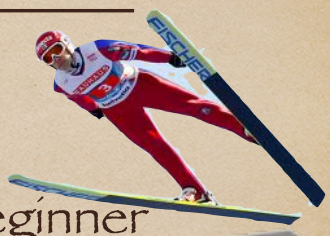
12th KIAS Protein Folding Winter School

January 21-25, 2013



Two Tutorials of Computing for beginner

- Bash Scripting / Using Linux
- Python Scripting

by Keehyoung Joo

Center for In-silico Protein Science

Center for Advanced Computation

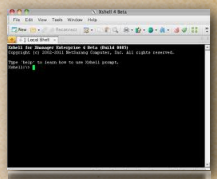
Korea Institute for Advanced Study

About tutoring...

- It's a kind of practical tutorial include real hand-on exercises using the actual computer.
- Welcome all kind of questions about computing-things.
- http://lee.kias.re.kr/~newton/tutorial/ws2013

Pre-requisites

- Linux or Mac OSX
  - gnome-terminal, xterm, ... (Linux)
  - terminal, iterm (Mac OSX)
- Windows
  - X-shell, Xmanager
  - putty (http://lee.kias.re.kr/~newton/tutorial/ws2013/putty.exe)



Bash shell Scripting for Beginner with examples

12th KIAS Protein Folding Winter School

January 21-25, 2013

Keehyoung Joo

Korea Institute for Advanced Study

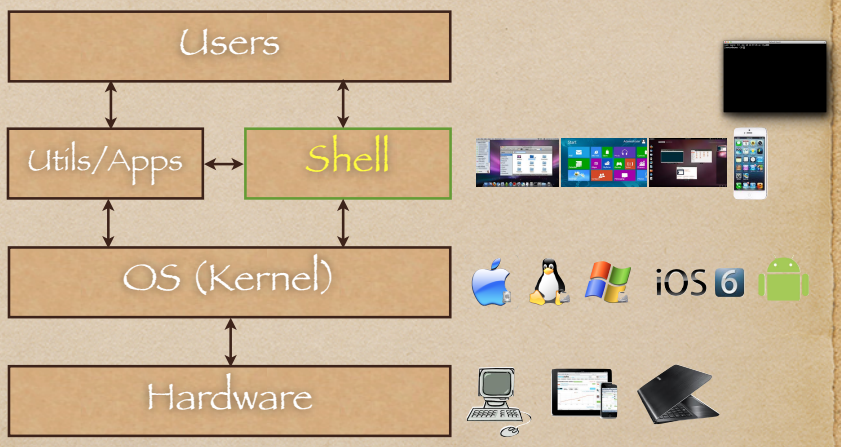
References ...

- Bash Guide for Beginners (http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html)
- Linux Shell Scripting Tutorial (http://www.freeos.com/guides/lsst/)
- Advanced Bash-scripting Guide (http://tldp.org/LDP/abs/html/)
- and, Googling for anthings

Contents

- What is the Shell?
- Basic Commands
- Shell Scripting (Programming)
- Conclusions...

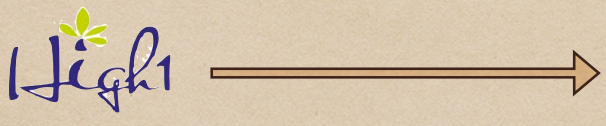
What is the SHELL?



- Interface btw OS and Users: interprets and executes user typed commands
  - GUI (Graphical Use Interfae) : Windows, Mac OSX, Linux, IOS, Android
  - CLI (Command Line Interface) : bash, csh, zsh, command.com, etc...

Let's login...

KiasWS2013



Xshell, putty, xterm, etc

KIAS Linux cluster "fold"
Center for Advanced Computation

```
% ssh  guest@fold.kias.re.kr
```

Now, you are connected...
and, you can do all things now with

```
[guest@fold ~] % _
```

"put your command, I execute it!"

