Computer Systems & Network Administration

Lecture 10. Shell Programming

Reference: NYCU CSCC NASA

Outline

- Variable pre-operations
- args, argc in Shell Scripts
- Arithmetic and Logics
- Control Structures: if-else, switch-case, for/while loops
- Input/output: Read from screen
- Defining Functions & Parsing Arguments
- Error Handling and Debug tool (sh -x)
- Regular Expressions

Bourne Shell

- We use Bourne Shell in this slide
- You can check your shell now by
  - `% echo $SHELL`
- Then change to Bourne Shell
  - `% 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(./)
```

Shebang

- `#!/bin/sh` at the top of the script
- Shebang (#!), or called Shabang, 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

Shebang examples

- `#!/bin/sh`
- `#!/bin/sh -x`
- `#!/bin/bash`
- `#!/usr/local/bin/bash`
- `#!/usr/bin/env bash`
- `#!/usr/bin/env python`

How to execute

- `$ sh test.sh`
    - Can execute without shebang

- `$ chmod a+x test.sh`

  `$ test.sh`

Variables

Variables

- Assignment

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

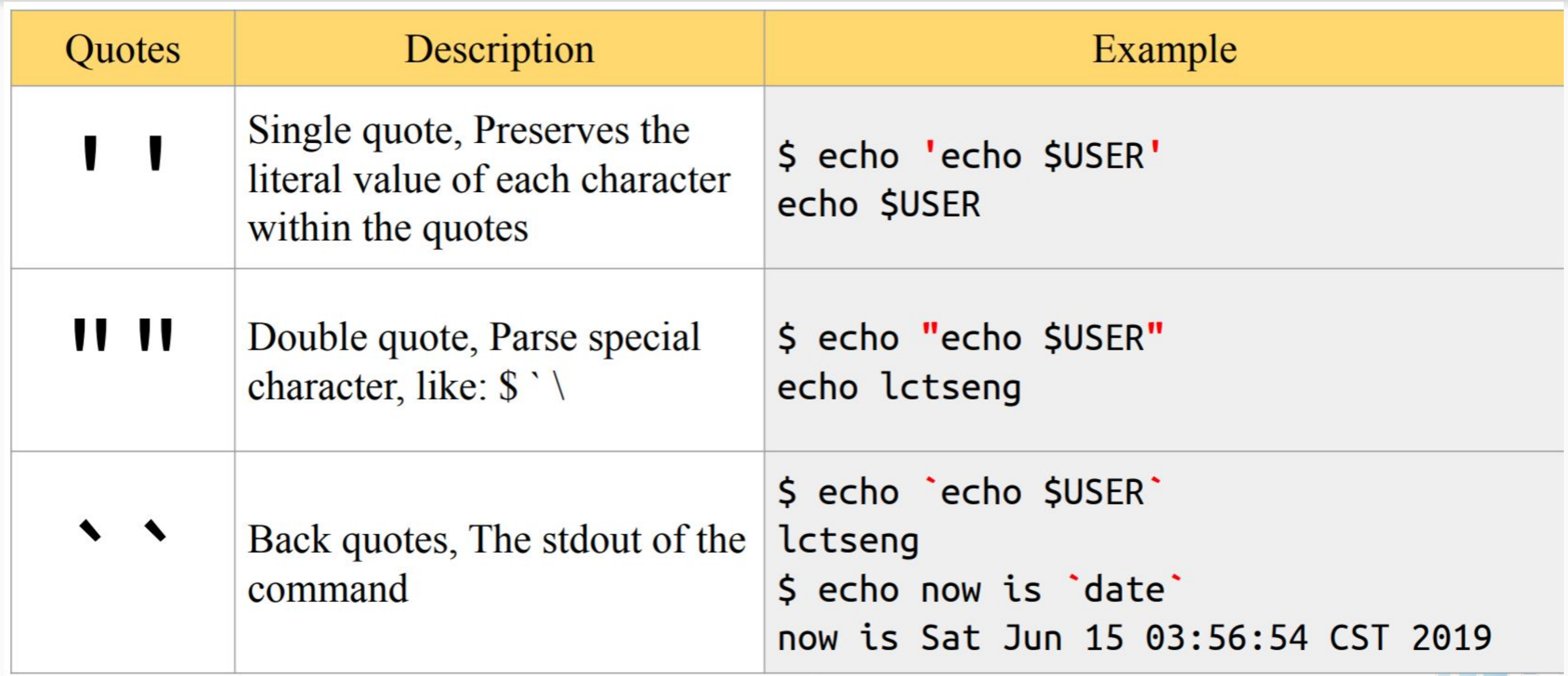
- Example

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

Variables

- There are two ways to call variable
- `$ echo "$PAGER"`
- `$ echo "${PAGER}"` <= Why?
  - Use {} to avoid ambiguity
- For example
- `$ temp_name="haha" && temp="hehe" # No Space Beside "="`
  - We want output is hehe_name
  - `$ echo $temp_name # => haha`
  - `$ echo ${temp}_name # => hehe_name`

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"`<br>`echo lctseng`

` `	Back quotes, The stdout of the command	``$ echo `echo $USER` ``
		`lctseng`
		``$ echo now is `date` ``
		`now is Sat Jun 15 03:56:54 CST 2019`

Shell variable operator

- sh(1): Parameter Expansion

Operator	Description
`${var:=value}`	If `var` is `null`, assign `value` to `var`
`${var:+value}`	If `var` is not `null`, return `value` (not assign to `var`)
`${var:-value}`	If `var` is not `null`, return `var`, otherwise return `value` (not assign to `var`)
`${var:?value}`	If `var` is `null`, print given value (stderr) and shell exits (The command stops immediately)

