Bash: Bourne again shell

Common command:

“pwd” //current dir.

“ls” //list

“man –ls” //help for ls.

“clear” //clear

“cd” //enter directory.

“ls > place” //redirect output

“grep search_term file_name” //search words in file

“ctrl+c” //terminate the command

Redirect Output:

$ who > users //redirect to users. overwrite

$ cat users //open users

$ who >> users //append

oko tty01 Sep 12 07:30

ai tty15 Sep 12 13:32

ruth tty21 Sep 12 10:10

pat tty24 Sep 12 13:07

steve tty25 Sep 12 13:03

$

Redirect Input:

command < file

More redirect:

Every time Unix is running commands, it opens three files:

Stdin 0

Sdtout 1

Stderr 2

$command 2 > file

//redirect stderr tofile. >>append.

/dev/null

$ command > /dev/null

//redirect command to /dev/null

//all the things that's redirected to /dev/null will be discarded.

Bash script syntax.

A script can contain:

Variables/Arguments/Flow-control logic

Build Bash Script

You script will start with an interpreter directive that's sometimes called a hashbang. //#!
#!/bin/bash ///bin/bash path to the bash executable
#comment

"bash my.sh" //to execute
or "chmod +x ./test.sh"+ "./test.sh"
//echo display text

"echo"
//the difference between no quote, one quote and double quote.
"echo –e "Message" " //display the message with special meaning, or it's display without translating.
"\\" \
"\a" ring
"\n" new line.
"\t" tab
"\r" return

Variable：

“a=”Hello” ”

“b=”Good” “ //b=”good”

“c=7” //c=7

Use: $+name

“echo $a” or “echo ${a}”

//{}is optional, {} is for making the boundary of the variable clear

“echo $b”

“declare –i d=123” //d is an integer.

“declare –r e=456” //e is read-only

Built-in variables.

“echo $HOME” //Home directory

“echo $pwd” //current directory

“echo $MACHTYPE” //Return machine type.

“echo $BASH_VERSION” //return bash version

“echo $SECONDS” //number of seconds the bash session has run handy for timing things.

“echo $0” //name of the script

“echo $#” //the number of arguments

“echo $n” //the argument. N=1 means the first argu.

“echo $?” //the state when last command exits, or the return value of the function.

Get build-in variables into local variable:

“d=$(pwd)”

“echo $d” //equals to “echo $pwd”

//if echo is used with variables, use “{}”

Variable substitute:

${var:-word} //if var==null, return word, don’t change var

${var:=word} //if var==null, var=word and return var.

${var:+word} //if var!=null, return word,don’t change var

Math:

“((expression))” //

“a= “((expression))” ”assign results to a;

“((e++))”

“((e*=3))”

//if not double brackets, treated as string, not integar.

Comparison operations:

“[[expression]]” //space between the sets of brackets. //use

Compare Strings: ‘>,<,=’

Compare Integers: ‘-gt,-lt,-le,-ge,-eq,-ne’ //dash gt

Compare Logic: “&&,||,!”

Compare String null:”-z, -n” //not null,null

“[[20 –gt 100]]”

“[[“cat” == “cats”]]”

“a=”” “

“b=”cat” “

“[[-z $a && -n $b]]”

String Manipulation:

//the difference between ‘’ and “”

//’’ don’t translate special words, “” do.

“a=”hello” ”

“b=”world” ”

“c=ab “ //string cat

“echo ${#a}” //string length of a

“d=${c:3}” //find substrings. c.3-c.10

“d=${c:3:4}” //start from 3 with 4 letters.

String substitute:
“fruit=”apple banana banana cherry” ”
“echo ${fruit/banana/durian}” //replace the first banana with durian.
“echo ${fruit//banana/durian}” //replace all of the banana
“echo ${fruit/#banana/durian}” //#check if its from the beginning. If not, no change will happen.
“echo ${fruit/c*/durian}” //match expression

Coloring and styling text. (skipped)

Date:
“date” //show the time
“date + “%d-%m-%Y” ”

“printf “Name:\t%s\nID:\t%04d\n” “String1” “Number1” ”
//Nmae:String1 ID:0035

Date+Printf

Today=$(date + “%d-%m-%Y”)

Time=$(date + “%H:%M:%S”)

“printf “%s%s” “Today” “Time” ”

Array: //use blank to separate the objects in array.

