SSE2033

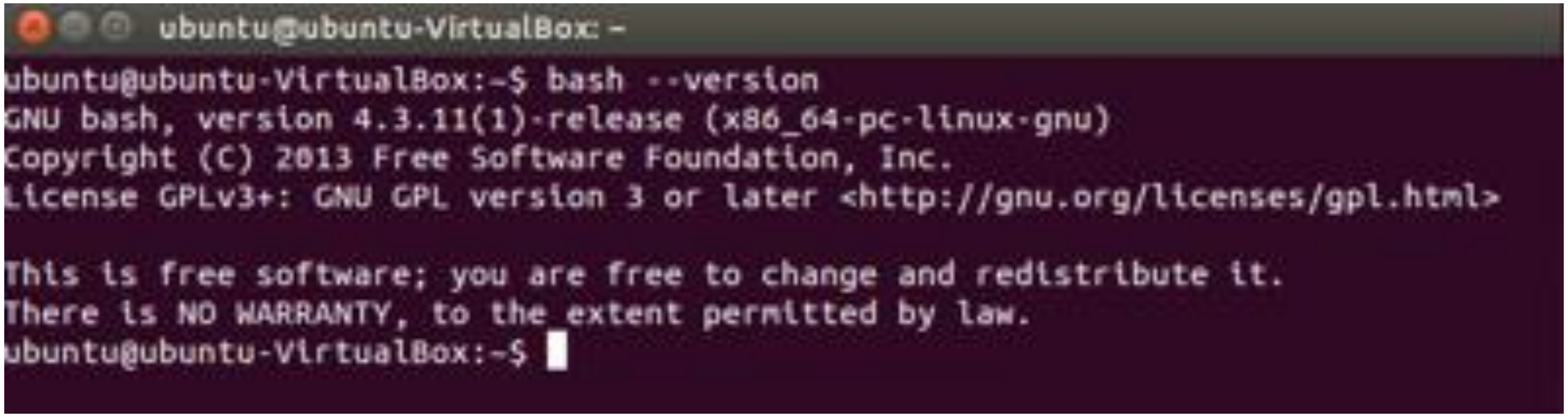
SUNG KYUN KWAN UNIVERSITY

Shell

Prof. Jinkyu Jeong (jinkyu@skku.edu)
TA – Gyusun Lee (gyusun.lee@csi.skku.edu)
TA – Jiwon Woo (jiwon.woo@csi.skku.edu)
Computer Systems and Intelligence Laboratory (http://csi.skku.edu)
Sung Kyun Kwan University

What is Shell?

- **Interface program between user & UNIX(Linux)**
  - Similar with Windows command prompt
  - Bash version: $/bin/bash –version
- **Shell takes role of,**
  - Control commands(Unix)
  - Advanced programming language



```
ubuntu@ubuntu-VirtualBox:~$ bash --version
GNU bash, version 4.3.11(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
ubuntu@ubuntu-VirtualBox:~$
```

Redirection & Pipes

- **File descriptor**
  - Use for process to access file or device
  - Standard file descriptor
    - stdin(0): Standard input (e.g., keyboard)
    - stdout(1): Standard output (e.g., terminal)
    - stderr(2): Standard error output (e.g., error message)
- **Redirection**
  - Input/Output redirection
  - >(overwrite), >>(append)
  - <(stdin)
- **Pipe**
  - Connect process (propagate stdout to stdin)
  - e.g. cat test.txt | grep “a”

Redirection & Pipes Practice

- **ls & ps command**

```
$ls –al > output.txt        (redirect result of “ls –al” to
output.txt)
$cat output.txt
$ps >> output.txt           (redirect result of “ps” to output.txt)
$cat output.txt
```

- **Result**

```
total 4
drwxr-xr-x   2 root root   24 Dec  3 01:30 .
dr-xr-x---. 10 root root 4096 Dec  3 01:30 ..
-rw-r--r--   1 root root    0 Dec  3 01:30 output.txt
   PID TTY          TIME CMD
 10679 pts/1    00:00:00 sudo
 10680 pts/1    00:00:00 su
 10681 pts/1    00:00:01 bash
 24822 pts/1    00:00:00 ps
```

Redirection & Pipes Practice (cont'd)

- **Search for files**
  - $find [search directory] –name [file name]
  - $find . –name "*.txt"
- **Cut**
  - $cut –d '[delimiter]' –f[field]
- **Pipe**
  - $cat "*.txt" | cut –d' ' –f1 | sort