Warm up (Basic Commands)

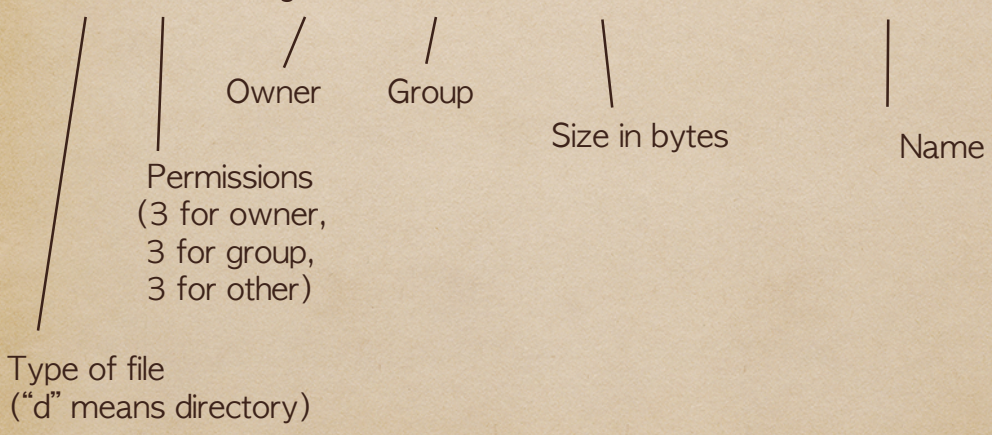
```
% pwd
% ls
% ls -l
% man ls
%
% pdbget.sh 4g8v
% less 4g8v.pdb
% df -h
% free
% ps ax
% whoami
% who
% w
% top
```

```
* For this tutorial
 % cd users/
 % mkdir [your own ID]
 % mkdir [your ID]       # for example
 % cd [your ID]
 % cp -r ../newton/* .

 % ls -l
 % cd bash               # for bash
 % ls -l
```

```
* Directories
% pwd              (print working directory)
/net/guest

% mkdir work       (make directory)
% cd work
% pwd
/net/guest/work

% cd /usr/bin
% pwd
% cd
% pwd              (change your home directory)
/net/guest

% rmdir work       (remove directory)
  or ‘rm -r‘       (recursive rm)
```

```
* Listing files
% ls /home        (print working directory)


% cd
% ls
% ls -a
% ls -al
% ls -tl
% ls -tlr
```

```
% cd users/newton/bash
% ls -l
total 236
-rwxrwxr-x 1 guest 2013winter    337 Jan 20 06:42 xyz.sh
-rwxrwxr-x 1 guest 2013winter 215469 Jan 20 06:43 4g8v.pdb
drwxrwxr-x 2 guest 2013winter   4096 Jan 21 06:12 work
drwxrwxr-x 2 guest 2013winter  12288 Jan 21 11:00 T0736
```



```
* Viewing Files, less
% cd /net/guest/users/newton/bash
% cat 4g8v.pdb
% less 4g8v.pdb

* Symbolic Links
% ln -s 4g8v.pdb my.pdb
% ls -l
% cat my.pdb
% less my.pdb
```

```
* File Copy, Move (rename), remove
% cd /net/guest/users/newton/bash
% cp 4g8v.pdb test.pdb                  (copy)
% ls -l
% mkdir temp
% mv test.pdb temp                      (move)

% cd temp/
% rm test.pdb                           (remove)
% cd ../
% rm -r temp                            (recursive rm)
```

```
* Shells
  bash              Bourn Again Shell
  tcsh              Turbo C-shell
  ksh, zsh, etc...
% echo $SHELL       (print your shell)
% chsh              (change your shell)

* Filename Expansion
% cd /net/guest/users/newton/bash/T0736
% ls b*.pdb
% ls b0??.pdb
% ls b00[12].pdb
% ls b00[13].pdb
% ls b00[1-3].pdb
% ls *.{pdb,ent}
```

```
* Saving your output
% cd /net/guest/users/newton/bash/T0736
% ls -al > output                (redirection)
% ls / > output
% ls /net > output
% ls /net/guest >> output        (append)

% ls test.pdb
% ls test.pdb > messages     (‘>’ does not redirect
                              standard error)
% ls test.pdb &> messages
% ls b001.pdb test.pdb 1> message
% ls b001.pdb test.pdb 2> message
% ls b001.pdb test.pdb &> message
```

```
* Etc commands
% who                  (who login?)
% w
% w | grep guest       (show line with “newton” string)
% finger guest         (show user status)

% free                 (show memory usage)
% df                   (show disk usage)

% ps                   (show process list)
% ps ax                (all process list with user id)

% top                  (show process status)
% man ls               (manual page for “ls”)

% find -name b001.pdb
% find /net -name “*.sh”
```

```
* Network Commands
1) Remote connect
 % ssh guest@fold.kias.re.kr
2) Remote copy
 % scp -r test.pdb guest@fold.kias.re.kr
3) wget
 % wget -O 1ton.pdb “http://www.pdb.org/pdb/download/downloadFile.do?
fileFormat=pdb&compression=NO&structureId=1ton”
3) ftp
 % lftp -u newton@lee.kias.re.kr
```

