

Introduction to Linux Basics

Part-II Shell Scripting

Georgia Advanced Computing Resource Center

University of Georgia

Zhuofei Hou, HPC Trainer

zhuofei@uga.edu

Outline

- What is GACRC?
- What are Linux Shell and Shell Scripting?
- Shell Scripting Syntax Basics
- Real Shell Scripting Examples

What is GACRC?

Who Are We?

- Georgia **A**dvanced **C**omputing **R**esource **C**enter
- Collaboration between the Office of Vice President for Research (**OVPR**) and the Office of the Vice President for Information Technology (**OVPIIT**)
- Guided by a faculty advisory committee (GACRC-AC)

Why Are We Here?

- To provide computing hardware and network infrastructure in support of high-performance computing (**HPC**) at UGA

Where Are We?

- <http://gacrc.uga.edu> (Web) <http://wiki.gacrc.uga.edu> (Wiki)
- <http://gacrc.uga.edu/help/> (Web Help)
- https://wiki.gacrc.uga.edu/wiki/Getting_Help (Wiki Help)

GACRC Users September 2015

| Colleges & Schools | Depts | PIs | Users |
|--|-------|-----|-------|
| Franklin College of Arts and Sciences | 14 | 117 | 661 |
| College of Agricultural & Environmental Sciences | 9 | 29 | 128 |
| College of Engineering | 1 | 12 | 33 |
| School of Forestry & Natural Resources | 1 | 12 | 31 |
| College of Veterinary Medicine | 4 | 12 | 29 |
| College of Public Health | 2 | 8 | 28 |
| College of Education | 2 | 5 | 20 |
| Terry College of Business | 3 | 5 | 10 |
| School of Ecology | 1 | 8 | 22 |
| School of Public and International Affairs | 1 | 3 | 3 |
| College of Pharmacy | 2 | 3 | 5 |
| | 40 | 214 | 970 |
| Centers & Institutes | 9 | 19 | 59 |
| TOTALS: | 49 | 233 | 1029 |

GACRC Users September 2015

| Centers & Institutes | PIs | Users |
|--|-----|-------|
| Center for Applied Isotope Study | 1 | 1 |
| Center for Computational Quantum Chemistry | 3 | 10 |
| Complex Carbohydrate Research Center | 6 | 28 |
| Georgia Genomics Facility | 1 | 5 |
| Institute of Bioinformatics | 1 | 1 |
| Savannah River Ecology Laboratory | 3 | 9 |
| Skidaway Institute of Oceanography | 2 | 2 |
| Center for Family Research | 1 | 1 |
| Carl Vinson Institute of Government | 1 | 2 |
| | 19 | 59 |

What are Linux Shell and Shell Scripting?

- Linux: A full-fledged operating system with 4 major parts
 - I. Kernel: Low-level OS, handling files, disks, RAM, networking, etc.
 - II. Supplied Programs: Web browsing, Audio, Video, DVD burning.....
 - III. **Shell**: A command-line user interface for a user to type and execute commands:
 - ✓ Bourne Shell (sh)
 - ✓ Korn Shell (ksh)
 - ✓ C Shell (csh)

