

Computer Programming

Lecture 4

이윤진
서울대학교
2007.12.31.

Slide Credits

- 엄현상 교수님
 - 서울대학교 컴퓨터공학부
 - Computer Programming, 2007 봄학기

Batch and Shell Programming

순서

- Batch Programming
- Shell Programming
 - Scripting Languages
- Bourne Shell Scripts
 - Variables and Backquotes
 - Quoting
 - Command-Line Arguments
 - Conditional Execution
 - Looping
 - Per-Case Execution
 - Array
 - Example
 - Miscellaneous
- Q&A

Batch Programming

- Batch
 - Group of Jobs (as Programs) Which Are Submitted for Processing on a Computer and Whose Results Are Obtained at a Later Time [webster.com]
 - Example

```
C:\>more test.bat
md mydir
echo start
pause
echo end
rd mydir
C:\>test.bat
```

Shell Programming

- Shell Script
 - Definition: Shell Program
 - Shell commands stored in a file
- Practical Use Cases
 - To automate tasks:
 - Simulation with Various Parameter-Value Assignments
 - Backups
 - Unpacking software and configuring it
 - Repeated operations on files
 - Etc.

UNIX Shells

- Bourne Shell (Steven Bourne)
 - 최초의 대중화된 shell
 - sh
- C Shell (Bill Joy)
 - C언어와 밀접
 - csh
- Korn Shell (Dave Korn)
 - Bourne shell보다 포괄적, 산업체에서 선호
 - ksh
- Bourne-Again Shell
 - GNU에서 Bourne shell 기반으로 작성
 - bash

UNIX Shells (계속)

- /etc/passwd

```
martini:~> more /etc/passwd
      : (omitted)
net001:x:3001:3001::/home/net/net001:/bin/bash
      : (omitted)
```

- echo \$SHELL
- ps

```
shell [ jin{41} ~] ps
  PID TTY          TIME CMD
 32061 pts/4    00:00:00 csh
 32092 pts/4    00:00:00 ps
```

Other Richer Scripting Languages

- perl or python
 - Suitable for a Lot of Arithmetic and String Manipulation
 - Probably Much Faster in Some Cases

Bourne Shell & Shell Programming

- /bin/sh

`which sh`

- .profile
 - Shell customization
- 대부분의 UNIX 시스템에서 사용
- 비교적 쉬운 shell script 언어
- `chmod +x file`

Bourne Shell Scripts

```
#!/bin/sh
# Generate some info
date
echo $HOME
echo $USER
echo $PATH
cal
```

sh -n 스크립트 : 문법 에러만을 검사, 명령을 실행하지 않음
sh -v 스크립트 : 명령을 실행하기 전에 에코
sh -x 스크립트 : 명령줄에서 처리한 다음 에코

- Result

```
2007. 04. 03. (화) 11:52:37 KST
/csehome/net001
net001
/usr/local/bin:/usr/bin:/bin
... (omitted)
```

Variables and Backquotes

- Variables
 - name=value
 - value is interpreted as a character string
 - do not insert a space before or after the =
 - \$name
 - to use the variable name in a statement
- Backquotes
 - substitute the command by its output
 - name=`ls`

Variables and Backquotes

```
cname="Computer Programming Class"  
gsnum=70  
echo $cname: "# of good students = $gsnum
```

Stored as a String

```
Computer Programming Class: # of good students = 70
```

```
... (same as above)  
bsnum=10  
stnum=$gsnum+$bsnum  
echo $cname: total "# of students = $stnum
```

Stored as a String

```
Computer Programming Class: total # of students = 70+10
```

```
... (same as above except for the last two lines)  
stnum=`expr $gsnum+$bsnum`  
echo $cname: total "# of students = $stnum
```

```
Computer Programming Class: total # of students = 80
```

Quoting

- ' (single quote): strong quoting
- " (double quote): weak quoting
- \ (backslash): quoting a single character

```
person=hatter
```

Expression	Value
<code>\$person</code>	<code>hatter</code>
<code>"\$person" [or] ""\$person"" [or] "\$person "</code>	<code>hatter</code>
<code>\\$person</code>	<code>\$person</code>
<code>'\$person'</code>	<code>\$person</code>
<code>""\$person"" [or] \'\$person\'</code>	<code>'hatter'</code>
<code>\"\$person\" [or] "\"\$person\""" [or] \\"\$person\"\"</code>	<code>"hatter"</code>
<code>\'\$person\'</code>	<code>'\$person'</code>
<code>""\$person""</code>	<code>"\$person"</code>

