Shell Programming

Put distinctive simple tools together to accomplish your goal…

ssuyi

Outline

- ❑ Variables and expansion
- ❑ args, argc in Shell Scripts
- ❑ Arithmetic and Logics
  - Test commands
- ❑ Control Structures: if-else, switch-case, for/while loops
- ❑ Input/output
- ❑ Functions & Parsing Arguments
- ❑ Error Handling and Debug tool (sh -x)
- ❑ A Shell Script Eample: Failure Detection on Servers

- ❑ Appendix: Regular Expression
- ❑ Appendix B: sed and awk

Bourne Shell

- ❑ We use Bourne Shell in this slide.

```
% echo $SHELL
/usr/local/bin/bash

% sh
$
```

Executable script

❑ Shebang

- `#!/bin/sh`

❑ Execution

- `chmod +x test.sh`
- `./test.sh`

Shell variables (1)

❑ Assignment

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

- Example:

```
sh  ➢ $ export PAGER=/usr/bin/less
CSH ➢ % setenv PAGER /usr/bin/less

sh  ➢ $ current_month=`date +%m`
CSH ➢ % set current_month =`date +%m`
```

Shell variables (2)

There are two ways to call variable…

❑ Usage

```
% echo "$PAGER"
% echo "${PAGER}"
```

- {} to avoid ambiguity

```
% temp_name="haha"
% temp="hehe"
% echo $temp
  – hehe
% echo $temp_name
  – haha
% echo ${temp}_name
  – hehe_name
% echo ${temp_name}
  – haha
```

More clear…

Shell variable operator (1)

value assignment

※ BadCond == !GoodCond

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

operator	description
`${var:=value}`	If !GoodCond, use the value and assign to var
`${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 exits

Print → stderr

The command stops immediately

"Parameter Expansion" 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
16
```

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*}
```

These operators do not change var. value…

Results:
32
happened end closing end
end
Nothing happened end closing
Nothing happened

Predefined shell variables

Similar to C program’ s "Int main(argc, args)" – arguments of program

- ❑ Environment Variables: env
- ❑ Other useful variables:

variable	description
$#	Number of positional arguments
$0	Command name
$1, $2, ..	Positional arguments
$*	List of positional arguments (useful in for loop)
$?	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 $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
```

test command

Checking things for us… e.g. file status, statements

- ❑ test(1)
  - test, [
  - `test expression`
  - `[ expression ]`
  - Test for: file, string, number
- ❑ Test and return 0 (true) or 1 (false) in $?
  - `% test –e News ; echo $?` → $? To obtain the return code
    - If there exist the file named "News"
  - `% test "haha" = "hehe" ; echo $?`
    - Whether "haha" **equal** "hehe"
  - `% test 10 –eq 11 ; echo $?`
    - Whether 10 **equal** 11

Details on the capability of test command – File test

- ❑ -e file
  - True if file exists (regardless of type)
- ❑ -s file
  - True if file exists and has a size greater than zero
- ❑ -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
- ❑ -f file
  - True if file exists and is a regular file
- ❑ -p file
  - True if file is a named pipe (FIFO)
- ❑ -L file
  - True if file exists and is a symbolic link
- ❑ -S file
  - True if file exists and is a socket
- ❑ -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
- ❑ -u file
  - True if file exists and its set user ID flag is set
- ❑ -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
- ❑ -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
- ❑ 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

Details on the capability of 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
  - 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

Details on the capability of 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.
- ❑ 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

test command – in script

- ❑ test command short format using []
  - `% test "haha" = "hehe" ; echo $?`

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

test command – in script

```
# AND – OR – NOT
$ [ 1 -eq 2 ] || [ 1 -eq 1 ] ; echo $?  # if not
0

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

$ [ 1 -eq 1 ] && [ 1 -eq 2 ] ; echo $?  # if
1

$ [ 1 -eq 2 ] && ~~[ 1 -eq 1 ]~~ ; echo $?
1

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

$ [ 1 -eq 2 ] ; echo $?
1
```

Arithmetic Expansion

```
echo $(( 1 + 2 ))

a=5566
echo $(( $a + 2 ))
echo $(( $a - 2 ))
echo $(( $a * 2 ))
echo $(( $a / 2 ))
echo $(( $a % 2 ))
```

```
3


5568
5564
11132
2783
0
```

if-then-else structure

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

```
#!/bin/sh

a=5566
b=5538

if [ $a -ne $b ] ; then
        echo "5538 not equal 5566";
fi
```

switch-case structure (1)

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

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

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
```

while loop

```
while [ … ] ; do
        action
done

break
continue
```

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

Read from stdin

```
#!/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
```

Read from file

- ❑ Set file to a file descriptor

```
#!/bin/sh

exec 3< "file"

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

- ❑ Set file in the end of while loop

