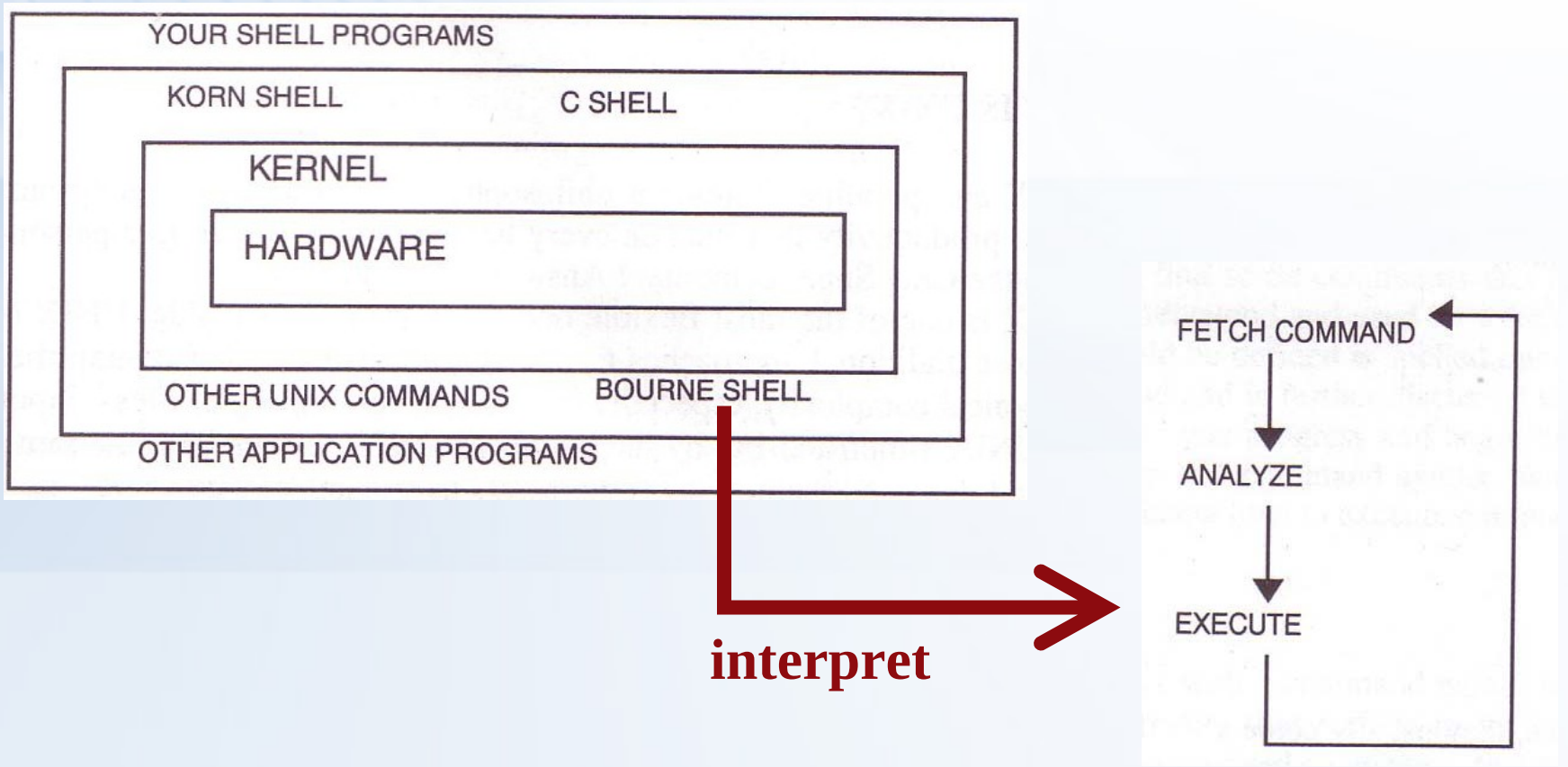
Shell and Shell Programming

The UNIX Shells

Shell	Originator	System Name	Prompt
Bourne	S. R. Bourne	/bin/sh	$
Korn	David Korn	/usr/local/ksh93	$
C	Bill Joy	/bin/csh	%

UNIX Kernel and shell



Shell Program (1)

- A collection of commands
- Ex:

```
#!/bin/sh

ls -al
touch aa
cp aa bb
```

Shell Program (2)