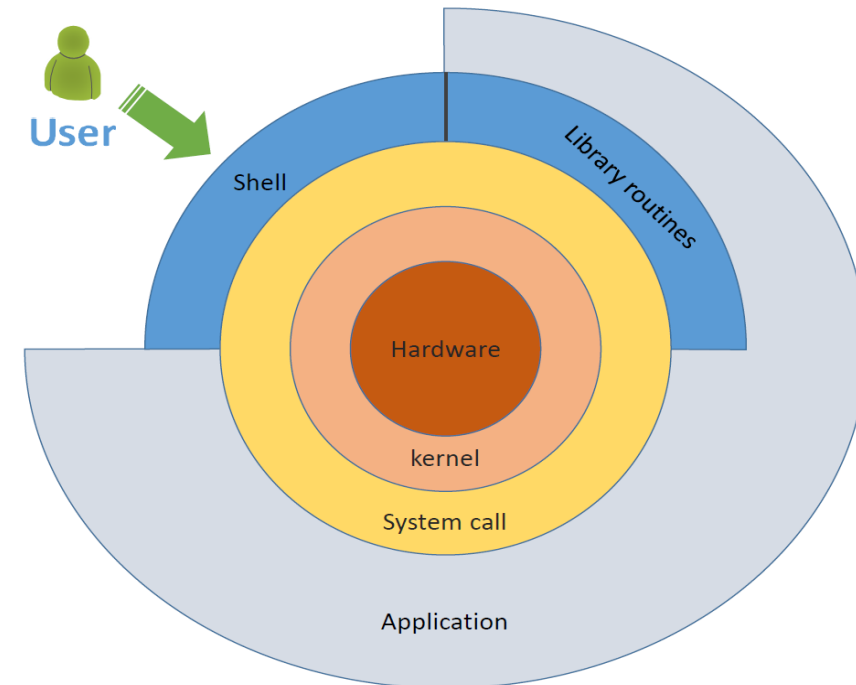
} UNIX standard shells

 - ✓ Bourne-Again Shell (**bash**) → Linux default shell
- IV. X: A graphical system providing graphical user interface(GUI)

What are Linux Shell and Shell Scripting?

- Linux Shell: A place to type and run commands on Linux
 - ✓ Command-line user interface for typing commands
 - ✓ Command interpreter to interpret & run commands
 - ✓ **Programming environment for scripting**
- Linux default: Bourne-Again Shell (**bash**)
- To open a shell on:

| | | |
|----------------------|--|-------------------------|
| Local Linux/Mac | shell window | Terminal |
| Local windows | shell window | SSH Secure Shell Client |
| Remote Linux machine | a shell will run immediately when log in | |



What are Linux Shell and Shell Scripting?

- Linux Shell Script: A **text file** running as a **program** to accomplish tasks on Linux that a single command cannot run
 - ✓ Variables
 - ✓ Expansion (**~**, **\$**, **`**, **\$(())**)
 - ✓ Quoting (**'**, **"**)
 - ✓ Commands (**|**, **;**)
 - ✓ Redirection (**>**, **>>**, **2>**, **2>&1**, **>&**, **<**)
 - ✓ Flow Control (**if-then-else**)
 - ✓ Loops (**for**, **while**)
- Linux Shell Scripting: Programming with **shell scripts** in **Linux shell**

Shell Scripting Syntax Basics – Variables

- Variable assignment: **name=value** (NO space! e.g., name =value is wrong!)

```
$ var1=kiwi      # all values held in variables are strings! var1="kiwi"
$ echo $var1     # echo prints the value of variable to screen
$ kiwi

$ var2=7         # same as var2="7"
$ echo $var2
$ 7

$ var3=$var1+7   # same as var3="kiwi+7"
$ echo $var3
$ kiwi+7

$ var4=10        # same as var4="10"
$ echo $var2+$var4
$ 7+10
```

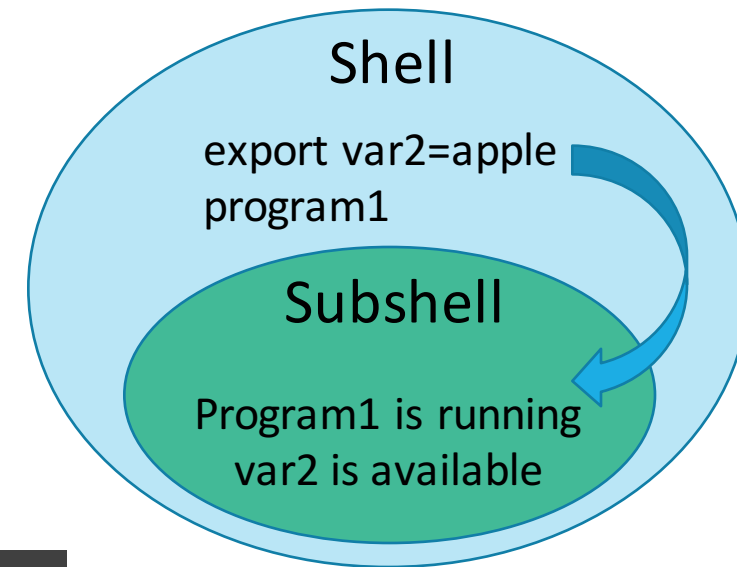
Shell Scripting Syntax Basics – Variables

- Exporting variables as global ***environment variables*** for use in a shell's child processes running in subshells → **export**

```
$ var1=kiwi
$ export var2=apple # var2=apple; export var2
$ printenv var1      # printenv prints env variables
$
$ printenv var2
$ apple
```

