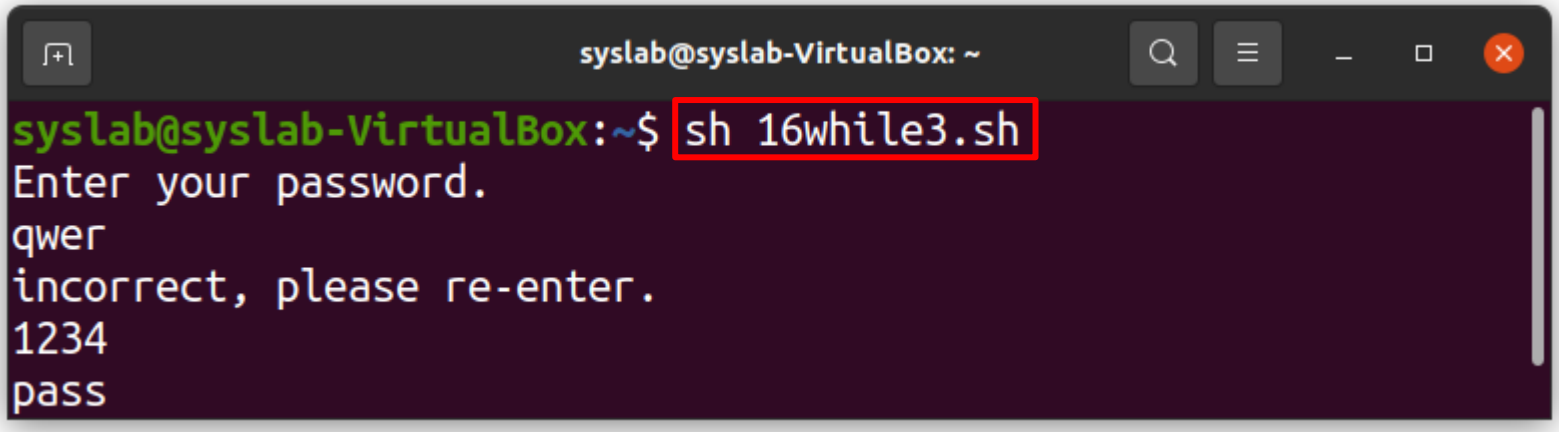
❖ Example 2)

- Password verification

```
 1 #!/bin/sh
 2 echo "Enter your password."
 3 read mypass
 4 while [ $mypass != "1234" ]
 5 do
 6         echo "incorrect, please re-enter."
 7         read mypass
 8 done
 9 echo "pass"
10 exit 0
```

① If the value of **mypass** variable is not '**1234**', run rows 6 to 7, and if it is '**1234**', exit the **while** statement

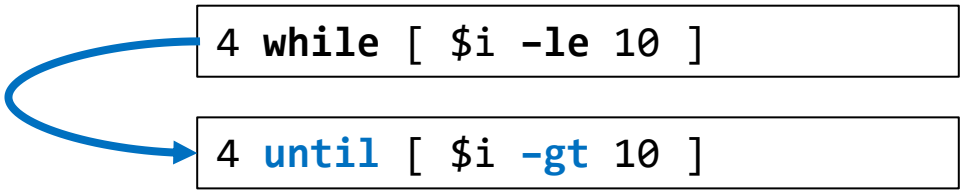
② Receive a value for the variable **mypass**, again



Iteration statement

until statement

- **Repeat until** the conditional expression is **true** (= while false) with almost the same purpose as the while statement
- To implement '`15while2.sh`' by using **`until`** statement, change row4



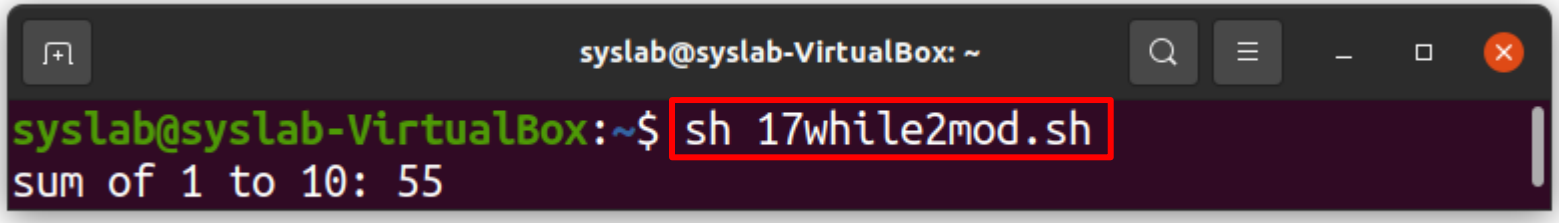
break, continue, exit, return statements

