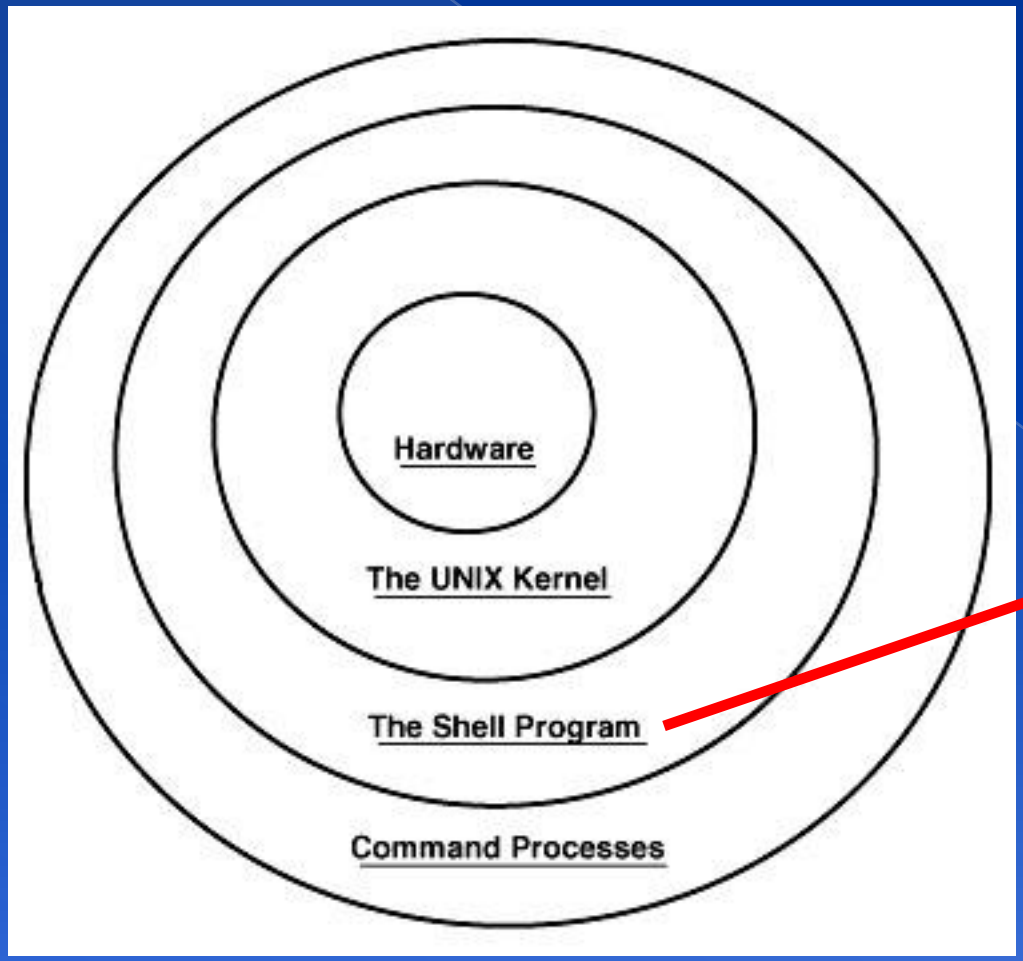
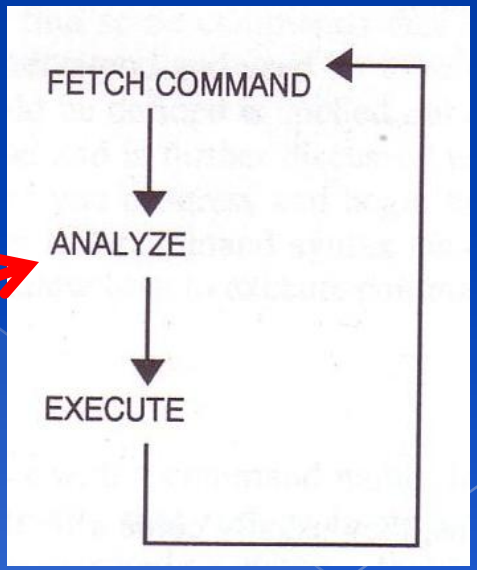
Shell and Shell Programming

Introduction - You, Kernel & Shell





Introduction - The UNIX Shells

Shell	Originator	System Name	Prompt
Bourne Shell	S. R. Bourne	/bin/sh	$
Csh	Bill Joy	/bin/csh	%
Tcsh	Ken Greer	/bin/tcsh	>
Korn Shell	David Korn	(shells/ksh93)	$
Z Shell	Paul Falstad	(shells/zsh)	%

- CSH - Berkeley C Shell
- BASH - Bourne Again SHell
- TCSH - TENEX C SHell

Introduction - Shell Program

- A.K.A. shell script
  - A collection of commands
- Ex:

```
#!/bin/sh

ls -al
touch aa
cp aa bb
```

- What you have to learn?
  - Some magic in UNIX environment
  - UNIX commands
  - Shell program structure

Shells - Startup files

- sh
  - /etc/profile — login shell, system wide
  - ~/.profile — login shell
  - ENV
- csh
  - /etc/csh.cshrc — always, system wide
  - /etc/csh.login — login shell, system wide
  - ~/.cshrc — always
  - ~/.login — login shell
  - ~/.logout — logout shell
  - /etc/csh.logout — logout shell, system wide
- tcsh
  - ~/.tcshrc — login shell
- bash
  - /etc/profile → ~/.bash_profile → ~/.bash_login → ~/.bash_profile
  - /etc/bash.bashrc → ~/.bashrc
  - BASH_ENV

Shells - Shell Special Characters (1)

- Reduce typing as much as possible

Characters	Description
*	Match any string of characters
?	Match any single alphanumeric character
[…]	Match any single character within []
[!...]	Match any single character not in []
~	Home directory

- Example
  - If following files:

    test1 test2 test3 test4 test-5 testmess

    are in current directory.