Application commands

```
% cd bash
% cat 4g8v.pdb

% grep “^ATOM” 4g8v.pdb
% grep “^ATOM” 4g8v.pdb | grep “ A “ > 4g8vA.pdb
% grep “^ATOM” 4g8v.pdb | grep “ B “ > 4g8vB.pdb
% grep “ CA “ 4g8vA

% grep ^HELIX 4g8v.pdb > 2ndary.out
% grep ^SHEET 4g8v.pdb >> 2ndary.out
% grep ^SSBOND 4g8v.pdb >> 2ndary.out

% egrep ‘ N | CA | C ‘ 4g8vA.pdb
% egrep ‘ N | CA | C ‘ 4g8vA.pdb | grep -v “ CA B”
% egrep ‘ N | CA | C ‘ 4g8vA.pdb | grep -v “ CA B” | grep “PRO”
```

Editors... (for text editing)

* Full featured editors in Linux/Unix

% vi

% emacs

* Easy, simple editor for beginner

% nano

Hello World ...

```
% cd work
% nano hello.sh

#!/bin/sh
echo "Hello World!"

% bash hello.sh        # execution  ( sh hello.sh )
Hello World!

% ls -l hello.sh
% chmod +x hello.sh    # change file mode
% ls -l hello.sh
% ./hello.sh           # execution
```

Shell Scripting

- Variables
- Arithmetics
- Branches
- Loops
- Functions

Example (multi.sh)

```
% cd work/
% ./multi.sh 2
```

bash interpreter

variable and shell parameter

branche

print statement

loop

arithmetic

```
#!/bin/sh

a=$1

if [ $a -lt 1 -o $a -gt 9 ]; then
    echo “The number is out of range [1,9]”
    exit
fi

echo "Multiplication Table for $a"

for i in 1 2 3 4 5 6 7 8 9
do
    m=$[a * i]
    echo "$a x $i = $m"
done
```

Variables

* Their values are always stored as strings (default)
* No need to declare a variable

- Environmental variables

```
% echo $SHELL
% echo $HOME
% echo $PATH
% echo $USER
```

- User defined variables

```
% i=10                          # no space
% echo $i                       # $ for using variable
% str=”Hello World”
% echo $str
```

Quotes...

```
% bash quotes.sh
```

1) Double quotes (partial quoting)

```
% echo $HOME
% str="My home directory is $HOME"        # double
% echo $str
My home directory is /net/guest
```

2) Single quotes (full quoting)

```
% echo $HOME
% str='My home directory is $HOME'        # single
% echo $str
My home directory is $HOME
```

Quotes... (cont.)

3) Back quotes (command substitution)

```
% pwd
/net/guest
% str=`pwd`              # or str=$(pwd)
% echo "My home directory is $str"
My home directory is /net/guest

% cd T0736               # in the bash directory
% ls b*.pdb              # or ls b???.pdb
% pdbs=`ls b*.pdb`
% echo $pdbs
```

Arithmetics...

Operators:

```
+           plus
-           minus
*           multiplication
/           division
**          exponentiation
%           modulo
```

Examples:

```
% let a=10+2*7        # no space, but let a=(10 + 2 *7)
% echo $a
24
% a=$((10+2*7))      # or a=$[10+2*7]
% echo $a
24
```

Arithmetics... (cont.)

Examples:

```
% a=1
% b=2
% c=$((a+b))  # or c=$[a+b]
% echo $c
```

Examples:

```
% cd work/
% cat arithmetic.sh
% ./arithmetic.sh
```

Arithmetics... (cont.)

```
Examples:
  % a=1.2                  # floating point
  % b=3.1                  # floating point
  % c=$[a+b]               # you get error...
...
  % echo “$a + $b”
  1.2 + 3.1                # it is just string...

  % echo “$a + $b” | bc
  4.3
  % c=`echo “$a + $b” | bc`
  % echo $c
  % echo “$a * $b” | bc -l
  % c=`echo “$a * $b” | bc -l`
  % echo $c
  3.72
```

Examples:

```
% cd work/
% cat float.sh
% ./float.sh
```

Branches

* Basic form

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

* Extended form 1

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

* Extended form 2

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

Examples:

```
% cd work/
% cat branches.sh
% ./branches.sh
```

* Number Comparisons:

-eq	(==)	equal
-ge	(>=)	greater than or equal
-le	(<=)	less than or equal
-ne	(!=)	not equal
-gt	(>)	greater than
-lt	(<)	less than

* String Comparisons:

```
=    equal
!=   not equal
-n   string is not empty
-z   string is empty
```