- Numeric expression to be evaluated? → **expr** or **\$((...))**

```
$ var1=10
$ var2=20
$ expr $var1 + $var2 # space and $ are required!
$ 30
$ echo $((var1+var2)) # space and $ are not required!
$ 30
```



Shell Scripting Syntax Basics – Variables

- bash automatically sets several ***shell variables*** at startup time (Note: Some shell variables may be environment variables* whereas others are local variables.)

| Shell Variables | Definition |
|------------------|---|
| HOME* | Home directory of the current user |
| PATH* | Search path for commands (colon-separated dirs in which shell looks for commands) |
| PWD* | Current working directory |
| SHELL* | Default shell currently being used |
| USER* | Current user's name |
| UID | Numeric user ID of the current user |
| LD_LIBRARY_PATH* | Shared library search path for C program (PERLLib, PYTHONPATH) |

Shell Scripting Syntax Basics – Variables

- Why we have those *shell variables*? → Configure user working environment!
- Example: **.bash_profile** for interactive login shell

```
if [ -f ~/.bashrc ]; then # if .bashrc exists and is a regular file, then
    . ~/.bashrc           # run/source it in current shell to
fi                         # make interactive login and non-login shell
                           # have the same environment

# User specific environment and startup programs
export PATH=$PATH:$HOME/bin    # append and export command searching path

# Zhuofei 2015-05-29
export PATH=$PATH:$HOME/scripts
```

Shell Scripting Syntax Basics – Variables

- Suggestion 1: “\$var” to prevent runtime errors in script

```
$ var="My Document"           # "My Document" is a directory
$ cd $var                     # same as cd My Document, 2 args
$ -bash: cd: My: No such file or directory
$ cd "$var"                   # same as cd "My Document", 1 args
My Document$
```

- Suggestion 2: \${var} to prevent unexpected behavior

```
$ var="apple"
$ echo "Mary has 3 $vars"      # variable vars is empty!
$ Mary has 3
$ echo "Mary has 3 ${var}s"    # ${var} is not working!
$ Mary has 3 {apple}s
$ echo "Mary has 3 ${var}s"    # ${var} is working!
$ Mary has 3 apples
```

Shell Scripting Syntax Basics – Expansion

➤ Tilde Expansion: `~`

```
$ cd ~username    # home directory associated username
$ cd ~            # replaced by $HOME
$ cd ~/           # same as above
```

➤ Variable Expansion: `$`

```
$ var=24
$ echo ${var}th  # outputs 24th; ${var} to prevent unexpected behavior!
```

➤ Command Substitution: ``command`` (back quota)

```
$ cd `pwd`        # same as cd /home/abclab/jsmith/workingDir
```

➤ Arithmetic Expansion: `$((expression))`

```
$ echo $(( (5+3*2)-1)/2 ))  # outputs 5; space is not required!
$ var1=24 ; var2=10        # ; for a sequence of commands
$ echo $((var1+var2))       # outputs 34
```

Shell Scripting Syntax Basics – Quoting

➤ Linux special characters

` ~ ! # % ^ & * () - + / \ | ; ' " , . < > ? {

➤ Quoting rules in bash

1. All special characters are disabled by enclosing single quotes `' '`
2. All special characters are disabled by enclosing double quotes `" "`
except for `!`, `$`, ```, `\`, and `{`
3. All special characters are disabled by a preceding backslash `\`

Shell Scripting Syntax Basics – Quoting

➤ Quoting Examples

```
$ FRUIT=apples
$ echo 'I like $FRUIT'           # $ is disabled by ' '
$ I like $FRUIT
$ echo "I like $FRUIT"          # $ is not disabled by " "
$ I like apples
$ echo "I like \ $FRUIT"        # $ is disabled by preceding \
$ I like $FRUIT
$ echo '`pwd`'                  # ` is disabled by ' '
$ `pwd`
$ echo "`pwd`"                  # ` is not disabled by " "
$ /home/abclab/jsmith
```


Shell Scripting Syntax Basics – Commands

- Pipeline **command1 | command2 | ...** connects std output of *command1* to the std input of *command2*, and so on (Demonstration)

```
$ ls -l | more
$ ls -l | grep ".sh"
$ ps aux | awk '{if($1=="zhuofei") print $0}' | more
$ qstat -u "*" | awk '{print $4}' | sort | less
$ qstat -u "*" | grep 'qw' | awk 'BEGIN{n=0} {n++} END{printf "%d jobs waiting on queue\n", n}'
```