Command	Result
% ls test*	test1 test2 test3 test4 test-5 testmess
% ls test?	test1 test2 test3 test4
% ls test[123]	test1 test2 test3
% ls ~	List files under your home ($HOME)

Shells - Shell Special Characters (2)

Char.	Purpose	Example
#	Start a shell comment	# this is a comment
;	Command separator	% ls test*; ls test?
&&	executes the first command, and then executes the second if first command success (exit code=0)	% cd foo/bar && make install clean
||	executes the first command, and then executes the second if first command fail (exit code≠0)	% cp x y || touch y
\	(1) Escape character (2) Command continuation indicator	% find /home -name *.mp3 % ls \ > test*
&	Background execution	% make buildworld &

Shells - Shell Environment Variables

- Controlling shell behaviors
  - There are many environment variables that control the shell behavior
- To dump them: % env
- To get value: $variable_name or ${variable_name}
- Useful Environment Variables

sh	csh	Description
HOME	HOME	User's home directory
MAIL	MAIL	User's mail box
PATH	PATH	Search path
PS1	prompt	Primary prompt string (waiting command)
PS2	prompt2	Secondary prompt string (after lines end with \)
	prompt3	Third prompt string (automatic spelling correction)
IFS		Internal field separators
	history	Number of history commands

Shells - Variables and Strings Quotes

Char.	Purpose
var=value set var=value	Assign value to variable
$var ${var}	Get shell variable
`cmd`	Substitution stdout
'string'	Quote character without substitution
"string"	Quote character with substitution

- % varname=`/bin/date`
- % echo $varname
- % echo 'Now is $varname'
- % echo "Now is $varname"

- % set varname2=`/bin/date`
- % echo $varname2
- % echo 'Now is $varname2'
- % echo "Now is $varname2"

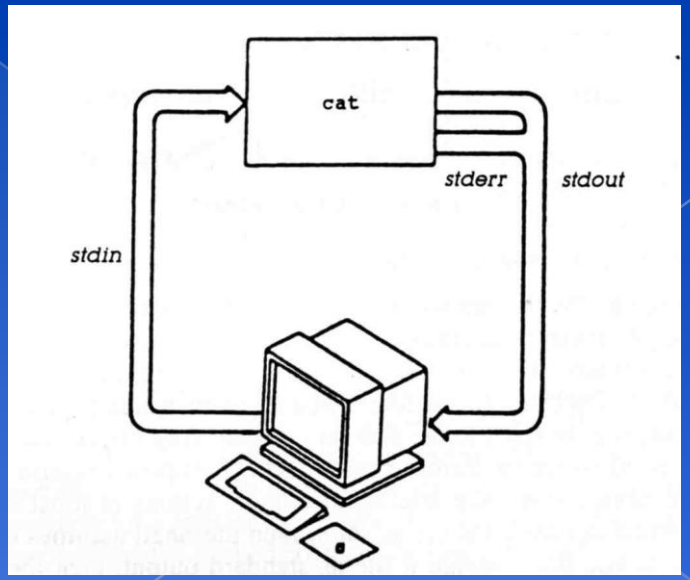
Tue Nov 13 14:54:57 CST 2007
Now is $varname
Now is Tue Nov 13 14:54:57 CST 2007

Shells - Input/Output Redirection (1)

- Every process has 3 default file descriptors

Name	I/O	Descriptor #
stdin	input	0
stdout	output	1
stderr	error output	2
User-defined	Input/output	3 ~ 19

- In normal situation
  - The terminal will be stdout and stderr
  - The keyboard will be stdin



Shells - Input/Output Redirection (2)

- Redirection
  - Change the direction of stdin, stdout, stderr or any other user-defined file descriptor
    - Create files
    - Append to files
    - Use existing files as input
    - Merge two output streams
    - Use part of the Shell command as input

Shells - Input/Output Redirection (3)

Operator	Description
<	Open the following file as stdin
>	Open the following file as stdout
>>	Append to the following file
<<del	Take stdin from here, up to the delimiter del
>&	Merge stdout with stderr
>>&	Append stdout to stderr
|	Pipe stdout into stdin
n>&-	Close file descriptor

Shells - Input/Output Redirection (4)

- Examples
  - % echo "we have several shell" > chapter1
  - % sed -e 's/shell/SHELL/g' < chapter1
    - we have several SHELL
  - % sed -e 's/SHELL/shell/g' < chapter1 > newchapter1
    - stdout goes to newchapter1 file
    - stderr still goes to terminal

  - % sed -e 's/SHELL/shell/g' < chapter1 > newchapter1 2> errchapter
    - stdout goes to newchapter1 and stderr goes to errchapter

  - % sed -e 's/SHELL/shell/g' < chapter1 2>&1
    - Both stdout and stderr go to terminal