    – Refer to ***man*** for more details about ***find*** and ***cut***

Basic Unix Commands

Command	Description	Command	Description
echo “some text”	Print out “some text” on terminal	read [var]	Read input and set var
wc –l [file]	Number of lines in file	Sort [file]	Sort file line by line
cp [srcfile] [dstfile]	Copy srcfile to dstfile	Uniq [file]	Remove same line in file
mv [oldname] [newname]	Rename or move of file	Basename [file]	Name of file (not directory)
rm [file]	Remove file	Dirname [file]	Directory of file (not name of file)
grep [pattern] [file]	File pattern in file	Head –n 2 [file]	Print out first 2 lines of file
cat [file]	Standard output of file	Tail –n 2 [file]	Print out last 2 lines of file
file [somefile]	File type of somefile	sed, awk	Control of string stream

For more details, refer to man page of each command (e.g., $man sed)

Shell Programming

- **Two ways of shell programming**
  - Line command
  - Execute a shell script

```
$for file in *
>do
>if grep –l ps $file
>then
>more $file
>fi
>done
```

Ex1) Line command

```
$cat test.sh
#!/bin/bash
for file in *
do
    if grep –l ps $file
    then
        cat $file
    fi
done
$bash test.sh
```

Ex2) Execute a shell script

Shell Scripts

- **Basically, a shell script is a text file with Unix commands in it**
- **Shell scripts usually begin with a #! and a shell name**
  - For example: #!/bin/bash
  - If they do not, the user's current shell will be used
- **Any Unix command can go in a shell script**
  - Commands are executed in order or in the flow determined by control statements
- **Different shells have different control structures**
- **To make script as executable file,**
  - $chmod +x [filename]

Grammars of Shell

- **Variable**
  - String, number, environment, parameter
- **Condition**
  - Boolean
- **Condition control**
  - if, elif, for, while, until, case
- **List**
- **Function**
- **Reserved command**

Shell Variable

- Declaration = Initialization of variable
- Case sensitive
- Dereference shell variable by “$”
- Can check value of variable by “echo” command

```
$VAR=TEST
$echo $VAR
Hello
$VAR=“Hello World”
$echo $VAR
Hello World
$read VAR
Best TA ever!
$echo $VAR
Best TA ever!
```

No space!

Shell Environment Variables

- Some variables are initialized at start of execution
- Name of environment variables are upper case
- Different user environment has different value

Environment Variable	Description
$HOME	Home directory of current user
$PATH	Directory for search command, divided by “:”
$LD_LIBRARY_PATH	Directory for search library, divided by “:”
$0	Name of shell script (bash by default)
$#	Number of passed parameter
$$	Process ID of shell script

Shell Parameter Variable

- **If shell script run with parameter variables,**
  - We can access to parameter as $1, $2, … in script

```
$cat test.sh
#!/bin/bash
echo $1
echo $2
echo $3
$./test.sh I am Groot
I
am
Groot
```

Shell Conditional Statement

- **If**
  - Check condition and executes command block

```
if condition
then
   statement1
   …
else
   statement1
      …
fi
```

```
#!/bin/bash

read num
if [ $num –lt 5 ]
then
   echo “Lower than 5”
else
  echo “Higher than 5”
fi
```

Shell Conditional Statement

- **elif**
  - Same as *else if*

```
#!/bin/bash

read num
if [ $num –lt 5 ]
then
    echo “Lower than 5”
elif [ $num –gt 8 ]
then
    echo “Greater than 8”
else
    echo “5~8”
fi
```

Conditions

- **File test**
  - -e: True if file exists
  - -d: True if file exists and is a directory
  - Usage: if [-e file.txt]
- **String test**
  - =, !=, <, >
  - Usage: if [<STRING1> != <STRING2>]
- **Arithmetic test**
  - -eq(equal), -ne(not equal), -le(less or equal than), -ge(greater or equal than), -lt(less than), -gt(greater than)
  - Usage: if [<INTEGER1> -eq <INTEGER2>]

For-loop

- **For**
  - Iterate for range of values
  - Range of values can be set of *strings*

```
for variable in values
do
   statements
done
```

```
#!/bin/bash

for x in a b c d e
do
   echo $x
done
```

```
a
b
c
d
e
```

While-loop

- For-loop is hard to use for fixed number of iteration
- While-loop

```
while condition
do
    statements
done
```

```
#!/bin/bash
for x in 1 2 3 4 5 6 7 8 9 10 11 12
do
    echo $x
done
```

```
#!/bin/bash
x=1
while [ $x –le 12 ]
do
    echo $x
    ((x++))       same as x=$(($x+1))
done
```