- **`break` statement**: usually used to terminate iteration
- **`continue` statement**: return to the conditional expression of the repeating statement
- **`exit` statement**: completely shutdowns program
- **`return` statement**: available within a function, returns the function to where it was called

Iteration statement

- **Example 3)**
  - until statement

```
 1 #!/bin/sh
 2 # modification of 15while2.sh
 3 # change while statement into until statement
 4 sum=0
 5 i=1
 6 until [ $i -gt 10 ]
 7 do
 8         sum=`expr $sum + $i`
 9         i=`expr $i + 1`
10 done
11 echo "sum of 1 to 10: "$sum
12 exit 0
```



Iteration statement

- **Example 4)**
  - iteration statement with case – esac statement

```
 1 #!/bin/sh
 2 echo "starting infinite loop. (b: break, c: continue, e: exit)"
 3 while [ 1 ] ; do
 4         read input
 5         case $input in
 6                 b | B)
 7                         break;;
 8                 c | C)
 9                         echo "continue: back to condition of while statement"
10                         continue;;
11                 e | E)
12                         echo "exit: terminate p/g(function)"
13                         exit 1;;
14         esac;
15 done
16 echo "break: escape from while loop."
17 exit 0
```

Iteration statement

❖ Example 4)

- Iteration statement with case – esac statement
  - row3: infinite loop
  - row5: branch according to values received in row 4
  - row6~7: if input value is b | B, execute **`break`** in row 7
    - ✓ terminate running **`while`** statement and execute row 16
  - row8~10: if input value is c | C, execute row9. Then in row 10, return to the condition of **`while`** statement in row3, **`[ 1 ]`**
  - row11~13: if input value is e | E, execute row12 and terminate program itself in row13

```
syslab@syslab-VirtualBox:~$ sh 18bce.sh
starting infinite loop. (b: break, c: continue, e: exit)
c
continue: back to condition of while statement
e
exit: terminate p/g(function)
syslab@syslab-VirtualBox:~$ sh 18bce.sh
starting infinite loop. (b: break, c: continue, e: exit)
b
break: escape from while loop.
```

Shell script application

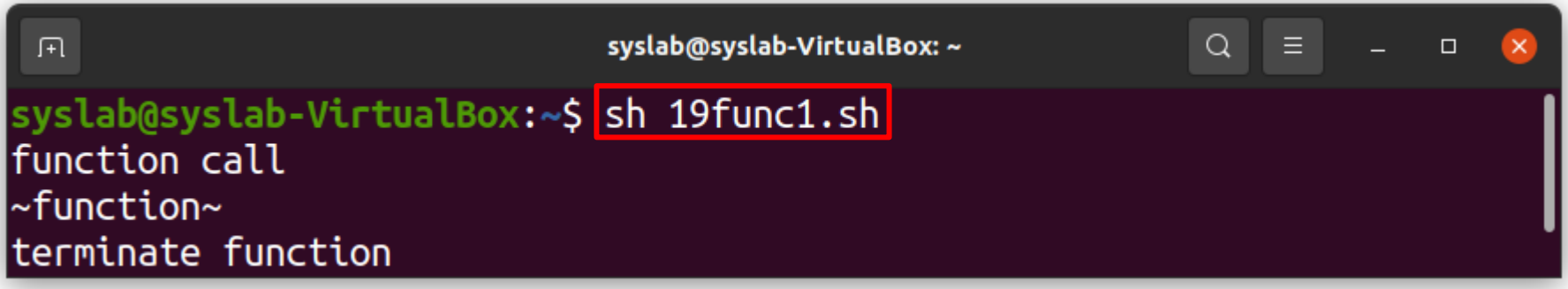
❖ User-defined function

- Function which is provided by the user of the program

```
1 #!/bin/sh
2 myFunction () {
3         echo "~function~"
4         return
5 }
6 echo "function call"
7 myFunction
8 echo "terminate function"
9 exit 0
```

```
function_name () {
        -contents-
}
...
function_name
```