Command-Line Arguments

- $\$n$: n^{th} argument

```
echo '1st command-line argument = $1'  
echo "2nd command-line argument = $2"  
echo 3rd argument = $3  
echo "# of arguments = $#"  
echo Entire list of arguments = $*
```

- Result

```
martini:~$ arg.sh 1 2 3  
1st command-line argument = $1  
2nd command-line argument = 2  
3rd argument = 3  
# of arguments = 3  
Entire list of arguments = 1 2 3
```

Conditional Execution

```
if [ condition ]
then
...
elif [ condition2 ]
...
else
...
fi
```

Insert a space
before [and after]

String comparisons	
[string]	True if string is not null
[string1 = string2]	True if string1 is equal to string2
[string != string2]	True if string1 is not equal to string2
[-n string]	True if string is not null
[-z string]	True if string is null

Arithmetic comparisons	
[expr1 -eq expr2]	True if expr1 is equal to expr2
[expr1 -ne expr2]	True if expr1 is not equal to expr2
[expr1 -gt expr2]	True if expr1 > expr2
[expr1 -ge expr2]	True if expr1 >= expr2
[expr1 -lt expr2]	True if expr1 < expr2
[expr1 -le expr2]	True if expr1 <= expr2
[! expr]	True if expr is false
[expr1 -a expr2]	True if expr1 AND expr2 is true
[expr1 -o expr2]	True if expr1 OR expr2 is true

File conditions	
[-f FILE]	True if file exists and is a regular file
[-d FILE]	True if file is a directory
[-r FILE]	True if file is readable
[-w FILE]	True if file is writable
[-x FILE]	True if file is executable
[-O FILE]	True if file's owner is a current user
[FILE1 -nt FILE2]	True if file1 is newer than file2
[FILE1 -ot FILE2]	True if file1 is older than file2
...	...

Conditional Execution

```
if [ $1 -lt $2 ]; then
    echo $1 is less than $2
elif [ $1 -gt $2 ]; then
    echo $1 is greater than $2
else
    echo $1 equals to $2
fi
```

- Result

```
martini:~$ cond.sh 1 2
1 is less than 2
martini:~$ cond.sh 2 1
2 is greater than 1
martini:~$ cond.sh 1 1
1 equals to 1
```

Bourne Shell Script Example

```
if [ -z $1 ] || [ -z $2 ]; then
    echo usage: $0 source_dir target_dir
else
    SRC_DIR=$1
    DST_DIR=$2
    OF=output.`date +%y%m%d`.tar.gz
    if [ -d $DST_DIR ]; then
        tar -cvzf $DST_DIR/$OF $SRC_DIR
    else
        mkdir $DST_DIR
        tar -cvzf $DST_DIR/$OF $SRC_DIR
    fi
fi
```

- Execution

```
martini:~$backup.sh srcdir dstdir
```

Looping

```
for i in list do  
...  
done
```

- Shell executes the loop once for each value in *list*
- If *list* is omitted, the shell forms a default list from the command-line arguments

```
for str in "test1", "test2", "test3" do  
echo $str  
done
```

- Result

```
test1  
test2  
test3
```

```
for str  
do  
echo $str  
done
```

```
echo_file *  
/* echo all filenames in the current  
directory */
```

Looping

Double Quotes around \$@

```
for arg in "$@"; do # correct
    echo $arg
done
for arg in "$*"; do # wrong
    echo $arg
done
```

- Result

```
martini:~$ loop.sh 1 "2 2" 3
1
2 2
3
1 2 2 3
```

Looping (계속)

```
martini:~$ ls  
backup.C backup.c backup1.C backup1.c
```

```
martini:~$ more for.sh  
for cfilename in *$1*u?.[cC]; do  
    echo $cfilename  
done
```

- Results

```
martini:~$ for.sh a  
backup.C  
backup.c
```

```
martini:~$ for.sh c  
backup.C  
backup.c
```

Looping (계속)

let i=i+1 # bash only

```
i=0
while [ $i -lt 3 ]; do
  echo $i
  i=`expr $i + 1`
done
```

- Result

```
martini:~$ while.sh
0
1
2
```

Per-Case Execution

- How to use

```
case param in  
  pattern [ | pattern ] ... ) statement ;;  
  pattern [ | pattern ] ... ) statement ;;  
  ....  
  *) statement ;;  
esac
```