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

Startup files

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

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

– test1 test2 test3 test4 test-5 testmess

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

Shell Special Characters (2)

Char.	Purpose	Example
#	Start a shell comment	# this is a comment
;	Command separator	% ls test*; ls test?
\	(1) Escape character (2) Command continuation indicator	% touch test*; ls test* % ls \ > test*
&	Background execution	% make buildworld &

Shell Special Characters (3)

Char.	Purpose
${var}	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”
  - **Mon Oct 11 13:24:29 CST 2004**
  - **Now is $varname**
  - **Now is Mon Oct 11 13:24:29 CST 2004**

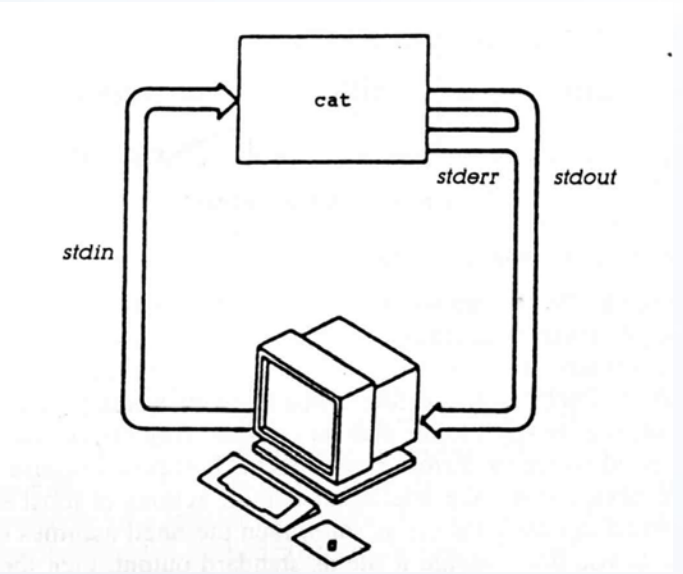
- % setenv varname2 `/bin/date`
- % echo $varname2
- % echo ‘Now is $varname2’
- % echo “Now is $varname2”

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



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**

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

Input/Output Redirection (4)

> Example

- % 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**

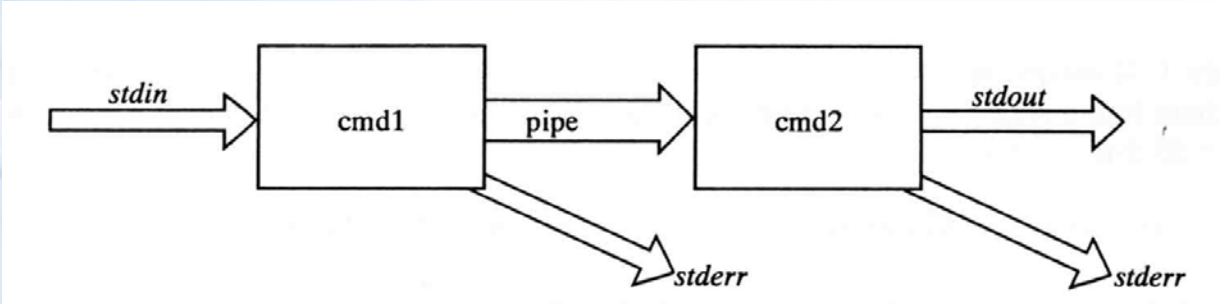
- % sed –e "s/SHELL/shell/g" < chapter1 2>&1
  - **Both stdout and stderr go to terminal**

- % 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

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
  - % command arguments 2>&1 | nextcommand
  - % command arguments |& nextcommand
    - **Merge stderr with stdout and pipe to next command**

Input/Output Redirection (6)

- % exec 4>&- # close file descriptor 4
- % exec 1>&- # close stdin

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 + 0)

Regular Expression (2)

- Pattern-matching
  - Contain letters, 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.
\{m,n\} \{m,\} \{m\}	Number of times of preceding R.E. At least m times and at most n 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

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

Commands – Select and file processing Related (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

Commands – Select and file processing Related (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 tytsai ***
    - > Print the filename that has “tytsai” as content
- Print the line number when using grep
  - **% grep –n tytsai /etc/passwd**
- Ignore case-sensitive
  - **% grep -i tytsai /etc/passwd**
    - > List any line contains any combination of “tytsai”
  - **% ps auxww | grep ^tytsai | wc –l**
    - > Count number of processes owned by tytsai