- row2~5: define the function
- row7: call function by only using function name



Shell script application

❖ Usage of function parameters

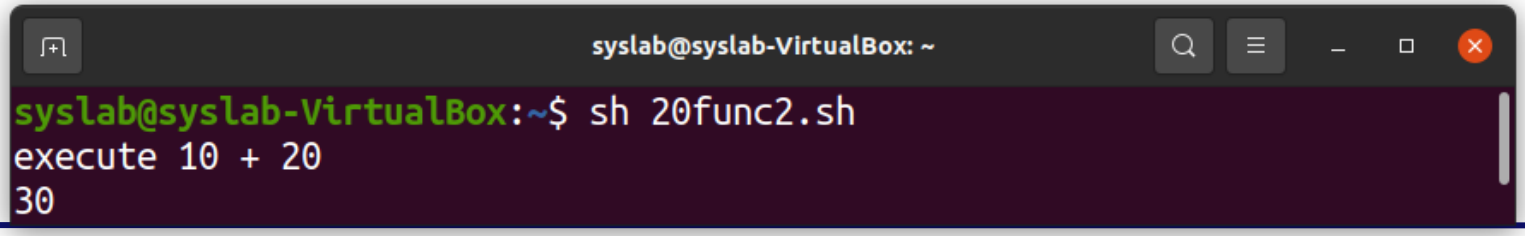
- To use the parameter (factor) of a function, attach the parameter after the function is called
- In function, use **$1**, **$2**, …

```
1 #!/bin/sh
2 sum () {
3         echo `expr $1 + $2`
4 }
5 echo "execute 10 + 20"
6 sum 10 20
7 exit 0
```

(1)

```
function_name () {
        -use $1, $2, etc...-
}
...
function_name parameter1 parameter2 ...
```

(1) To pass parameters to function when calling function, separate parameters by blank space. Then it will insert them one by one



Shell script application

❖ eval

- Execute a text by recognizing it as a statement

```
1 #!/bin/sh
2 str="ls -l 21eval.sh"
3 echo $str    (1)
4 eval $str    (2)
5 exit 0
```

(1) Literally print out '**ls -l eval.sh**' which is value of variable **`str`**

(2) Execute '**`ls -l eval.sh`**' which is recognized as a command

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 21eval.sh
ls -l 21eval.sh
-rw-rw-r-- 1 syslab syslab 59 Sep 13 16:19 21eval.sh
```

Shell script application

❖ export

- Make a specific variable a global variable and use it across the whole shell session

exp1.sh

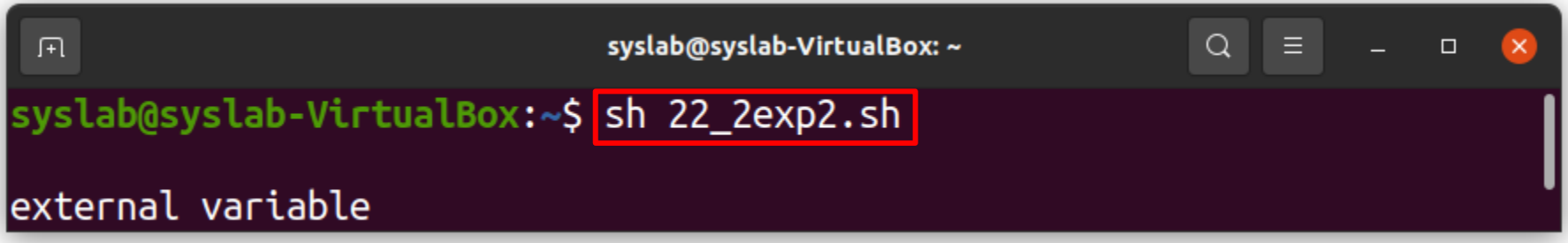
```
  1 #!/bin/sh
  2 echo $var1
  3 echo $var2
  4 exit 0
"22_1exp1.sh" 4 lines --100%--
```

- exp1.sh row2~3: print out var1 and var2

exp2.sh