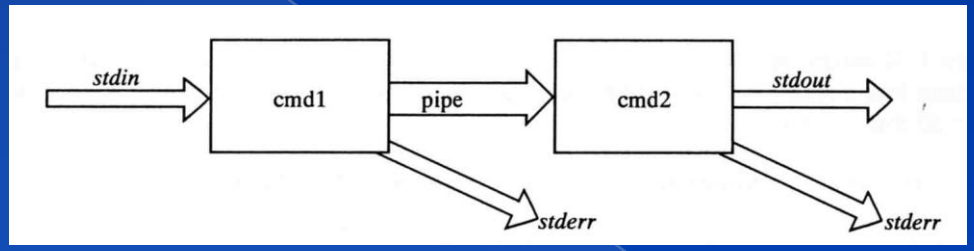
  - % sed -e 's/SHELL/shell/g' < chapter1 > newchapter1 2>&1
    - Both stdout and stderr go to newchapter1

  - % sed -e 's/SHELL/shell/g' < chapter1 >& newchapter1

noclobber

Shells - Input/Output Redirection (5)

- pipe
  - Connect the stdout of one command to the stdin of another
  - Two commands will operate asynchronously



- Example
  - % dmesg | grep CPU | less
  - To merge stderr with stdout and pipe to next command
    - % command arguments 2>&1 | nextcommand
    - % command arguments |& nextcommand
  - % exec 4>&- # close file descriptor 4
  - % exec 1>&- # close stdout

Shells - Built-in Shell Commands (1)

sh	csh	description
Alias/unalias	alias/unalias	command alias
ulimit	limit/unlimit	limit job's resource usage
cd	cd	change directory
echo	echo	write arguments on stdout
eval	eval	evaluate and execute arguments
exec	exec	execute arguments
exit	exit	exit shell

Shells - Built-in Shell Commands (2)

sh	csh	description
	goto	Goto label within shell program
	history	Display history list
jobs	jobs	List active jobs
%[job no.]	%[job no.]	Bring a process to foreground
kill	kill	Send a signal to a job
fg, bg	fg, bg	Bring a process to foreground/background
	stop	Stop a background process
	suspend	Suspend the shell
login	login, logout	Login/logout

Shells - Built-in Shell Commands (3)

sh	csh	description
set/unset		Set/Unset shell's parameters
	set/unset	Set/Unset a local variable
export	setenv/unsetenv	Set/Unset a global variable
	nice	Change nice value
	nohup	Ignore hangups
	notify	Notify user when jobs status changes
trap	onintr	Manage execution signals
	dirs	print directory stack
	popd, pushd	Pop/push directory stack

Shells - Built-in Shell Commands (4)

sh	csh	description
hash	rehash	Evaluate the internal hash table of the contents of directories
read		Read a line from stdin
shift	shift	Shift shell parameters
.	source	Read and execute a file
times	time	Display execution time
umask	umask	Set default file permission
test		Evaluation conditional expressions
expr	@	Display or set shell variables
wait	wait	Wait for background jobs to finish

Shells - Built-in Shell Commands (5)

- References:
  - http://www.unet.univie.ac.at/aix/aixuser/usrosdev/list_bourne_builtin_cmds.htm
  - http://www.europa.idv.tw/UNIX-Shell/csh/V2-01-09.html
  - http://www.unix.org.ua/orelly/unix/unixnut/ch04_06.htm
  - http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/aixuser/usrosdev/list_c_builtin_cmds.htm

  - sh(1)
  - csh(1)
  - tcsh(1)

Useful Commands - File and Directory Related

Command	Purpose
cd	Change directory
ls	List a directory's content
pwd	Print working directory
mkdir	Make a new directory
rmdir	Remove existing directory
cat	Concatenate file
cp	Copy file
ln	Link two names to one file
mv	Move file
rm	Remove file
split	Split a file into n line chunks
stat	Display file status

Useful Commands - Select and file processing (1)

Command	Purpose
awk	Pattern scanning and processing language
cut	Select columns
diff	Compare and select difference in two files
grep	Select lines
head	Display first lines of a file
sed	Edit streams of data
tail	Select trailing lines
uniq	Select uniq lines
wc	Count characters, words or lines of a file
join	Join two files, matching row by row
sort	Sort and merge multiple files together
tr	Transform character

Useful Commands - Select and file processing (2)

- Example usage:
  - Look first few lines or last few lines
    - % head /var/log/message
    - % tail /var/log/message
  - Find the occurrence of certain pattern in file
    - % grep -l lwhsu *
      - Print the filename that has "lwhsu" as content
  - Print the line number when using grep
    - % grep -n lwhsu /etc/passwd
  - Ignore case-sensitive
    - % grep -i lwhsu /etc/passwd
      - List any line contains any combination of "lwhsu"
    - % ps auxww | grep lwhsu | wc -l
      - Count number of processes owned by lwhsu

Useful Commands - Select and file processing (3)

- List lwhsu's id, uid, home, shell in /etc/passwd
  - % grep lwhsu /etc/passwd | cut -f1,3,6,7 -d:
    - lwhsu:1001:/home/lwhsu:/bin/tcsh
- Cut out file permission and file name from ls output
  - % ls -l | grep -v ^total | cut -c1-12 -c45-

    -rw-r--r-- 2 a

    -rw-r--r-- 2 b

    -rw-r--r-- 3 chapter1

Useful Commands - Select and file processing (4)

- Use awk to generate the same behavior of cut
  - % awk -F: '{print $1 " " $6}' /etc/passwd
    - nobody /nonexistent
    - lwhsu /home/lwhsu
  - % ls -l | grep -v ^total | awk '{print $1 " " $10}'