```
#!/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}
```

```
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
16
```

Not print 15 here

Predefined shell variables

- Environment Variables
- Other useful variables
  - Similar to C program's "int main(argc, argv)" – arguments of program

sh	Description
`$#`	Number of positional arguments (start from 0)
`$0`	Command name (Ex: What command user exec your script)
`$1, $2, ..`	Positional arguments
`$* / $@`	Explain in next slide
`$?`	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
```

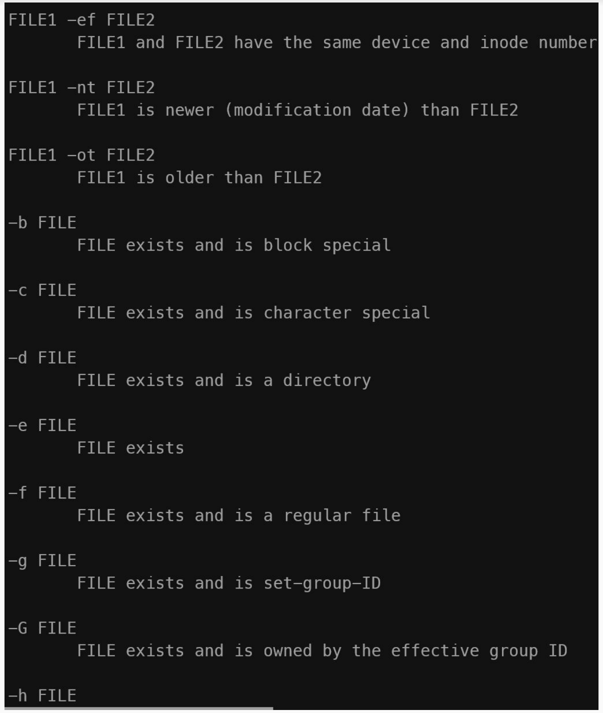
“Test” command

The "test" command

- Checking file status, string, numbers, etc
- Test and !!! return 0 (true) or 1 (false) !!!
  - % 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
- Can be used with `[ expression ]`
  - `ex. [ 10 -eq 11 ]`

