Shell Programming

tsaimh (2022-2025, CC BY-SA)
lctseng (2019-2021, CC BY-SA)
? (1996-2018)

國立陽明交通大學資工系資訊中心
Information Technology Center, Department of Computer Science, NYCU

Why Shell Programming

- Just like coding in C/C++
  - Variables
  - If-else
  - Loop
  - Read from keyboard
  - Output to screen
  - Execute other commands
    - In C/C++: system()
- Using shell syntax

Outline

- Variable pre-operations
- args, argc in Shell Scripts
- Arithmetic and Logics
  - Test commands
- Control Structures: if-else, switch-case, for/while loops
- Input/output: Read from keyboard
- Defining Functions & Parsing Arguments
- Error Handling and Debug tool (sh -x)
- Regular Expression
- Advanced scripting: sed and awk
- A Shell Script Sample: Failure Detection on Servers

Bourne Shell

- We use Bourne Shell in this slide
- Check your login shell

```
$ echo $SHELL
/bin/tcsh
```

- Print the current shell

```
$ ps -p $$
3463  0  S    0:00.01 tcsh
$ sh
$ ps -p $$
3474  0  S    0:00.00 sh
```

Sample script

- Print "Hello World" 3 times

```
#!/bin/sh
# ^ shebang: tell the system which interpreter to use

for i in `seq 1 3` ;  do
    echo "Hello world $i" # the body of the script
done
```

- Output

```
$ chmod +x test.sh  # grant execution permission
$ ./test.sh         # execute the script. Must specify the directory(./)
```

Executable script

- Shebang (#!), or called Shabang
  - Sharp (#) + Bang (!)
    - or Hash Bang
  - Specify which interpreter is going to execute this script
  - Many interpreted language uses # as comment indicators
  - The first widely known appearance of this feature was on BSD

Executable script

- Shebang examples
  - `#!/bin/sh`
  - `#!/bin/sh -x`
  - `#!/bin/bash`
  - `#!/usr/local/bin/bash`
  - `#!/usr/bin/env bash`
  - `#!/usr/bin/env python`
- Execution
  - $ sh test.sh
    - Can execute without shebang
  - $ chmod a+x test.sh
  - $ ./test.sh

Shell variables (1)

- Assignment

	Syntax	Scope
Variable	my=test	Process
Local variable	local my=test	Function
Environment variable	export my	Process and sub-process

- Example

```
$ export PAGER=/usr/bin/less
$ current_month=`date +%m`
$ myFun() { local arg1="$1" }
```

Shell variables (2)

- There are two ways to call variable
  - $ echo "$PAGER"
  - $ echo "${PAGER}" <= Why?
    - Use {} to avoid ambiguity
- Example

```
$ temp_name="haha" && temp="hehe" # No Space Beside "="
$ echo $temp
hehe
$ echo $temp_name
haha
$ echo ${temp}_name
hehe_name
$ echo ${temp_name}
haha
```

Quotation marks

Quotes	Description	Example
' '	Single quote, Preserves the literal value of each character within the quotes	`$ echo 'echo $USER'` `echo $USER`
" "	Double quote, Parse special character, like: $ ` \	`$ echo "echo $USER"` `echo tsaimh`
` `	Back quotes, The stdout of the command	`$ echo `echo $USER`` `tsaimh` `$ echo now is `date`` `now is Sat Aug 15 03:56:54 CST 2022`

Shell variable operator (1)

- sh(1): Parameter Expansion

Operator	Description
${var:=value}	If "Bad", use the **given value** and assign to var.
${var:+value}	If "Good", use the **given value**. Otherwise, null is used but not assign to var. => Replace if "Good", not assign to var.
${var:-value}	If "Good", use the **value of var**. Otherwise, use the **given value** but not assign to var => Replace if "Bad", not assign to var.
${var:?value}	If "Bad", print given value (stderr) and shell exits (The command stops immediately).