Per-Case Execution

```
echo Enter your command \(who, list, or quit\)
while read command; do
case $command in
  who) who; echo Done with running who;;
  list) ls; echo Done with running ls;;
  quit) break;;
  *) echo Enter who, list, or quit;;
esac
done
```

- Result

```
martini:~$ case.sh
Enter your command (who, list, or quit)
who
net001 pts/2 Mar 31 13:26 ... (omitted)
Done with running who
list
...
Done with running ls
date
Enter who, list, or quit
quit
martini:~$
```

Array (bash Only)

- Variable Containing Multiple Values
 - No Maximum Limit to the Size
 - No Requirements That Member Variables Be Indexed or Assigned Contiguously
 - Zero-Based

Array (계속)

```
a1=(one two three)
```

```
echo ${a1[*]}
```

```
echo a1[*]
```

```
echo ${a1[@]}
```

```
echo a1[@]
```

```
echo ${a1[0]}
```

```
echo a1[0]
```

```
echo ${a1[3]}
```

```
unset a1[3]
```

```
unset a1[*]
```

```
echo ${a1[0]}
```

- Result

```
one two three
```

```
a1[*]
```

```
one two three
```

```
a1[@]
```

```
one
```

```
a1[0]
```

Bash Script Example (계속)

```
i=0
status=0
if [ $# -lt 1 ]; then
    echo Usage: $0 [-b base1 base2...] [-c expo1 expo2...]
    exit 1
fi
for arg in "$@"; do
    if [ -n "`echo $arg | grep '-'"` ]; then
        if [ $arg = -b ]; then
            status=1
        elif [ $arg = -c ]; then
            status=2
        fi
    else
        case $status in
            1 ) base[$i]=$arg; i=`expr $i + 1` ;;
            2 ) exponent[$i]=$arg; i=`expr $i + 1` ;;
            * ) echo Usage: $0 [-b base1 base2...] [-c expo1 expo2...]; exit 1 ;;
        esac
    fi
done
# to be continued on the next slide
```

Bash Script Example (계속)

```
if ! [ -d OUTPUT ]; then
  mkdir OUTPUT
  if [ $? -gt 0 ]; then # if the exit status of the previous command indicates a failure
    echo Remove/Rename the OUTPUT File
    exit 1
  fi
fi
for i in ${base[@]}; do
  for j in ${exponent[@]}; do
    if ! [ -e ./OUTPUT/output.$i.$j ]; then
      ./pow $i $j >> ./OUTPUT/output.$i.$j
    fi
  done
done
```

- Execution

```
martini:~$powsim.sh -b 3 4 5 -c 12 14 16
```

Miscellaneous

- Stream Editor: sed
 - Global Search and Replace
- Character Translator: tr
 - Regular Expressions
- Functions
- grep
- ...

Stream Editor sed

- 텍스트를 한 번에 한 줄씩 처리 (출력, 삭제, 치환 등)
 - 입력: 표준입력이나 파일, 출력: 표준출력이나 파일
- 셸 스크립트에서 파이프에 걸어 쓸 수 있는 도구
- <http://kldp.org/HOWTO/html/Adv-Bash-Scr-HOWTO/x12718.html>

```
for filename in *$1* # 디렉토리에서 일치하는 모든 파일을 탐색.
do
    if [ -f "$filename" ] # 찾았다면...
    then
        fname=`basename $filename` # 경로를 떼어내고,
        n=`echo $fname | sed -e "s/$1/$2/"` # 새 이름으로 바꾼 다음,
        mv $fname $n # 이름 바꿈.
    fi
done
```

Character Translator: tr

- 문자변환필터
 - 지정된 대로 문자를 교체 또는 삭제
 - 소문자를 대문자로 바꾸거나 그 외 텍스트 처리작업을 할 때 주로 사용됨
 - <http://kldp.org/HOWTO/html/Adv-Bash-Scr-HOWTO/textproc.html>

```
#!/bin/bash
# 파일 내용을 모두 대문자로 바꿈.
E_BADARGS=65
if [ -z "$1" ] # 명령어줄 인자 여부의 표준 확인 작업.
then
    echo "사용법: `basename $0` filename"
    exit $E_BADARGS
fi
tr a-z A-Z <"$1"
exit 0
```