Commands – Select and file processing Related (3)

- List tytsai's id, uid, home, shell in /etc/passwd
  - **% grep tytsai /etc/passwd | cut –f1,3,6,7 –d:**
    > tytsai:1001:/home/tytsai:/bin/tcsh

- Cut out file permission and file name from ls output
  - **% ls -l | grep –v ^total | cut -c1-12 -c45-**
    > drwxr-xr-x GNUstep/
    > drwx------ Mail/
    > drwx------ News/

Commands – Select and file processing Related (4)

- Use awk to generate the same behavior of cut
  - **% awk –F: '{print $1 " " $6}' /etc/passwd**
    > nobody /nonexistent
    > tytsai /home/tytsai

Commands – Select and file processing Related (5)

- Sort
  - **% ls –al | sort +4 -5 –r**
    > List directory contents and sort by file size decreasingly
  - **% sort –t: +0 -1 /etc/passwd | grep –v ^#**
    > List records in /etc/passwd increasingly by id
- Translate characters
  - **% tr “[A-Z]” “[a-z]” < file1 > file2**
  - **% grep tytsai /etc/passwd | tr "[:]" "[\n]“**
  - **% tr –d “[\t]” < file1**
    > Delete tab in file1
  - **% tr –s “[ ]” “[ ]” < file1**
    > Delete multiple space in file1

Built-in Shell Commands (1)

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

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

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

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

Built-in Shell Commands (5)

- 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

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 ambiguous
    - **% 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

Shell variable operator (2)

> Ex:

```
#!/bin/sh

var1="haha"

echo ${var1:+"hehe"}
echo ${var1}
echo ${var2:-"wow"}
echo ${var2="what"}
echo ${var2}
```

Results:

```
hehe
haha
wow
what
what
```

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 (1)

sh	csh	description
HOME	home	User's home
MAIL	MAIL	User's mail file
PATH	PATH	Search path
PS1	prompt	Primary prompt string
PS2		Secondary prompt string
IFS		Internal field separators
	history	Number of history commands

Predefined shell variables (2)

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 (1)

- test command can test
  - File
  - String
  - Number
- 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 command – File test

-b file
測試是否檔案爲 block device file

-c file
測試是否檔案爲 character device file

-d file
測試是否檔案爲 directory

-e file
測試檔案是否存在

-f file
測試檔案是否存在，並且檔是是否爲 regular file

-g file
測試檔案是否 set-group-id permission 是否有打開

-h file
測試檔案是否爲 symbolic link

-k file
測試檔案 stricky bit 是否有打開

-p file
測試檔案是否爲 pile

-r file
測試檔案是否爲 readonly (by your script)

-s file
測試檔案是否存在，並且不是空的

-S file
測試檔案是否爲 socket file

-u file
測試檔案是否 set-user-id permission 是否有打開

-w file
測試檔案是否可以寫入 (by your script)

-x file
測試檔案是否可以執行 (by your script)

-O file
測試檔案是否爲你所擁有

-G file
測試檔案是否被你的 group 所擁有

-N file
測試檔案是否有新資料

f1 -nt f2
測試 f1 檔案是否比 f2 檔案還新

f1 -ot f2
測試 f1 檔案是否比 f2 檔案還舊

f1 -ef f2
測試 f1 檔案是否爲 f2 檔案的 hard link

Test command – String test

```
-z s
        測試是否爲 empty string

-n s
        測試是否不是 empty string

s1 = s2
        測試 s1 是否跟 s2 相同

s1 != s2
        測試 s1 是否跟 s2 不相同

s1 \> s2
        測試 s1 是否大於 s2

s1 \< s2
        測試 s1 是否小於 s2
```

- Example
  - % test "haha" \> "hehe"; echo $?
    - **1**

Test command – Number test

```
n1 -eq n2
        是否相等

n1 -ne n2
        是否不相等

n1 -lt n2
        是否 n1 小於 n2

n1 -le n2
        是否 n1 小於或等於 n2

n1 -gt n2
        是否 n1 大於 n2

n1 -ge n2
        是否 n1 大於或等於 n2
```