    -rw-r--r-- a

    -rw-r--r-- b

    -rw-r--r-- chapter1

Useful Commands - Select and file processing (5)

- sort (useful arguments: -r, -u, -k, -n)
  - **-n** (numeric keys sorting),
  - **+n** (sorting using n-th field, counting from zero)
  - % ls -al | sort +4 -5 -r
  (% ls -al | sort -k 5,5 -r)
    - List directory contents and sort by file size decreasingly
  - % sort -t: +0 -1 /etc/passwd | grep -v ^#
  (%sort -t: -k 1,1 /etc/passwd | grep -v ^#)
    - List records in /etc/passwd increasingly by id

- tr - Translate characters
  - % tr "[A-Z]" "[a-z]" < file1 > file2
  - % grep lwhsu /etc/passwd | tr "[:]" "[\n]"
  - % tr -d "[\t]" < file1
    - Delete tabs in file1
  - % tr -s "[ ]" "[ ]" < file1
    - Delete multiple space in file1

Useful Commands - xargs

- xargs - construct argument list(s) and execute utility

  -n number

  -I replstr

  -J replstr

  -s size

  …

```
% ls
2.sh      3.csh     4.csh     4.sh      bsd1.ping   testin
% ls | xargs echo
2.sh 3.csh 4.csh 4.sh bsd1.ping testin
% ls | xargs -n1 echo
2.sh
3.csh
4.csh
4.sh
bsd1.ping
testin
```

```
% ls | xargs -I % -n1 echo % here %
2.sh here 2.sh
3.csh here 3.csh
4.csh here 4.csh
4.sh here 4.sh
bsd1.ping here bsd1.ping
testin here testin
```

```
% ls | xargs -J % -n1 echo % here %
2.sh here %
3.csh here %
4.csh here %
4.sh here %
bsd1.ping here %
testin here %
```

The Unix Way

- lots of little tools, each good at one thing
  - Combine yourself to achieve your goal
- Example
  - Quest: To get all cs97 student id/account/cname/ename
  - Hints:

    All user home dir are created by his/her student id.

    User command can get some useful info.

    %user lwhsu

    username: lwhsu studentID: 9455648 許立文 Li-Wen Hsu
  - Approach
    - % cd /u/cs/97
    - % ls # you will get all cs95 student id
    - % ls | xargs -n 1 # print student id each in one line
    - % ls | xargs -n 1 user # get result you want

Shell Programming

Shell variables (1)

- Assignment

	Bourne Shell	C Shell
Local variable	my=test	set my=test
Global variable	export my	setenv my test

  - Example:
    - $ export PAGER=/usr/bin/less
    - % setenv PAGER /usr/bin/less

    - $ current_month=`date +%m`
    - % set current_month =`date +%m`

Shell variables (2)

- Access
  - % echo "$PAGER"
  - % echo "${PAGER}"
  - Use {} to avoid ambiguity
    - % temp_name="haha"
    - % temp="hehe"
    - % echo $temp
      - hehe
    - % echo $temp_name
      - haha
    - % echo ${temp}_name
      - hehe_name
    - % echo ${temp_name}
      - haha

Shell variable operator (1)

BadCond : var is not set or the value is null
GoodCond : var is set and is not null

operator	description
${var:=value}	If BadCond, assign value to var
${var:+value}	If GoodCond, use value instead else null value is used but not assign to var
${var:-value}	If !GoodCond, use the value but not assign to var
${var:?value}	If !GoodCond, print value and shell exists

"Parameter Expansion" section in sh(1)

Shell variable operator (2)

- Ex:

```
#!/bin/sh

var1="haha"
echo "01" ${var1:+"hehe"}
echo "02" ${var1}
echo "03" ${var2:+"hehe"}
echo "04" ${var2}
echo "05" ${var1:="hehehe"}
echo "06" ${var1}
echo "07" ${var2:="hehehe"}
echo "08" ${var2}
echo "09" ${var1:-"he"}
echo "10" ${var1}
echo "11" ${var3:-"he"}
echo "12" ${var3}
echo "13" ${var1:?"hoho"}
echo "14" ${var1}
echo "15" ${var3:?"hoho"}
echo "16" ${var3}
```

- Result:

```
01 hehe
02 haha
03
04
05 haha
06 haha
07 hehehe
08 hehehe
09 haha
10 haha
11 he
12
13 haha
14 haha
hoho
```

Shell variable operator (3)

operator	description
${#var}	String length
${var#pattern}	Remove the smallest prefix
${var##pattern}	Remove the largest prefix
${var%pattern}	Remove the smallest suffix
${var%%pattern}	Remove the largest suffix

```
#!/bin/sh

var="Nothing happened end closing end"

echo ${#var}
echo ${var#*ing}
echo ${var##*ing}
echo ${var%end*}
echo ${var%%end*}
```

Results:

```
32
happened end closing end
end
Nothing happened end closing
Nothing happened
```

Predefined shell variables

- Environment Variables
- Other useful variables:

sh	csh	description
$#	$#	Number of positional arguments
$0	$0	Command name
$1, $2, ..	$1, $2, .. $argv[n]	Positional arguments
$*	$*, $argv[*]	List of positional arguments (useful in for loop)
$?	$?	Return code from last command
$$	$$	Process number of current command
$!	$!	Process number of last background command

test command

- test, [-- condition evaluation utility
  - test expression
  - [expression]
  - Test for:
    - File
    - String
    - Number

```
lucky7:~ -lwhsu- ls -i /bin/test /bin/[
164948 /bin/[    164948 /bin/test
```