Until-loop

- **Until-loop iterates statements until condition becomes true**

```
until condition
do
    statements
done
```

```
#!/bin/bash
x=1
while [ $x –le 12 ]
do
    echo $x
    ((x++))
done
```

```
#!/bin/bash
x=1
until [ $x –gt 12 ]
do
    echo $x
    ((x++))
done
```

Case Statement

Usage of case statement

```
case variable in
    pattern [|pattern] …) statements;;
    pattern [|pattern] …) statements;;
    …
esac
```

```
#!/bin/bash
case “$input” in
   yes|y|Yes|YES) echo “YES!”;;
   [nN]*) echo “NO!”;;
   *) echo “bad input”;;
esac
```

```
y
YES!
```

```
N
NO!
```

```
Apple
bad input
```

Reference

- https://wiki.kldp.org/wiki.php/DocbookSgml/Shell_Programming-TRANS
- http://www.softintegration.com/docs/ch/shell/

[Lab - Practice #1]

- **Bash calculator**
  - Write a bash program for calculate inputs
  - Maximum # of inputs: 5

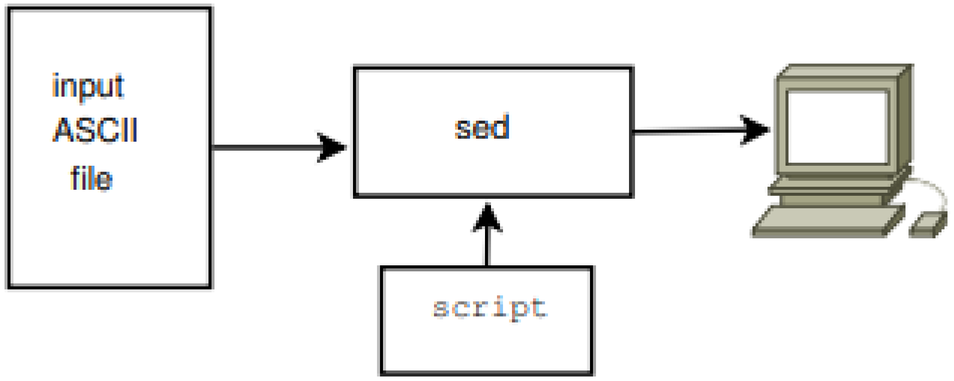
```
$ ./lab1.sh 3 + 5 * 2
3 + 5 * 2 = 13
```

- **Tips**
  - $1=3, $2=+, $3=5, $4=*, $5=2 for above example

Shell cont'

sed: Stream-oriented, Text Editor

- **Look for patterns one line at a time, like grep**
- ***Change* lines of the file**
- **sed has three options**
  - -e: script is on the command line (default)
  - -f: finds all rules that are applied in a specific (script) file
  - -n: suppresses the output



Invoking sed

- $sed –e ‘address command’ inputfile
- $sed –f script.sed inputfile
- Each instructions given to *sed* consists of an address and command
- Sample sed-script file:

```
#This line is a comment
2,14 s/A/B/
30d
40d
```

1. From lines 2 to 14, substitute the character A with B
2. Line 30 – delete
3. Line 40 – delete

Usage of sed

- $sed ‘s/[pattern to erase]/[pattern to add]/g’

```
$cat test.txt
total 4
drwxr-xr-x      2    root root 24      Dec  3   01:30    .
dr-xr-x---      10   root root 4096    Dec  3   01:30    ..
-rw-r--r--      1    root root 0       Dec  3   01:30    output.txt
$cat test.txt | sed ‘s/[0-9]//g’
total
drwxr-xr-x      root root         Dec       :    .
dr-xr-x---      root root         Dec       :    ..
-rw-r--r--      root root         Dec       :    output.txt
$cat test.txt | sed ‘s/$/>>>/g’ | sed ‘s/^/<<</g’
<<<drwxr-xr-x   2    root root 24      Dec  3   01:30    .>>>
<<<dr-xr-x---   10   root root 4096    Dec  3   01:30    ..>>>
<<<-rw-r--r--   1    root root 0       Dec  3   01:30    output.txt>>>
```

Entire-Pattern & Numbered-Buffer

- &: designates the entire pattern (just matched)
- \(and \): designate a numbered pattern later on identified by its respective number-id such as: \1, \2, \3, etc.