```
#!/bin/sh

while read line ; do
        echo "$line"
done < "file"
```

Create tmp file/dir

```
TMPDIR=`mktemp -d tmp.XXXXXX`
TMPFILE=`mktemp ${TMPDIR}/tmp.XXXXXX`

echo "program output" >> ${TMPFILE}
```

functions (1)

- ❑ Define function

```
func ( ) {
   command_list
}
```

- ❑ Removing function definition

```
unset func
```

- ❑ Function execution

```
func arg1 arg2
```

- ❑ Function definition is <u>local to the current shell</u>

※ Define the function before first use…

functions (2) - scoping

```
func () {
        # global variable
        echo $a
        a="hello"
}
a="5566"

func
echo $a
```

Result:

```
5566
hello
```

```
func () {
        # local variable
        local a="world"
        echo $a
}
a="5566"

func
echo $a
```

Result:

```
world
hello
```

functions (3) - arguments check

```
#!/bin/sh
func () {
    if [ $# -eq 2 ] ; then
        local group=$1
        local desc=$2
        echo "$group is $desc"
    else
        echo "wrong args"
    fi
}
func 5566 "gg"
func 5566 "gg" 123
func 5566
func
```

Result:

```
5566 is gg
wrong args
wrong args
wrong args
```

functions (4) – return value

```
#!/bin/sh
func () {
    if [ $1 -eq 1 ] ; then
        return 1
    else
        return 2
    fi
}
func 1
echo $?                     # 1
func 2
echo $?                     # 2
```

Parsing arguments

- ❑ Use getopts (recommended)

```
#!/bin/sh

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 gg
2-th arg
one of ABC
3-th arg
one of ABC
4-th arg
one of ABC
6-th arg
gg
```

- ":" means additional arg.
- $OPTARG: content of arguments
- $OPTIND: the index of the arguments

Handling Error Conditions

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

Handling Error Conditions – Internal Error(1)

❑ Ex:

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
```

Handling Error Conditions – Internal Error(2)

- ❑ EX:

```
#!/bin/sh

help () {
    echo "Usage: $0 -c [ -f flag ]"
    exit 1
}

has_c=""
flag=""
invalid=""

while getopts cf: op ; do
    case $op in
        c) has_c="1" ;;
        f) flag=$OPTARG ;;
        *) invalid="1" ;;
    esac
done

if [ -z $has_c ] ; then
    echo "No c!"
    help
fi

if [ ! -z $flag ] && [ $flag != "correct" ] ; then
    echo "Error flag!"
    help
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

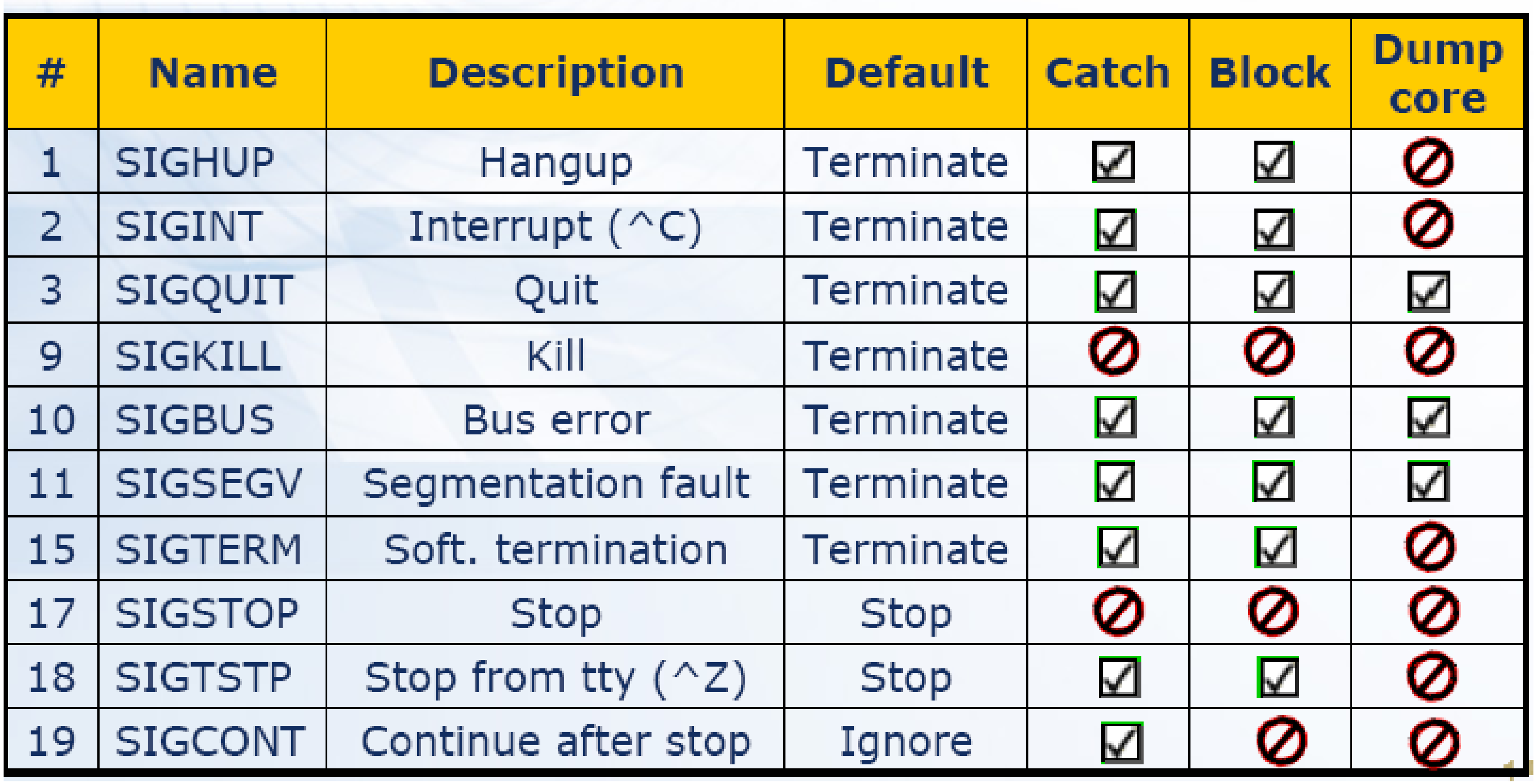
Usag: trap "[commands]" list of signals looking for…