- Test and return 0 (true) or 1 (false) in $?
  - % test -e News ; echo $?
    - If there exist the file named "News"
  - % test "haha" = "hehe" ; echo $?
    - Whether "haha" equal "hehe"
  - % test 10 -eq 11 ; echo $?
    - Whether 10 equal 11

test(1)

test command - File test

- -b file
  - True if file exists and is a block special file.
- -c file
  - True if file exists and is a character special file.
- -d file
  - True if file exists and is a directory.
- -e file
  - True if file exists (regardless of type).
- -f file
  - True if file exists and is a regular file.
- -g file
  - True if file exists and its set group ID flag is set.
- -k file
  - True if file exists and its sticky bit is set.
- -p file
  - True if file is a named pipe (FIFO).
- -r file
  - True if file exists and is readable.
- -s file
  - True if file exists and has a size greater than zero.
- -u file
  - True if file exists and its set user ID flag is set.
- -w file
  - True if file exists and is writable.
- -x file
  - True if file exists and is executable.
- -L file
  - True if file exists and is a symbolic link.
- -O file
  - True if file exists and its owner matches the effective user id of this process.
- -G file
  - True if file exists and its group matches the effective group id of this process.
- -S file
  - True if file exists and is a socket.
- file1 -nt file2
  - True if file1 exists and is newer than file2.
- file1 -ot file2
  - True if file1 exists and is older than file2.
- file1 -ef file2
  - True if file1 and file2 exist and refer to the same file.

```
lucky7:~ -lwhsu- ls -l a.sh
-rwxr-xr-x  1 root  users  - 376 Oct 28 09:48 a.sh
lucky7:~ -lwhsu- test -x a.sh ; echo $?
1
lucky7:~ -lwhsu- sudo test -x a.sh ; echo $?
0
```

test command - String test

- string
  - True if string is not the null string.
- -z string
  - True if the length of string is zero.
- -n string
  - True if the length of string is nonzero.
- s1 = s2
  - True if the strings s1 and s2 are identical.
- s1 != s2
  - True if the strings s1 and s2 are not identical.
- s1 < s2
  - True if string s1 comes before s2 based on the binary value of their characters.
- s1 > s2
  - True if string s1 comes after s2 based on the binary value of their characters.

Example:

```
% test "haha" \> "hehe"; echo $?
1
```

test command - Number test

- n1 -eq n2
  - True if the integers n1 and n2 are algebraically equal.
- n1 -ne n2
  - True if the integers n1 and n2 are not algebraically equal.
- n1 -gt n2
  - True if the integer n1 is algebraically greater than the integer n2.
- n1 -ge n2
  - True if the integer n1 is algebraically greater than or equal to the integer n2.
- n1 -lt n2
  - True if the integer n1 is algebraically less than the integer n2.
- n1 -le n2
  - True if the integer n1 is algebraically less than or equalto the integer n2.

Example:

```
% test 10 -gt 10 ; echo $?
1
% test 10 -ge 10 ; echo $?
0
```

test command - short format

- test command short format using [] or ()
  - % test "haha" = "hehe" ; echo $?

```
if test "haha" = "hehe" ; then
        echo "haha equals hehe"
else
        echo "haha does not equal hehe"
fi
```


```
if [ "haha" = "hehe" ] ; then
        echo "haha equals hehe"
else
        echo "haha doesn't equal hehe"
fi
```

```
if ( "haha" == "hehe" )  then
        echo "haha equals hehe"
else
        echo "haha doesn't equal hehe"
endif
```

test command - combination

- ! expression
  - True if expression is false.
- expression1 -a expression2
  - True if both expression1 and expression2 are true.
- expression1 -o expression2
  - True if either expression1 or expression2 are true.
  - The -a operator has higher precedence than the -o operator.
- (expression)
  - True if expression is true.

expr command

- Evaluate arguments and return 0 (true) or 1 (false) in $?
- Operators: +, -, *, /, %, =, !=, <, <=, >, >=
- Example:

```
% a=10
% a=`expr $a + 10` ; echo
$a
```

```
% set a=10
% set a=`expr $a + 10`;
echo $a
% @ a = $a + 10 ; echo $a
```

```
% a=10
% a=`expr $a \* 2`; echo
$a

% expr 4 = 5 ; echo $?
    ➔ 0
          1
% expr 5 = 5 ; echo $?
    ➔ 1
          0
```

Arithmetic Expansion:
$((expression))

if-then-else structure

```
if [ test conditions ] ;
then
        command-list
else
        command-list
fi
```

```
#!/bin/sh

a=10
b=12

if [ $a != $b ] ; then
        echo "$a not equal $b"
fi
```

```
if ( test conditions )
then
        command-list
else
        command-list
endif
```

```
#!/bin/tcsh

set a=10
set b=12

if ( $a != $b ) then
        echo "$a not equal $b"
endif
```

switch-case structure (1)

```
case $var in
    value1)
      action1
    ;;
    value2)
      action2
    ;;
    value3|value4)
      action3
    ;;
    *)
      default-action
    ;;
esac
```

```
switch ( $var )
    case value1:
      action1
      breaksw
    case value2:
      action2
      breaksw
    case value3:
    case value4:
      action3
      breaksw
    default:
      default-action
      breaksw
endsw
```

switch-case structure (2)