$$\mathrm{s}/\overset{\&}{\boxed{-----}}/---\&----/$$

$$\mathrm{s}/\overset{\backslash 1}{\boxed{\backslash(---\backslash)}}\overset{\backslash 2}{\boxed{\backslash(----\backslash)}}\overset{\backslash 3}{\boxed{\backslash(--\backslash)}}/---\backslash 1----\backslash 2--\backslash 3---/$$

Examples (1)

```
$cat example.txt
6793304567
6793304568
6793304569
$cat example.txt | sed ‘s/\([0-9]\{4\}\)\([0-9]\{2\}\)\([0-9]\{4\}\)/\1-\2--\3---/’
6793-30--4567---
6793-30--4568---
6793-30--4569---
$cat example.txt | sed ‘s/\([0-9]\{4\}\)\([0-9]\{2\}\)\([0-9]\{4\}\)/--\1-\2--\3---/’
--6793-30--4567---
--6793-30--4568---
--6793-30--4569---
```

Examples (1)

```
$cat example.txt
6793304567
6793304568
6793304569
$cat example.txt | sed ‘s/([0-9]{4})([0-9]{2})([0-9]{4})/1-2--3---/’
6793-30--4567---
6793-30--4568---
6793-30--4569---
$cat example.txt | sed ‘s/([0-9]{4})([0-9]{2})([0-9]{4})/--1-2--3---/’
--6793-30--4567---
--6793-30--4568---
--6793-30--4569---
```

Examples (2)

```
$cat example.txt
6793304567
6793304568
6793304569
$cat example.txt | sed ‘s/[0-9]\{4\}/&%/’
6793%304567
6793%304568
6793%304569
$cat example.txt | sed ‘s/[0-9]\{4\}/&%/2’
67933045%67
67933045%68
67933045%69
$cat example.txt | sed ‘s/[0-9]\{4\}/&%/g’
6793%3045%67
6793%3045%68
6793%3045%69
```

Examples (3)

```
$cat example.txt
I had a black dog, a white dog, a yellow dog and
a fine white cat and a pink cat as well as a croc.
These are my animals: dogs, cats and a croc.
$cat example.txt | sed ‘1 s/dog/DOG/g’
I had a black DOG, a white DOG, a yellow DOG and
a fine white cat and a pink cat as well as a croc.
These are my animals: dogs, cats and a croc.
$cat example.txt | sed ‘1,3 s/dog/DOG/1’
I had a black DOG, a white dog, a yellow dog and
a fine white cat and a pink cat as well as a croc.
These are my animals: DOGs, cats and a croc.
$cat example.txt | sed ‘s/dog/DOG/g’
I had a black DOG, a white DOG, a yellow DOG and
a fine white cat and a pink cat as well as a croc.
These are my animals: DOGs, cats and a croc.
```

Transforming Characters (option y)

```
$cat example.txt
I had a black dog, a white dog, a yellow dog and
a fine white cat and a pink cat as well as a croc.
These are my animals: dogs, cats and a croc.
$cat example.txt | sed ‘1 y/abcdt/ADCBQ’
I hAB A DlACk Bog , A whiQe Bog , A yellow Bog AnB
A fine whiQe CAQ AnB A pink CAQ As well As A CroC .
These Are my AnimAls : Bogs , CAQs AnB A CroC .
```

For the additional options and functionalities, please refer to *man* page ($man sed)

awk: Pattern Scanning and Processing

- **awk's purpose**
  - A general purpose programmable filter that handles text (strings) as easily as numbers
- **Scans text files line-by-line and searches for patterns**
- **Works in a way similar to *sed* but it is more versatile**

awk Invocation

- **$awk –f [awk script] [input file]**
- **$awk ‘{awk-commands}’ [input file]**

```
$cat example.txt
total 4
drwxr-xr-x 2   root    root    24      Dec 3   01:30   .
dr-xr-x--- 10  root    root    4096    Dec 3   01:30   ..
-rw-r--r-- 1   root    root    0       Dec 3   01:30   output.txt
$awk ‘{print $3 $2}’ example.txt
4
root2
root10
root1
```

Example

```
$cat example.txt
total 4
drwxr-xr-x 2   root    root    24      Dec 3   01:30   .
dr-xr-x--- 10  root    root    4096    Dec 3   01:30   ..
-rw-r--r-- 1   root    root    0       Dec 3   01:30   output.txt
$cat example.txt | awk ‘{print $1 > temp1.txt; print $2 > temp2.txt}’
$cat temp1.txt
total
drwxr-xr-x
dr-xr-x---
-rw-r--r–
$cat temp2.txt
4
2
10
1
```