  - Good: var is set and is not null.
  - Bad: var is not set or the value is null.
    - Bad == not Good

Shell variable operator (2)

- Script

```
#!/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
(program exited)
```

Shell variable operator (3)

Operator	Description	
`${#var}`	String length	These operators do not change the value of var
`${var#pattern}`	Remove the smallest prefix	
`${var##pattern}`	Remove the largest prefix	
`${var%pattern}`	Remove the smallest suffix	
`${var%%pattern}`	Remove the largest suffix	

- Script

```
#!/bin/sh
var="Nothing happened end closing end"
echo ${#var}
echo ${var#*ing}
echo ${var##*ing}
echo ${var%end*}
echo ${var%%end*}
```

- Result

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

Predefined shell variables

- Environment Variables
- Other useful variables
  - Similar to C program's "`int main(argc, argv)`" – arguments of program
  - e.g. `ls -a ~`

Predefined shell variables

- Example:
  - `ls -a ~`

sh	Description
`$#`	Number of positional arguments (start from 0)
`$0`	Command name (Ex: What command user exec your script)
`$1, $2, ..`	Positional arguments
`$* / $@`	• List of positional arguments (useful in for loop) • ${*:2} : Get the list of argument after $2
`$?`	Return code from last command
`$$`	Process number of current command (pid)
`$!`	Process number of last background command

Usage of $* 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 "In loop: $i"
done

% test.sh 1 2 3
In loop: 1 2 3
```

```
for i in "$@" ; do
  echo "In loop: $i"
done
% test.sh 1 2 3
In loop: 1
In loop: 2
In loop: 3
```

The "test" command

- Checking file status, string, numbers, etc
- test(1)
  - test expression
  - [expression]
- 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

- -e file
  - True if file exists (regardless of type)
- -s file
  - True if file exists and has size greater than zero
- -d file
  - True if file exists and is a directory
- -f file
  - True if file exists and is a regular file

Test command – File test

- -L file
  - True if file exists and is a symbolic link
- -r file
  - True if file exists and is readable
- -w file
  - True if file exists and is writable
- -x file
  - True if file exists and is executable

Test command – File test

- 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

Test command – String test

- -z string
  - True if the length of string is zero
- -n string
  - True if the length of string is nonzero
- string
  - True if string is not the null string
- s1 = s2 (though some implementation recognize ==)
  - 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 (lexicographical order)
- s1 > s2
  - True if string s1 comes after s2 based on the binary value of their characters

Test command – Number comparison

- Number comparison with “>”

```
$ test 14 > 123 ; echo $?
0 # True
```

- The correct way is to use “-gt” (greater-than)

```
$ test 14 -gt 123 ; echo $?
1 # False
```

Test command – Number test

- n1 -eq n2 **==, !=, >, <, >=, <= fashion does not apply here**
  - 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 equal to the integer n2

Test Command – Combination

- ! expression
  - True if expression is false.
  - $ [! A == B] => Test expression, invert the internal result
  - $! [A == B] => Invert the whole test command result
- expression1 -a expression2
  - True if both expression1 and expression2 are true.
  - $ [A == B –a C == D]
- expression1 -o expression2
  - True if either expression1 or expression2 are true.
  - The -a operator has higher precedence than the -o operator.
  - $ [A == B –o C == D]

Test Command – Combination Example

- ! ["A" = "A" -o 1 -eq 1]
  - false
- [! "A" = "A" -o 1 -eq 1]
  - true

Test Command – In Script

- Add space beside = <= != [ ]…
  - $ [A=B] # error
  - $ [A=B] # error
  - $ [A = B] # error
- If the var may be null or may not be set, add ""
  - $ [$var = "A"] may be parsed to [= "A"] and cause syntax error!!
  - $ ["$var" = "A"] become ["" = "A"]

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

Logical Operations with Short Circuit Evaluation (1)

- Another way to combine test results
- AND, OR, NOT (&&, ||, !)