- Example

```
case $# in
    0)
        echo "Enter file name:"
        read argument1
        ;;
    1)
        argument1=$1
        ;;
    *)
        echo "[Usage] comm file"
esac
```

```
switch ($#)
    case 0:
        echo "Enter file
name:"
        read argument1
        breaksw
    case 1:
        argument=$1
        breaksw
    default:
        echo "[Usage] comm
file"
endsw
```

For loop

```
for var in var1 var2 …
do
      action
done
```

```
for dir in bin doc src
do
    cd $dir
    for file in *
    do
        echo $file
    done
    cd ..
done
```


```
foreach var (var1 var2 …)
      action
end
```

```
foreach dir ( bin doc src )
    cd $dir
    foreach file ( * )
        echo $file
    end
    cd ..
end
```

While loop

```
while […]
do
        action
done
```

```
month=1
while [ ${month} -le 12 ]
do
    echo $month
    month=`expr $month + 1`
done
```


```
while (…)
    action
end
```

```
set month=1
while ( ${month} <= 12 )
    echo $month
    @ month += 1
end
```

Until loop

```
until […]
do
        action
done
```

```
month=1
until [ ${month} -gt 12 ]
do
    echo $month
    month=`expr $month + 1`
done
```

Read from input

```
#!/bin/sh

echo "hello! How are you ?"

read line

if [ "$line" = "fine, thank you" ] ; then
    echo "right answer"
else
    echo "wrong answer, pig head"
fi
```

```
#!/bin/tcsh

echo "hello! How are you ?"

set line=$<

if ( "$line" == "fine, thank you" ) then
    echo "right answer"
else
    echo "wrong answer, pig head"
endif
```

Read from file

```
#!/bin/sh

exec 3< "file"

while read line <&3 ;
do
    echo "$line"
done
```

```
#!/bin/tcsh

set lc=1

while ( 1 )
    set line=`sed -n $lc,${lc}p "file"`
    if ( "$line" == "" ) then
        break
    endif

    echo $line
    @ lc ++
end
```

Shell functions (1)

- Define function

```
function_name ( ) {
  command_list
}
```

```
dir () {
        ls -l | less
}
```

- Removing function definition

```
unset function_name
```

- Function execution

```
function_name
```

- Function definition is local to the current shell

Shell functions (2)

Example:

```
#!/bin/sh

function1 () {
    result=`expr ${a:=0} + ${b:=0}`
}

a=5
b=10

function1

echo $result
```

$* and $@

- The difference between $* and $@
  - $* : all arguments are formed into a long string
  - $@ : all arguments are formed into separated strings

- Examples: test.sh

```
for i in "$*" ;  do
    echo $i
done

% test.sh 1 2 3
1 2 3
```

```
for i in "$@" ;  do
    echo $i
done

% test.sh 1 2 3
1
2
3
```

Parsing arguments (1)

- Use shift and getopt

```
#!/bin/sh
while [ "`echo $1 | cut -c1`" =
"-" ] ; do
   case $1 in
      -a|-b|-c)
         options="${options}
$1" ;;
      *)
         echo "$1: invalid
argument" ;;
    esac
    shift
done
```

```
#!/bin/sh
args=`getopt abo: $*`
if [ $? -ne 0 ]; then
    echo "Usage: getopt.sh [-a] [-b]
[-o file]"
    exit 2
fi
set -- $args
for i ; do
   case "$i" in
      -a|-b)
         echo flag $i set;
sflags="${i#-}$sflags";
         shift;;
      -o)
         echo oarg is "'"$2"'";
oarg="$2"; shift;
         shift;;
      --)
         shift; break ;;
    esac
done
echo "Do something about remainder
($*)"
```

Parsing arguments (2)

- Use getopts (recommended)

```
#!/bin/sh

while getopts abcf:o op
# The 'f' followed by ':' indicates the option takes an argument
do
   case $op in
      a|b|c)   echo "OPT=ABC";;
      f)    echo  $OPTARG;;   # $OPTARG is the following argument
      o)    echo  "OPT=o";;
      *)    echo  "Deafult";;
   esac
done
shift `expr $OPTIND - 1`      # Index of the first non-option argument
echo "The left arguments $*"
```

Handling Error Conditions

- Internal error
  - Caused by some command's failing to perform
    - User-error
      - Invalid input
      - Unmatched shell-script usage
    - Command failure
- External error
  - By the system telling you that some system-level event has occurred by sending signal

Handling Error Conditions - Internal Error

- Ex:

```
#!/bin/sh
UsageString="Usage: $0 -man=val1 -woman=val2"

if [ $# != 2 ] ; then
    echo "$UsageString"
else
    echo "ok!"
    man=`echo $1 | cut -c6-`
    woman=`echo $2 | cut -c8-`
    echo "Man is ${man}"
    echo "Woman is ${woman}"
fi
```

Handling Error Conditions - External Error (1)

- Using trap in Bourne shell
  - trap [command-list] [signal-list]
    - Perform command-list when receiving any signal in signal-list

```
trap ( rm tmp*; exit0) 1 2 3 14 15
```

```
trap "" 1 2 3
```

Ignore signal 1 2 3

Handling Error Conditions - External Error (2)

#	Name	Description	Default	Catch	Block	Dump core
1	SIGHUP	Hangup	Terminate			
2	SIGINT	Interrupt (^C)	Terminate			
3	SIGQUIT	Quit	Terminate			
9	SIGKILL	Kill	Terminate			
10	SIGBUS	Bus error	Terminate			
11	SIGSEGV	Segmentation fault	Terminate			
15	SIGTERM	Soft. termination	Terminate			
17	SIGSTOP	Stop	Stop			
18	SIGTSTP	Stop from tty (^Z)	Stop			
19	SIGCONT	Continue after stop	Ignore			