The "test" command – File test

- -e file
  - True if file exists (regardless of type)
- -d file
  - True if file exists and is a directory
- -f file
  - True if file exists and is a regular file
- `$ man test` to see more



```
FILE1 -ef FILE2
       FILE1 and FILE2 have the same device and inode number

FILE1 -nt FILE2
       FILE1 is newer (modification date) than FILE2

FILE1 -ot FILE2
       FILE1 is older than FILE2

-b FILE
       FILE exists and is block special

-c FILE
       FILE exists and is character special

-d FILE
       FILE exists and is a directory

-e FILE
       FILE exists

-f FILE
       FILE exists and is a regular file

-g FILE
       FILE exists and is set-group-ID

-G FILE
       FILE exists and is owned by the effective group ID

-h FILE
```

The "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 ==)
- s1 != s2
- s1 < s2
    - True if string s1 comes before s2 based on the binary value of their characters

The "test" command – Number test

- n1 -eq n2
  - ==
- n1 -ne n2
  - !=
- n1 -gt n2
  - >
- n1 -ge n2
  - >=
- n1 -lt n2
  - <
- n1 -le n2
  - <=

==, !=, >, <, >=, <= fashion does not apply here

The "test" command – Combination

- `expression1 -a expression2`
  - `exp1 && exp2 $ [ A == B -a C == D ]`
- `expression1 -o expression2`
  - `exp1 || exp2 $ [ A == B -o C == D ]`
- -a operator has higher precedence than the -o operator
- `! expression`
  - True if expression is false.
  - `$ [ ! A == B ]` => Test expression, invert the internal result
  - `$ ! [ A == B ]` => Invert the whole test command result
  - `! [ "A" = "A" -o 1 -eq 1 ] => false`
  - `[ ! "A" = "A" -o 1 -eq 1 ] => true`

The "test" command – In Script

- Add space beside = <= != [ ]...
  - `$ [A=B] # error`
  - `$ [ A=B ] # error`
  - `$ [A = B] # error`
  - `$ [ A = B ] # ok`
- 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
```

expr command

- 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 $?
1
```

expr command

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

Arithmetic Expansion

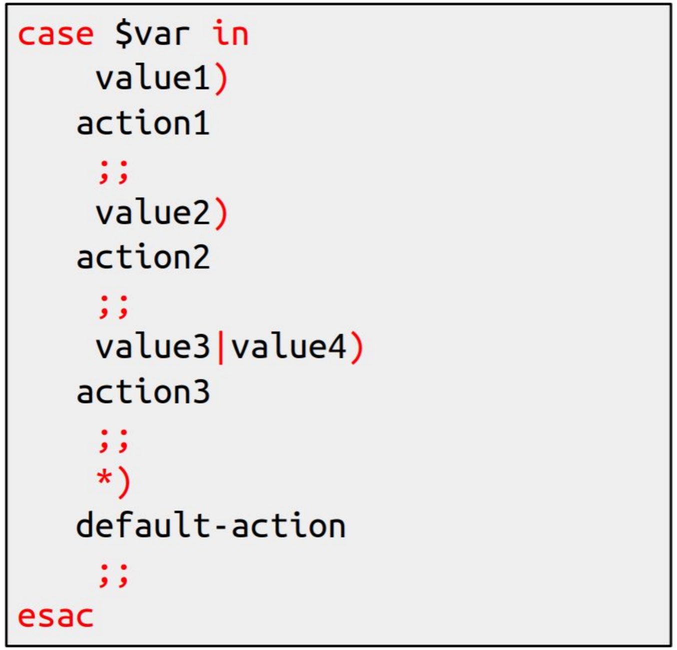
```
echo $(( 1 + 2 ))
a=8
a=$(( $a + 9 ))
a=$(( $a + 17 ))
a=$(( $a + 9453 ))
echo $a
```