```
[ 1 -eq 2 ] || [ 1 -eq 1 ] ; echo $?
0
[ 1 -eq 1 ] || [ 1 -eq 2 ] ; echo $?
0
[ 1 -eq 1 ] && [ 1 -eq 2 ] ; echo $?
1
```

```
[ 1 -eq 2 ] && [ 1 -eq 1 ] ; echo $?
1
! [ 1 -eq 2 ] ; echo $?
0
[ ! 1 -eq 2 ] ; echo $?
0
```

Logical Operations with Short Circuit Evaluation (2)

- $ expr1 && expr2
  - if expr1 is false then expr2 won't be evaluated
- $ expr1 || expr2
  - if expr1 is true then expr2 won't be evaluated
- Ex:
  - $ [-e SomeFile] && rm SomeFile
  - $ checkSomething || exit 1

Arithmetic Expansion

```
echo $(( 1 + 2 ))
a=8
a=$(( $a + 9 ))
a=$(( $a + 17 ))
a=$(( $a + 9453 ))
echo $a
```

```
3
// a=8
// a=17
// a=34
// a=9487
9487
```

if-then-else structure

```
if [ test conditions ] ; then
      command-list
elif [ test conditions ] ; then
      command-list
else
      command-list
fi
# Or in one line
if [ a = a ]; then echo "Yes"; else echo "No"; fi
```

switch-case structure (1)

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

```
case $sshd_enable in
    [Yy][Ee][Ss])
        action1
    ;;
    [Nn][Oo])
        action2
    ;;
    *)
        ???
    ;;
esac
```

For loop

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

```
a=""
for var in `ls`; do
      a="$a $var"
done
echo $a
```

```
for i in A B C D E F G; do
      mkdir $i;
done
```

While loop

```
while [ expression ] ; do
      action
done

break
continue

while read name ; do
      echo "Hi $name"
done
```

Read from stdin

```
#!/bin/sh
echo -n "Do you want to 'rm -rf /' (yes/no)? "
read answer # read from stdin and assign to variable
case $answer in
      [Yy][Ee][Ss])
            echo "Hahaha"
      ;;
      [Nn][Oo])
            echo "No~~~"
      ;;
      *)
            echo "removing..."
      ;;
esac
```

Create tmp file/dir

- TMPDIR=`mktemp –d tmp.XXXXXX`
- TMPFILE=`mktemp ${TMPDIR}/tmp.XXXXXX`
- echo "program output" >> ${TMPFILE}

functions (1)

- Define function

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

- Removing function definition

```
unset function_name
```

- Function execution

```
function_name
```

- Function definition is local to the current shell
- Define the function before first use

functions (2) - scoping

```
func() {
    # global variable
    echo $a
    a="bar"
}
a="foo"
func
echo $a
```

```
foo
bar
```

```
func() {
        # local variable
        local a="bar"
        echo $a
}
a="foo"
func
echo $a
```

```
bar
foo
```

functions (3) - arguments check

```
func() {
    if [ $# -eq 2 ] ; then
        echo $1 $2
    else
        echo "Wrong"
    fi
}
func
func hi
func hello world
```

```
Wrong
Wrong
hello world
```

functions (4) - return value

```
func() {
    if [ $# -eq 2 ] ; then
        return 2
    else
        return 0
    fi
}
func
echo $?
func hello world
echo $?
```

```
0
2
```

Scope

- Local var can only be read and written inside the function.
- Subprocess can only read the environment variable, the modification of the variable will NOT be effective to the current process. (Subprocess may include some PIPE execution)
- If something wrong, try to print every variable.

```
#!/bin/sh
a=10
export b=20
cat test.sh | while read line; do
    echo "$a $b $line"
    b=$((b+1))
done
echo b is $b
```

test.sh

```
10 20 #!/bin/sh
10 21 a=10
10 22 export b=20
10 23 cat test.sh | while read line; do
10 24 echo "$a $b $line"
10 25 b=$((b+1))
10 26 done
10 27 echo b is $b
b is 20
```

Parsing arguments

- Use getopts

```
#!/bin/sh
echo "Initial OPTIND: $OPTIND"
while getopts abcf: op ; do
    echo "${OPTIND}-th arg"

    case $op in
        a|b|c)
            echo "one of ABC" ;;
        f)
            echo $OPTARG ;;
        *)
            echo "Default" ;;
        esac
done
```