Examples:

```
% cd work/
% cat string.sh
% ./string.sh
```

* Files operators

```
-e   check for file existence
-d   check for directory existence
-s   check if file is nonzero size
....
```

Examples:

```
% cd work/
% cat file.sh
% ./file.sh
```

* Logical operators:

```
!    Not
-a   AND
-o   OR
```

Examples:

```
% cd work/
% cat logic.sh
% ./logic.sh
```

% cat logic.sh

```
#!/bin/sh
echo -n "Enter a number in range [1,10]: "
read num

if [ $num -ge 1 -a $num -le 10 ]; then
    echo "$num * $num = $[$num*$num]"
else
    echo "$num is out of range [1,10] !"
fi
```

* Logical operators:

```
&&   AND
||   OR
```

Examples:

```
% cd work/
% cat logic2.sh
% ./logic2.sh
```

% cat logic2.sh

```
#!/bin/sh
echo -n "Enter a number in range [1,10]: "
read num

if [ $num -ge 1 ] && [ $num -le 10 ]; then
    echo "$num * $num = $[$num*$num]"
else
    echo "$num is out of range [1,10] !"
fi
```

Shell Arguments

```
#!/bin/sh

echo "Total number of arguments = $#"
echo "Shell script name         = $0"
echo "First arguemnt            = $1"
echo "Second arguemnt           = $2"
echo “Third argument            = $3”
echo "All arguments (a word)    = $*"
echo "All arguments in array    = $@"
```

```
% cd work/
% ./args.sh 1 2 test
```

Loops (for statement)

```
#!/bin/sh
for i in 1 2 3 4 5 6 7 8 9
do
    echo “number $i”
done
```

```
#!/bin/sh
names=$@
for $i in $names
do
    echo “argument $i”
done
```

```
#!/bin/sh
for i in `seq 1 100`
do
    if [ $i -lt 5 ]; then
        continue
    elif [ $i -gt 10 ]; then
        break
    fi
    echo "number $i"
done
```

```
% cd work/
% ./for1.sh
% ./for2.sh
% ./for3.sh 1 2 test
```

Loops (while statement)

```
while expression
do
   statements
done
```

```
#!/bin/sh

i=1
sum=0
while [ $i -le 10 ]
do
    sum=$[sum + i]
    echo "$i sum = $sum"
    i=$[i+1]
done
```

```
% cd work/
% ./while.sh
```

Strings...

`${#string}`	*gives the string length*
`${string:position}`	*extracts sub-string from $string at $position*
`${string:position:length}`	*extracts $length characters of sub-string from $string at $position*

```
% str="0123456789"

% echo ${#str}
10
% echo ${str:6}
6789
% echo ${str:6:2}
67
```

Examples

```
#!/bin/sh
pdb=$1

while read line
do
    if [ "${line:0:4}" != "ATOM" ]; then
        continue
    fi
    resnam=${line:17:3}
    anam=${line:12:4}
    x=${line:30:8};y=${line:38:8};z=${line:46:8}
    if [ "$anam" != " CA " ]; then
        continue
    fi
    echo "$resnam $anam: x,y,z= $x $y $z"
done < $pdb
```

```
% cd bash/
% ./xyz.sh 4g8vA.pdb
```

Arrays...

```
% a[0]=1
% a[1]=2
% a[2]='A'
% echo "${a[0]} ${a[1]} ${a[2]}"
% echo "${a[*]}"
% echo "${#a[@]}"

% a=(1 2 3)
% s=('A' 'B' 'C' 'D')
% echo ${a[*]}
% echo ${s[*]}
```

```
% cd work/
% cat array.sh
% ./array.sh
```

Functions

```
#!/bin/sh

function tm_score () {
 local pdb1=$1
 local pdb2=$2
 echo "`tmscore $pdb1 $pdb2 | grep ^TM | awk '{print $3}'`"
}

a=$1
b=$2

tm_score $a $b
```

```
% cd T076
% ./func.sh b001.pdb native.pdb
```

More scripts...

In the T0736 directory.

```
* TM-score calculation for given pdbs
% ./tm.sh b*.pdb

* Pair-wise TM-score calculation for given pdb list.
% cat list
% ./pair.sh list

* Pair-wise TM-score calculation for given pdb list
with function evaluation.
% ./pair_func.sh list

* get pdbs
% ./pdbget.sh 1ton 4g8v 1fsd
```

Conclusion

- Bash is very clear and powerful scripting tool to manipulate your data.
- Let's bash on your terminal with your own ideas.