“a=()” //empty array.

“b={“apple” “banana” “cherry”} ”

“echo ${b[2]}” //retrieve b[2], start from 0

“b[5]=”kiwi” ”//add to b[5]

“b+=(“mango”) //add mango”

${array_name[index]}//read data from array.

Key Array

“declare –A myarray”

“myArray[color]=blue” //use qupte when have space

“myArray[“office”]=”HQ West” ” //use qupte when have space

echo ${myarray[“office”] } is ${myarray[color]}}

the number of elements in the arry

length=${#array_name[@]}

#or

length=${#array_name[*]}

get the length of a single element

lengthn=${#array_name[n]}

Working with files.

“echo “Some text” > file.txt” //write “Some txt”, create file.txt and overwrite everything in it.

“echo “Some text” >> file.txt” //add to end

How to set up a list of instructions to execute automatically.

“vi ftp.txt” //create a file

“write your code in ftp.txt”

“bash < ftp.txt” //input ftp.txt.

Using here document //redirect the information between two delimiters.

```
command << delimiter
    document
delimiter
```

“cat << EndOfText”

“This is a”

“Multiline”

“Strubg”

“EndOfText” //Two EndOfText is not considered

Control Structures.

1. If else

If [expression] //must have blank

Then

    Statement1

    Statement2

Fi

2. If else fi

if [expression]

then

Statement(s) to be executed if expression is true

else

Statement(s) to be executed if expression is not true

Fi

3. if elseif fi

if [expression 1]

then

Statement(s) to be executed if expression 1 is true

elif [expression 2]

then Statement(s) to be executed if expression 2 is

```
true
elif [ expression 3 ]
then Statement(s) to be executed if expression 3 is true
else
  Statement(s) to be executed if no expression is true
fi
```

4. case

```
case %theVar in
    ValueOfVar1) commands
    ;;
    ValueOfVar2) commands
    ;;
    …
    *) commands
    ;;
esac
```

5. for

```
for theVar in 1 2 3 4 5
do
```

```
    commands
done
//for theVar=1 commands..
// for theVar=2 commands..
//loop

for str in ‘The string’
do
    echo $str
done
//”The string”

for FILE in $HOME./bash*
do
    echo $FILE
done
// /root/.bash_history
// /root/.bash_logout
// /root/.bash_profile
// /root/.bashrc
```

6. while [expression] //mind the blank

```
do
    commands
done

COUNTER=0
while [ $COUNTER -lt 5 ]
do
        COUNTER='expr $COUNTER+1'
        echo $COUNTER
done
```

7. Break

Use break to jump out of the loop.

“ break” or “break n”//n starts from 1, the outer loop, the number is bigger.

8. Continue

Same in C.

Function in Shell:

Define:

function function_name () //don’t have to define arguments

{

  list of commands

  [return value]

}

```
# Define your function here
Hello () {
      echo "Url is http://see.xidian.edu.cn/cpp/shell/"
}

# Invoke your function
Hello
```

Or

```
#!/bin/bash
funWithReturn(){
       echo "The function is to get the sum of two numbers..."
```

```
    echo -n "Input first number: "
    read aNum
    echo -n "Input another number: "
    read anotherNum
    echo "The two numbers are $aNum and $anotherNum !"
    return $(($aNum+$anotherNum))
}
funWithReturn
# Capture value returnd by last command
ret=$? //receive the result from function
echo "The sum of two numbers is $ret !"


#!/bin/bash
funWithParam(){
    echo "The value of the first parameter is $1 !" //call the first arguments
    echo "The value of the second parameter is $2 !"
    echo "The value of the tenth parameter is $10 !"
    echo "The value of the tenth parameter is ${10} !"
```

```
    echo "The value of the eleventh parameter is ${11} !"
    echo "The amount of the parameters is $# !"
    // the number of arguments
    echo "The string of the parameters is $* !"
    //all the arguments passed to the function
}
funWithParam 1 2 3 4 5 6 7 8 9 34 73
```