Handling Error Conditions - External Error (3)

- Using onintr in C shell
  - onintr label
    - Transfer control to label when an interrupt (CTRL-C) occurs
  - onintr -
    - Disable interrupt
  - onintr
    - Restore the default action

```
onitr catch
…
Do something in here
…
exit 0

catch:
        set nonomatch
        rm temp*
        exit 1
```

Debugging Shell Script

- Ex:

```
#!/bin/sh -x

var1="haha"
echo "01" ${var1:+"hehe"}
echo "02" ${var1}
echo "03" ${var2:+"hehe"}
echo "04" ${var2}
echo "05" ${var1:="hehehe"}
echo "06" ${var1}
echo "07" ${var2:="hehehe"}
echo "08" ${var2}
echo "09" ${var1:-"he"}
echo "10" ${var1}
echo "11" ${var3:-"he"}
echo "12" ${var3}
echo "13" ${var1:?"hoho"}
echo "14" ${var1}
echo "15" ${var3:?"hoho"}
echo "16" ${var3}
```

- Result:

```
+ var1=haha
+ echo 01 hehe
01 hehe
+ echo 02 haha
02 haha
+ echo 03
03
+ echo 04
04
+ echo 05 haha
05 haha
+ echo 06 haha
06 haha
+ echo 07 hehehe
07 hehehe
+ echo 08 hehehe
08 hehehe
+ echo 09 haha
09 haha
+ echo 10 haha
10 haha
+ echo 11 he
11 he
+ echo 12
12
+ echo 13 haha
13 haha
+ echo 14 haha
14 haha
hoho
```

Appendix A: Regular Expression

Regular Expression (1)

- Informal definition
  - Basis:
    - A single character "a" is a R.E.
  - Hypothesis
    - If r and s are R.E.
  - Inductive
    - Union: r + s is R.E
      - Ex: a + b
    - Concatenation: rs is R.E.
      - Ex: ab
    - Kleene closure: r* is R.E.
      - Ex: a*
- Example:
  - (1+2+3+4+5+6+7+8+9) (1+2+3+4+5+6+7+8+9)*
  - Letter: (A + B + C + … + Z + a + b + c + … + z)
  - Digit: (0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9)

Regular Expression (2)

- Pattern-matching
  - Characters, number and special operators

operator	Description
.	Match any single character
[]	Match any character found in []
[^]	Match any character not found in []
^	Match following R.E. only if occurs at start of a line
$	Match following R.E. only if occurs at end of a line
*	Match zero or more occurrence of preceding R.E.
?	Match zero or one occurrence of preceding R.E.
+	Match one or more occurrence of preceding R.E.
{m,n}	Number of times of preceding R.E. At least m times and at most n times
{m,}	Number of times of preceding R.E. At least m times.
{m}	Number of times of preceding R.E. Exactly m times.
\	Escape character

Regular Expression (3)

- Example:
  - `r.n`
    - Any 3-character string that start with r and end with n
      - r1n, rxn, r&n will match
      - r1xn, axn will not match
  - `..Z..`
    - Any 5-character strings that have Z as 3rd character
      - aeZoo, 12Zos will match
      - aeooZ, aeZooa will not match
  - `r[a-z]n`
    - Any 3-character strings that start with r and end with n and the 2nd character is a alphabet
      - rxn will match
      - r1n, r&n will not match
  - `[A-Za-z][0-9]`
    - Any 2-character strings that 1st character is a alphabet and 2nd is a number
      - A2 will match
      - 2c, 22, A2A will not match

Regular Expression (4)

- `^Windy`
  - Any string starts with Windy
    - Windy is great ➔ match
    - My Windy is great ➔ not match
- `^..Z..`
  - Any string ..Z.. and ..Z.. starts in a line
- `[E,e][N,n][D,d]$`
  - Any string ends with any combination of "end"
- `^$`
  - Match blank line
- `ZA*P`
  - "A" can be appeared 0 or more times
    - ZP, ZAP, ZAAP, …
- `ZAA*P`
    - ZAP, ZAAP, …
- `[A-Za-z] [A-Za-z]*`
  - String of characters
- `[+\-][0-9] [0-9]*`
  - Integer with a preceding + or -

Regular Expression (5)

- `[+\-]\{0,1\}[0-9][0-9]*`
  - Match any legal integer expression
- `[+\-]\{0,1\}[0-9][0-9]*\.\{0,1\} [0-9][0-9]*`
  - Match any real or integer decimal
- `[A-Z]\{2\}Z[0-9]\{2\}`
  - Two capital characters followed by Z followed by two numbers

Appendix B: sed and awk

sed - Stream EDitor (1)

- Syntax
  - sed -e "command" -e "command" … file
  - sed -f script-file file
    - Sed will read the file line by line and do the commands, then output to stdout
    - Ex:
      - sed -e '1,10d' -e 's/yellow/black/g' yel.dat
- Command format
  - [address1[,address2]]function[argument]
    - From address 1 to address 2
    - Do what action
- Address format
  - n ➔ line number
  - /R.E./ ➔ the line that matches R.E

sed - Stream EDitor (2)