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

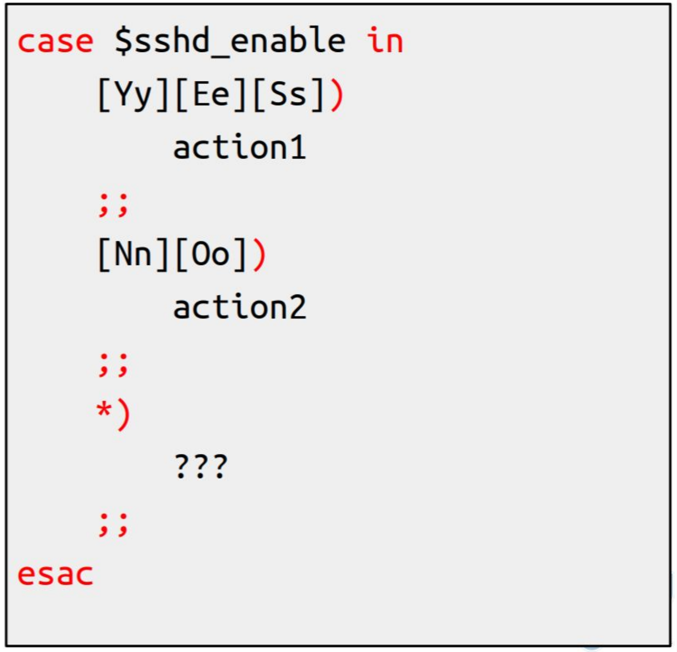
if-then-else structure

```
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



```
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

- Sometimes we want use tmp file
  - You can generate random filename in /tmp
  - A linux tool can help you achive this
- mktemp(file), mktemp -d(dir)
  - Will print filename/dirname
  - `mktemp -h`
  - ``TMPDIR=`mktemp -d` ``
  - ``TMPFILE=`mktemp -p ${TMPDIR}` ``

Functions

Functions

- Define function

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

- Remove function definition

```
unset function_name
```

- Function execution

```
function_name
```

- Function definition is local to the current shell

Functions - scope

```
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 - arguments check

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

```
Wrong
Wrong
hello world
```

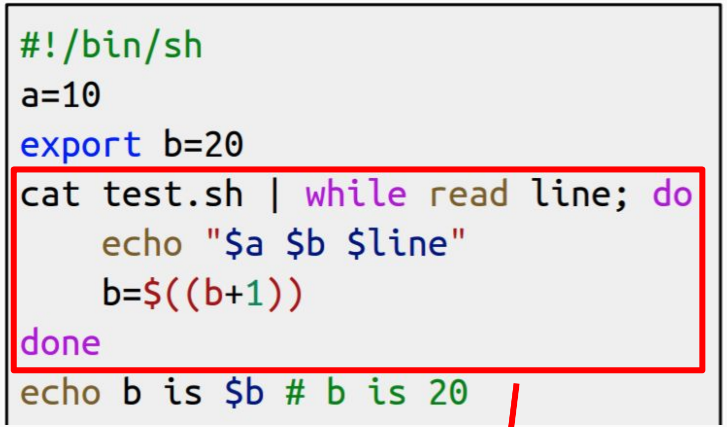
Functions - return value

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

```
2
0
```

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 # b is 20
```

this example use pipe(subprocess), so modifies to var are not effective to current process

Errors

Handling Error Conditions

- Internal error
    - Program crash
    - Caused by some command's failing to perform
    - User-error
        - Invalid input
        - Unmatched shell-script usage
- External error
    - Signal from OS
    - Ctrl + C (SIGINT)

Handling Error Conditions – External Error

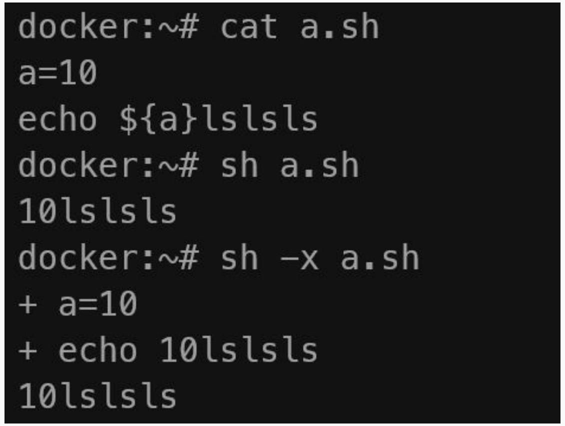
- Using trap in Bourne shell
  - `trap [command-list] [signal-list]`
  - `ex. trap "somethingtodo; exit 0" 1 2 3 14 15`
    - when catch signal 1 2 3 14 15, exit 0
  - `ex. trap "" 18`
    - do nothing => ignore signal 18