- Example
  - % test 10 –gt 10 ; echo $?
    - **1**
  - % test 10 –ge 10 ; echo $?
    - **0**

test command (2)

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

```
if test “haha” = “hehe” ; then
        echo “haha equals hehe”
else
        echo “haha do 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
```

expr command (1)

- Evaluate arguments and return 0 (true) or 1 (false) in $?
  - % a=10
  - % a=`expr $a + 10` ; echo $a
  - % set a=10
  - % set a=`expr $a + 10`; echo $a
  - % @ a = $a + 10 ; echo $a

expr command (2)

```
- +
        unary 正, 負

!
        logical negation

~
        bitwise negation

+ - * / %
        加, 減, 乘, 除 與取餘數

 << >>
        bitwise shift

== !=
        相等與不相等

 & ^ |
        bitwise and xor or

 && ||
        logical and or

 expr ? expr :
        conditional expression

 = *= /= %=
        assignment
```

- Example
  - % a=10
  - % a=`expr $a \*2`; echo $a
  - % a=5
  - % b=5
  - % cc=`expr $a = = $b`

if-then-else structure

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

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

```
#!/bin/sh

a=10
b=12

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

```
#!/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
```

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
```

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:

```
UsageString="Usage: ./command -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 occurs**
- onintr –
  - **Disable interrupt**
- onintr
  - **Restore the default action**

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

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

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: do the substitution for the n-th match**
    - **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)

- Ex:
  - sed –e ‘s/tytsai/TYTSAI/2’ file
  - sed –e ‘s/tytsai/TYTSAI/g’ file
  - sed –e ‘s/tytsai/TYTSAI/p’ file
  - sed –n –e ‘s/tytsai/TYTSAI/p’ file
  - sed –e ‘s/tytsai/TYTSAI/w wfile’ file

File Content:
I am jon
I am john
I am tytsai
I am tytsai
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\
[address]i \
[address]c \
```

- Ex:
  - sed –f sed.src file

Content of sed.src

```
/tytsai/i \
Meet tytsa, Hello
```

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

Results:
I am jon
I am john
How are your?
I am tytsai
How are your?
I am tytsai
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 –e ‘/^tytsai/p’

sed – Stream EDitor
other commands

> | | r | w | y | ! | n | q | = | N | D | P
| h | H | g | G | x | b | t |

awk

- Syntax
  - awk Pattern {Action} file
  - awk –f script-file file
    - **awk will read the file line by line and evaluate the pattern, then do the action if the test is true**
    - **Ex:**
      - > awk '{print Hello World}' file
      - > awk '/MA/ {print $1}' list

Amy	32	0800995995	nctu.csie
$1	$2	$3	$4

awk – Pattern format

> 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 tytsai
I am tytsai
I am nothing

> Actions

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

- Ex:
  - awk ‘BEGIN {FS=“:”} /tytsai/ {print $3}’ /etc/passwd
    - **1001**
  - awk 'BEGIN {FS=":"} /^tytsai/{print $3 $6}' /etc/passwd
    - **1001/home/tytsai**
  - awk 'BEGIN {FS=":"} /^tytsai/{print $3 " " $6}' /etc/passwd
  - awk 'BEGIN {FS=":" ;OFS="=="} /^tytsai/{print $3 ,$6}' /etc/passwd
    - **1001==/home/tytsai**


example

檢查某一台機器是否當掉 (1)

- Useful details
  - /sbin/ping –c 3 ccbsd1

```
PING ccbsd1 (140.113.209.61): 56 data bytes
64 bytes from 140.113.209.61: icmp_seq=0 ttl=64 time=0.101 ms
64 bytes from 140.113.209.61: icmp_seq=1 ttl=64 time=0.092 ms
64 bytes from 140.113.209.61: icmp_seq=2 ttl=64 time=0.104 ms

--- ccbsd1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.092/0.099/0.104/0.005 ms
```

檢查某一台機器是否當掉（2）

```
# [Usage] isAlive.sh ccbsd1
Usage="[Usage] isAlive.sh host”
temp="$1.ping"
Admin="tytsai"

/sbin/ping –c 10 $1 | /usr/bin/grep 'transmitted' > $temp
Lost=`awk –F" " '{print $7}' $temp ` | awk –F"%" '{print $1}' `

if [ $Lost –ge 50 ] ; then
        mail –s  "$1 failed" $Admin < $temp
fi
```