- List **command1 ; command2 ; ... ;** simply runs commands in sequence on a single command line

```
$ pwd ; ls
$ cd .. ; ls
$ mkdir ./subdir ; cd ./subdir ; touch file1 ; ls
```

Shell Scripting Syntax Basics – Redirection

- Standard output redirection: **>** and **>>**

```
$ ls > outfile           # std output of a command is written to outfile
$ ls >> outfile          # std output of a command is appended to outfile
$ ./myprog > outfile     # std output of a program is written to outfile
```

- Standard error redirection: **2>**, **2>&1** and **>&**

```
$ ./myprog > outfile 2> errorfile    # std output and error → separate files
$ ./myprog > outfile 2>&1             # std output and error → single file
$ ./myprog >& outfile                 # same as above
```

- Standard input redirection: **<**

```
$ ./myprog < infile                # std input is from infile
```

- General usage

```
$ ./myprog < infile > outfile 2>&1
```

Shell Scripting Syntax Basics – Flow Control

➤ if-fi Block

```
if [ expression ] : if expression is evaluated to be true
then
    body1
else
    body2
fi
```

➤ Example (Demonstration)

```
echo "Please enter you name:"
read name                                # read a line from standard input
if [ "$name" == "zhuofei" ]             # true if strings are equal
then
    echo "Hello, ${name}!"
else
    echo "Hi, ${name}, you are not zhuofei!"
fi
```

Shell Scripting Syntax Basics – Flow Control

| | Test Expression | Description |
|-----------------|------------------------|--|
| File testing | -e file | True if file exists |
| | -d or -f file | True if file exists and is a directory or a regular file |
| | -r or -w or -x file | True if file exists and is readable or writable or executable |
| | -s file | True if file exists and has a nonzero size |
| | file1 -nt or -ot file2 | True if file1 is newer or older than file2 |
| String testing | -z or -n string | True if the length of string is zero or nonzero |
| | str1 == str2 | True if the strings are equal |
| | str1 != str2 | True if the strings are not equal |
| ARITH testing | arg1 OP arg2 | OP is one of -eq , -ne , -lt , -le , -gt , or -ge . Arg1 and arg2 may be +/- integers |
| Logical testing | ! expr | True if expr is false |
| | expr1 -a expr2 | True if both expr1 AND expr2 are true |
| | expr1 -o expr2 | True if either expr1 OR expr2 is true |

Shell Scripting Syntax Basics – Loops

➤ for Loop

```
for variable in list
do
    body
done
```

➤ Example (Demonstration)

```
for file in *.doc *.docx
do
    echo "$file is a MS word file!"
done
```

while Loop

```
while [ expression ]
do
    body
done
```

```
i=1
while [ $i -le 10 ]
do
    echo $i
    i=`expr $i + 1`
done
```

Real Shell Scripting Examples

- To create a shell script, simply put bash commands into a text file
- To run the script:
 1. Prepend `#!/bin/bash` to the very top of the script (the 1st line and left-justified)
 2. Make the script executable: `chmod 700 script.sh`
 3. Run the script: `./script.sh` OR
`script.sh` (if `.` is in PATH)

Real Shell Scripting Examples

- Example 1: A script to submit all job submission scripts in current working dir

```
#!/bin/bash

SUBDIR=`pwd`
CTR=1

for sub in ${SUBDIR}/*.sh ; do
    if [ "`basename ${sub}`" != "`basename $0`" ] ; then

        qsub -q rcc-30d ${sub} > ${SUBDIR}/outfile_${CTR}

        echo "`basename ${sub}` submitted!"
        CTR=$((CTR+1))
    fi
done
printf "\nTotally %d jobs submitted!\n\n" $((CTR-1))
qstat -u `id -un`
```

Real Shell Scripting Examples

- Example 2: A serial job submission script on zcluster

```
#!/bin/bash
cd `pwd`
time ./myprog < myin > myout
```

- Example 3: A MPI job submission script on zcluster (*default MPICH2 and PGI compilers*)

```
#!/bin/bash
cd `pwd`
export LD_LIBRARY_PATH=/usr/local/mpich2/1.4.1p1/pgi123/lib:${LD_LIBRARY_PATH}
mpirun -np $NSLOTS ./myprog
```

https://wiki.gacrc.uga.edu/wiki/Running_Jobs_on_zcluster

Thank You!