```
$ ./test.sh -a -b -c -f hi
Initial OPTIND: 1
2-th arg
one of ABC
3-th arg
one of ABC
4-th arg
one of ABC
6-th arg
hi
```

- ":" means additional arg.
- $OPTARG: content of additional arguments
- $OPTIND: index of the next argument
  - Need manually reset for the second call

Handling Error Conditions

- Internal error
  - Program crash
  - Failing to perform sub commands
  - Invalid input
  - Syntax error
- External error
  - Signal from OS
    - The system telling you that some system-level event has occurred
  - Ctrl+C
    - SIGINT

Handling Error Conditions – Internal Error

cut: you must specify a list of bytes, characters, or fields
Try 'cut --help' for more information.

- Example:
  - Handling the errors by yourself

program name

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

if [ $# != 2 ] ; then
        echo "$UsageString"
else
    echo "ok!"
    man=`echo $1 | cut -c 6-`
    woman=`echo $2 | cut -c 8-`
    echo "Man is ${man}"
    echo "Woman is ${woman}"
fi
```

How about c but not -c?

Handling Error Conditions – External Error (1)

- Using trap in Bourne shell
  - To handle events like Ctrl+C (SIGINT, signal number is 2)
  - trap [command-list] [signal-list]
    - Perform command-list when receiving any signal in signal-list

```
trap "rm tmp*; exit 0" 1 2 3 14 15
trap "" 1 2 3 # Ignore signal 1 2 3
```

Handling Error Conditions – External Error (2)

Catch: perform something when trapped
Block: prevent system actions

#	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	✓	✗	✗

Debugging Shell Script —Debug tools in sh

- Example:

Debug Mode

```
#!/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}
```

Print out the **substitution results**

- 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
```

ShellCheck

- Find potential bugs in your shell scripts
  - https://www.shellcheck.net/
- In FreeBSD
  - devel/hs-ShellCheck
  - pkg install hs-ShellCheck

Regular Expression

Pattern Matching

國立陽明交通大學資工系資訊中心

Information Technology Center, Department of Computer Science, NYCU

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*

Regular Expression (2)

- Pattern-matching
  - Special operators

operator	Description
.	Any single character
[ ]	Any character in []
[^]	Any character not in []
^	start of a line
$	end of a line
*	zero or more
?	zero or one
+	one or more
{m,n}	At least m times and at most n times
{m,}	At least m times.
{m}	Exactly m times.
\	Escape character

Regular Expression (3)

- Examples
  - 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, aeZoom will not match
  - r[a-z]n
    - Any 3-character string that start with r and end with n and the 2nd character is an alphabet
      - rxn will match
      - r1n, r&n will not match

Regular Expression (4)

- Examples
  - ^John
    - Any string starts with John
      - John Snow -> will match
      - Hi John -> will not match
  - [Ee][Nn][Dd]$
    - Any string ends with any combination of "end"
  - [A-Za-z0-9]+
    - String of characters

Regular Expression (5)

- Utilities using RE
  - grep
  - awk
  - sed
  - find
- Different tools, different RE
  - BRE (Basic)
  - ERE (Extended)
  - PCRE (Perl Compatible)
  - https://en.wikipedia.org/wiki/Regular_expression#Standards

Advanced scripting - sed and awk

Details on using sed and awk...

國立陽明交通大學資工系資訊中心
Information Technology Center, Department of Computer Science, NYCU

sed – Stream EDitor (1)

- sed(1)
  - sed -e "command" -e "command"… file
  - sed -f script-file file
    - Sed will (1) read the file line by line and (2) do the commands, then (3) output to stdout
    - e.g. `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)

- Address format
  - 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

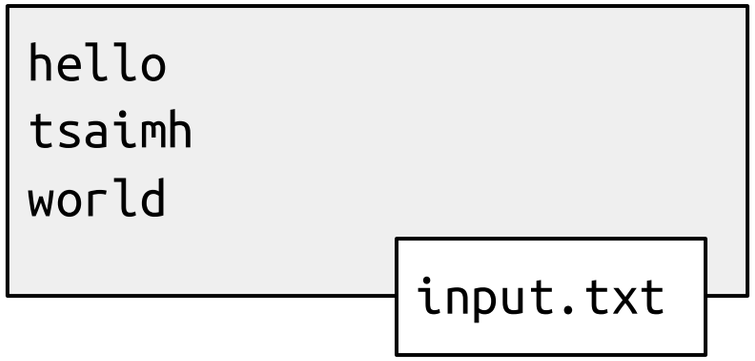
Function: print (1)

- print
  - Syntax:
    - [addr1, addr2]p
- Ex:
  - sed -n -e '/^tsaimh/p' # Print out the lines that begins with tsaimh

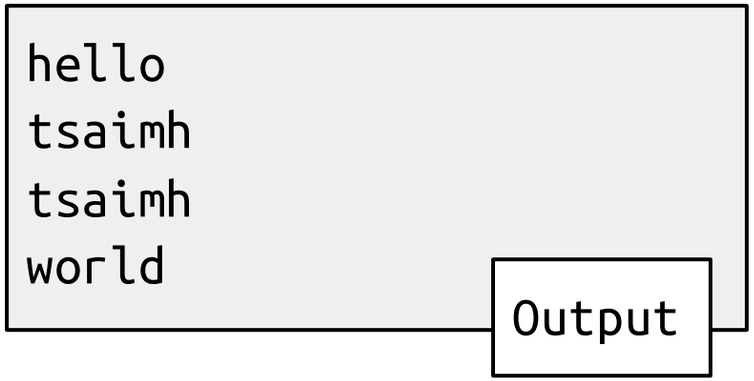
-n: By default, each line of input is echoed to the standard output after all of the commands have been applied to it. The -n option suppresses this behavior.

sed – Stream Editor

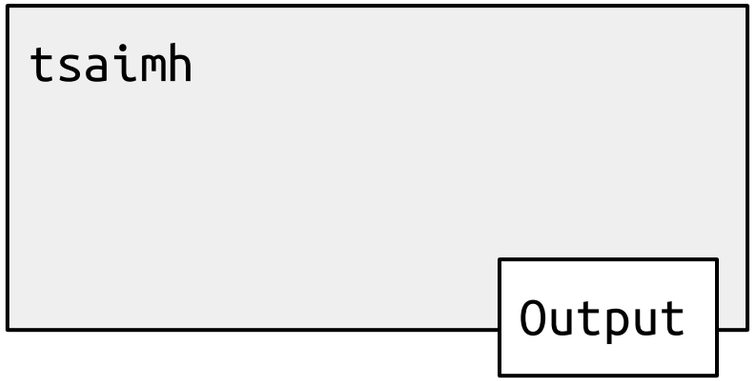
Function: print (2)



- sed -e '/^tsaimh/p' input.txt



- sed -n -e '/^tsaimh/p' input.txt



sed – Stream Editor

Function: substitution (1)

- substitution
  - Syntax
    - 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 a file

sed – Stream Editor

Function: substitution (2)

- Example:
  - `sed -e 's/tsaimh/TSAIMH/2' file.txt`