- Example of address format
  - sed -e 10d
  - sed -e /man/d
  - sed -e 10,100d
  - sed -e 10,/man/d
    - Delete line from line 10 to the line contain "man"

sed - Stream EDitor Function: substitution (1)

- substitution
  - Syntax

    [address] s/pattern/replace/flags
  - Flags
    - **N**: Make the substitution only for the N'th occurrence
    - **g**: replace all matches
    - **p**: print the matched and replaced line
    - **w**: write the matched and replaced line to file

sed - Stream EDitor Function: substitution (2)

- Example:
  - sed -e 's/lwshu/lwhsu/2' file
  - sed -e 's/lwshu/lwhsu/g' file
  - sed -e 's/lwshu/lwhsu/p' file
  - sed -n -e 's/lwshu/lwhsu/p' file
  - sed -e 's/lwshu/lwhsu/w wfile' file

```
File Content:
I am jon
I am john
I am lwshu
I am lwshu
I am nothing
```

sed - Stream EDitor Function: delete

- delete
  - Syntax:
    [address]d
- Ex:
  - sed -e 10d
  - sed -e /man/d
  - sed -e 10,100d
  - sed -e 10,/man/d

sed - Stream EDitor
Function: append, insert, change

- append, insert, change
  - Syntax:

[address]a\ text	[address]i\ text	[address]c\ text

- Ex:
  - sed -f sed.src file

sed.src:

```
/lwhsu/i \
Meet lwhsu
```

file Content:

```
I am jon
I am john
I am lwhsu
I am lwhsu
I am nothing
```

Results:

```
I am jon
I am john
Meet lwhsu
I am lwhsu
Meet lwhsu
I am lwhsu
I am nothing
```

sed - Stream EDitor Function: transform

- transform
  - Syntax:

    [add1,addr2]y/xyz.../abc.../
- Ex:
  - sed -e 'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/' file
    - Lowercase to uppercase

sed - Stream EDitor Function: print

- print
  - Syntax:
    [addr1, addr2]p
- Ex:
  - sed -n -e '/^lwhsu/p'

awk

- Syntax
  - awk [-F fs] ['awk_program' | -f program_file] [data_file ……]
    - Read the file line by line and evaluate the pattern, then do the action if the test is true
    - Example:
      - `awk '{print "Hello World"}' file`
      - `awk '/lwhsu/ {print $1}' /etc/passwd`

Amy	32	0800123231	nctu.cs
$1	$2	$3	$4

- Program structure

```
pattern1    {action1}
pattern2    {action2}
…
```

awk - Pattern formats

- Relational expression
  - ==, <, <=, >, >=, !=, ~, !~
  - A ~ B means whether A contains substring B
- Regular Expression
  - awk '/[0-9]+/ {print "This is an integer" }
  - awk '/[A-Za-z]+/ {print "This is a string" }
  - awk '/^$/ {print "this is a blank line."}
- BEGIN
  - It will be true when the awk start to work before reading any data
    - awk 'BEGIN {print "Nice to meet you"}`
- End
  - It will be true when the awk finished processing all data and is ready to exit
    - awk 'END {print "Bye Bye"}

awk - Action format

File Content:
I am jon
I am john
I am lwhsu
I am lwhsu
I am nothing

- Print
- Assignment
- if(expression) statement [else statement2]
  - awk '/lwhsu/ { if($2 ~ /am/) print $1}' file
- while(expression) statement
  - awk 'BEGIN {count=0} /lwhsu/ {while (count < 3) {print count; count++}}' file
  - awk 'BEGIN {count=0} /lwhsu/ {while (count < 3) {print count; count++}; count=0}' file
- for (init ; test ; incr) action
  - awk '/lwhsu/ {for (i=0;i<3;i++) print i}' file

awk - built-in variables (1)

- $0, $1, $2, ...
  - Column variables
- NF
  - Number of fields in current line
- NR
  - Number of line processed
- FILENAME
  - the name of the file being processed
- FS
  - Field separator
- OFS
  - Output field separator

awk - built-in variables (2)

- Examples:
  - awk 'BEGIN {FS=":"} /lwhsu/ {print $3}' /etc/passwd
    - 1001
  - awk 'BEGIN {FS=":"} /^lwhsu/{print $3 $6}' /etc/passwd
    - 1001/home/lwhsu
  - awk 'BEGIN {FS=":"} /^lwhsu/{print $3 " " $6}' /etc/passwd
    - 1001 /home/lwhsu
  - awk 'BEGIN {FS=":" ;OFS="=="} /^lwhsu/{print $3 ,$6}' /etc/passwd
    - 1001==/home/lwhsu

Appendix C: Command History

Command History in csh/tcsh

- !n - exec previous command line n
- !-n - exec current command line minus n
- !! - exec last command (the same as !-1)
- !str - exec previous command line beginning with str
- !?str? - exec previous command line containing str

```
% history
9   8:30    nroff -man ypwhich.1
10  8:31    cp ypwhich.1 ypwhich.1.old
11  8:31    vi ypwhich.1
12  8:32    diff ypwhich.1.old ypwhich.1
13  8:32    history
% !?old?
```

Command History in csh/tcsh

- !!:n - use the nth word of previous command
- !!:m-n - select words m ~ n of previous command
- !!:* - use all arguments of previous command
- !!:s/str1/str2/ - substitute str1 with str2 in previous command

```
% history
15  8:35    cd /etc
16  8:35    ls HOSTS FSTAB
17  8:35    history
% cat !-2:*:s/HOSTS/hosts/:s/FSTAB/fstab
```