SIGNAL

- ubuntu 20.04 SIGNAL list
- Some SIGNAL can’t be catched
    - ex. SIGKILL(9), SIGSTOP(19)
    - wiki

Debugging Shell Script — Debug tools in sh

- /bin/sh -x
    - Print out the substitution results



```
docker:~# cat a.sh
a=10
echo ${a}lslsls
docker:~# sh a.sh
10lslsls
docker:~# sh -x a.sh
+ a=10
+ echo 10lslsls
10lslsls
```

Regular Expressions

Regular Expressions

- We assume you familiar with regular expressions(regex)
  - if not, here is tutorial
  - convenient tool regex101 to test regex

Regular Expressions

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

Sed - stream editor

- `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`
- `sed -n` (no print to screen)
- Command format
    - `[address1[,address2]]function[argument]`

Sed - stream editor: substitution

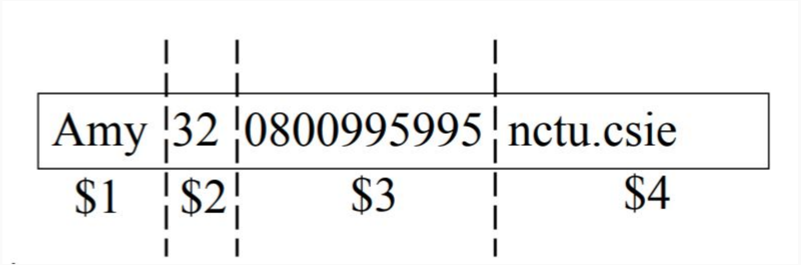
- s/pattern/replace/flags
- Flags
  - N: Make the substitution only for the N'th occurrence
  - g: replace all matches
  - p: print the matched and replaced line
  - w: write the matched and replaced line to a file
- Example
  - `sed -e 's/lctseng/LCTSENG/2' file.txt`
  - `sed -e 's/lctseng/LCTSENG/g' file.txt`
  - `sed -e 's/lctseng/LCTSENG/p' file.txt`
  - `sed -e 's/lctseng/LCTSENG/w wfile' file.txt`

Sed - stream editor: delete

- [address]d
- Example
  - `sed -e 10d`
    - Delete line 10
  - `sed -e /man/d`
  - `sed -e 10,100d`
  - `sed -e 10,/man/d`
    - Delete line from line 10 to the line contain "man"

awk

- `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
    - fs means Field separator
    - ex.
        - `awk '{print "Hello World"}' file`
        - `awk '{print $1}' file`



- structure
    - pattern { action }
    - missing pattern means always matches
    - missing { action } means print the line

awk - 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 - Pattern formats

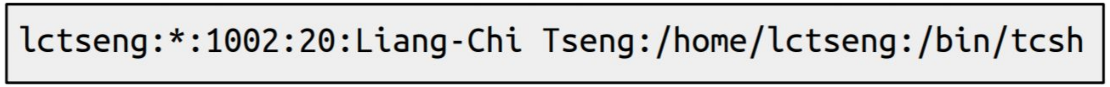
- actions
  - `if( expression ) statement [; else statement2]`
    - `awk ' { if(something) print $1}' file`
  - `awk 'BEGIN {count=0} /lctseng/ {while (count < 3) {print count;count++}}' file`
  - `awk '{for (i=0;i<3;i++) print i}' file`

awk - built-in variables

- $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

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



```
lctseng:*:1002:20:Liang-Chi Tseng:/home/lctseng:/bin/tcsh
```

Questions?