```
I am jon
I am john
I am tsaimh
I am tsaimh
I am nothing
```

  - `sed -e 's/tsaimh/TSAIMH/g' file.txt`

```
I am jon
I am john
I am TSAIMH
I am TSAIMH
I am nothing
```

file.txt

```
I am jon
I am john
I am tsaimh
I am tsaimh
I am nothing
```

sed – Stream Editor

Function: substitution (3)

- Example:
  - `sed -e 's/tsaimh/TSAIMH/p' file.txt`

```
I am jon
I am john
I am TSAIMH
I am TSAIMH
I am TSAIMH
I am TSAIMH
I am nothing
```

  - `sed -n -e 's/tsaimh/TSAIMH/p' file.txt`

```
I am TSAIMH
I am TSAIMH
```

  - `sed -e 's/tsaimh/TSAIMH/w wfile' file.txt`
  - `cat wfile`

```
I am TSAIMH
I am TSAIMH
```

file.txt

```
I am jon
I am john
I am tsaimh
I am tsaimh
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

- Function
  - append
    - append after the line
  - insert
    - insert before the line
  - change
    - replace whole line
- Example:
  - sed -f sed.src file.txt

sed.src

```
/tsaimh/i \
Meet tsaimh, Hello
```

file.txt

```
I am jon
I am john
I am tsaimh
I am tsaimh
I am nothing
```

Result

```
I am jon
I am john
Meet tsaimh, Hello
I am tsaimh
Meet tsaimh, Hello
I am tsaimh
I am nothing
```