```
  1 #!/bin/sh
  2 var1="local variable"
  3 export var2="external variable"
  4 sh 22_1exp1.sh
  5 exit 0
"22_2exp2.sh" 5 lines --100%--
```

- exp2.sh row2: var1 is local variable.
  so, used only in exp2.sh
- exp2.sh row3: declare var2 as global variable (external variable) and assign value in var2
- exp2.sh row4: execute 'exp1.sh'



Shell script application

❖ printf

- Print out by using a similar format with the printf( ) function in C

```
1 #!/bin/sh
2 var1=200.5
3 var2='shell script! \(^o^)/ '
4 printf "%5.2f \n\n \t %s \n" $var1 "$var2"
5 exit 0
```

(1) (2)

(1) **%5.2f** is a format specifier which means to print out a total of five digits and up to two decimal places
  - ✓ **\n**: New line
  - ✓ **\t**: Tab character
  - ✓ **%s**: Print out a string

(2) Value of **$var2** has blank spaces, so the variable must be wrapped up with “ ” to avoid an error

```
syslab@syslab-VirtualBox:~$ sh 23printf.sh
200.50

        shell script! \(^o^)/
```

Shell script application

- **set, $(command)**
  - Use '**$**' format to use Linux commands as a result
  - Use **set** command to use results as parameters

```
1 #!/bin/sh
2 echo "Today's date: $(date)"
3 set $(date)
4 echo "It is $5 $6 now."
5 exit 0
```

(1) 1 **$(date)** is a result of **date** command

(2) 2 The result of **$** are stored into parameter **$1**, **$2**, **$3**, ...

- row4: print out time which is fifth~sixth parameter

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 24set.sh
Today's date: Tue 13 Sep 2022 04:53:08 PM KST
It is 04:53:08 PM now.
```

$0

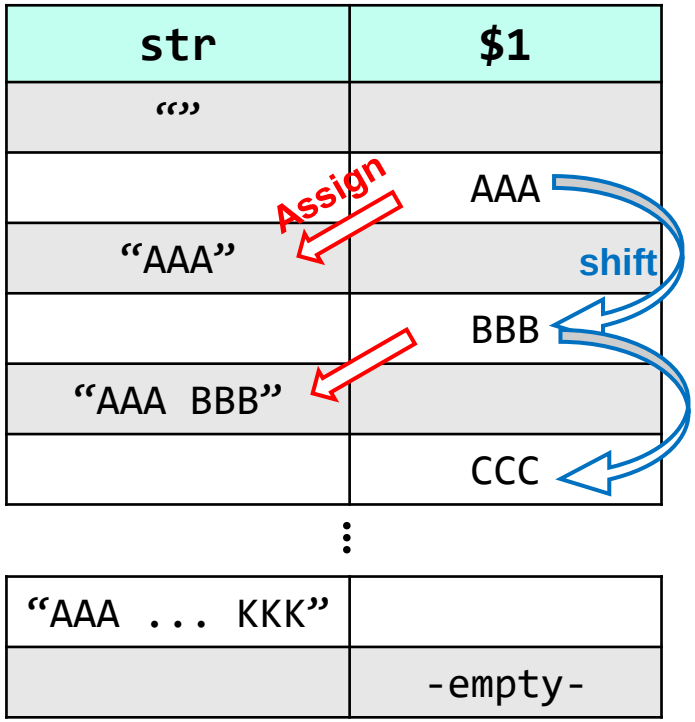
$1 $2 $3 $4 $5 $6 $7

Shell script application

❖ shift

- Left shift parameter variables

```
1 #!/bin/sh
2 myfunc () {
3         str=""
4         while [ "$1" != "" ]; do
5                 str="$str $1"
6                 shift
7         done
8         echo $str
9 }
10 myfunc AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
11 exit 0 $1 $2 $3 ...
```



- row3: initialize **str** variable to cumulate result
- row4: Run repeatedly, while the **$1** parameter is not empty
- row5: assign **$1** value into variable **str**

(1) left shift each of all parameters (e.g. $2 now becomes $1, $3 becomes $2, etc...)

- row8: after the `while` statement, print out accumulated value in variable **str**

```
syslab@syslab-VirtualBox: ~
syslab@syslab-VirtualBox:~$ sh 25shift.sh
AAA BBB CCC DDD EEE FFF GGG HHH III JJJ KKK
 $1  $2  $3  ...
```

End of the content

- **For more detailed(complex) shell script examples and descriptions.**
  - https://bash.cyberciti.biz/guide/Main_Page