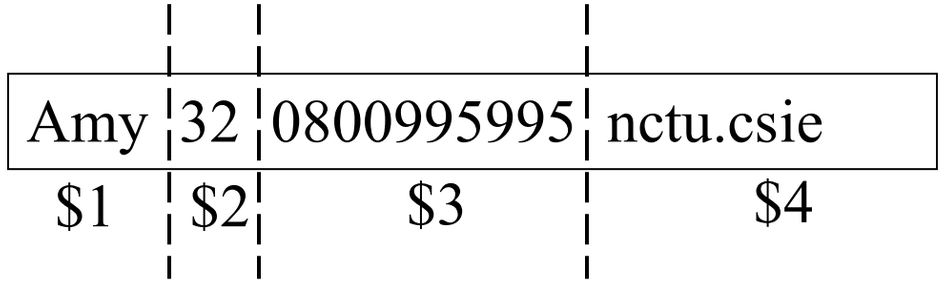
sed – Stream EDitor

Example: split lines into multiple files

- `cat host.txt | sed -e '/cs/w cs_host.txt' -e '/google/w google_host.txt'`

awk

- awk(1)
  - awk [-F fs] ['awk_program' | -f program_file] [data_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 '{print $1}' file
- Program structure
  - pattern { action }
  - missing pattern means always matches
  - missing { action } means print the line



awk – Pattern formats

- pattern formats
  - 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
    - before reading any data
      - awk ' BEGIN {print "Nice to meet you"}'
  - END
    - after the last line is read
      - awk ' END {print "Bye Bye"}'

awk – action format

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

variable usage: no need for "$"

reset count after printing

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, set by -F
- OFS
  - Output field separator

awk – built-in variables (2)

- Ex:
  - awk 'BEGIN {FS=":"} /tsaimh/ {print $3}' /etc/passwd
    - 1002
  - awk 'BEGIN {FS=":"} /^tsaimh/{print $3 $6}' /etc/passwd
    - 1002/home/tsaimh
  - awk 'BEGIN {FS=":"} /^tsaimh/{print $3 " " $6}' /etc/passwd
    - 1002 /home/tsaimh
  - awk 'BEGIN {FS=":" ;OFS="=="} /^tsaimh/{print $3 ,$6}' /etc/passwd
    - 1002==/home/tsaimh

```
tsaimh:*:1002:20:Meng-Hsun Tsai:/home/tsaimh:/bin/tcsh
```

Shell Script Examples

國立陽明交通大學資工系資訊中心

Information Technology Center, Department of Computer Science, NYCU

check alive(1)

- ping

```
$ /sbin/ping -c 3 bsd1.cs.nctu.edu.tw

PING bsd1.cs.nctu.edu.tw (140.113.235.131): 56 data bytes
64 bytes from 140.113.235.131: icmp_seq=0 ttl=64 time=0.044 ms
64 bytes from 140.113.235.131: icmp_seq=1 ttl=64 time=0.068 ms
64 bytes from 140.113.235.131: icmp_seq=2 ttl=64 time=0.056 ms

--- bsd1.cs.nctu.edu.tw ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.044/0.056/0.068/0.010 ms
```

check alive(2)

```
#!/bin/sh
# [Usage] isAlive.sh bsd1.cs.nctu.edu.tw

Usage="[Usage] $0 host"
temp="$1.ping"
Admin=”tsaimh fs"
count="3"

if [ $# != 1 ] ; then
  echo $Usage
else
  /sbin/ping -c ${count:=10} $1 | /usr/bin/grep 'transmitted' > $temp
  Lost=`awk -F" " '{print $7}' $temp | awk -F"." '{print $1}' `

  if [ ${Lost:=0} -ge 50 ] ; then
    mail -s  "$1 failed" $Admin < $temp
  fi
  /bin/rm $temp
fi
```

default 10 times

Grep "tran…"
write to the temp file

Mail and del. $temp

- awk on $temp using space as delimeter
- How many % packet loss?

Reference

- awk(1)
- sed(1)
- http://www.grymoire.com/Unix/Awk.html
- http://www.grymoire.com/Unix/Sed.html
- https://en.wikipedia.org/wiki/Regular_expression