```
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	☑	⊘	⊘

Debugging Shell Script

Debug tools in sh…

❑Ex:

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}
```

❑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
```

Debug msgs.
print out the
substitution results…

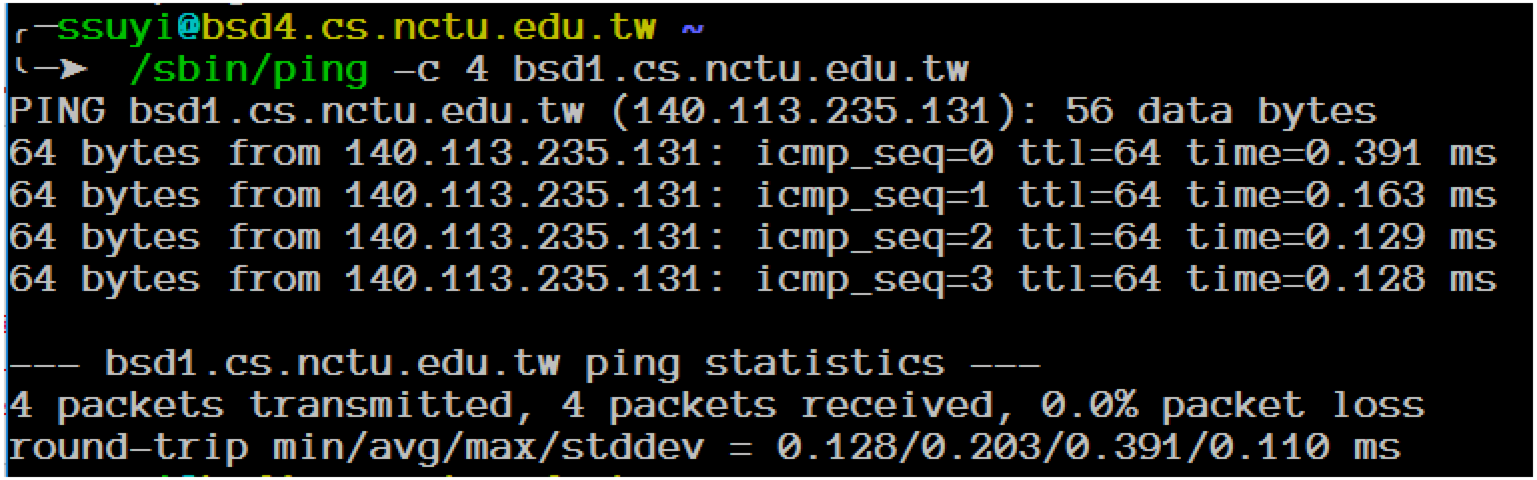
Useful tools

- ❑ ps (1)
- ❑ xargs (1)
- ❑ tail (1)
- ❑ head (1)
- ❑ cut (1)
- ❑ sort (1)
- ❑ tr (1)

Shell Script Examples

check alive (1)

- ping



```
┌─ssuyi@bsd4.cs.nctu.edu.tw ~
└─➤  /sbin/ping -c 4 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=0.391 ms
64 bytes from 140.113.235.131: icmp_seq=1 ttl=64 time=0.163 ms
64 bytes from 140.113.235.131: icmp_seq=2 ttl=64 time=0.129 ms
64 bytes from 140.113.235.131: icmp_seq=3 ttl=64 time=0.128 ms

--- bsd1.cs.nctu.edu.tw ping statistics ---
4 packets transmitted, 4 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.128/0.203/0.391/0.110 ms
```

check alive (2)

```
#!/bin/sh
# [Usage] isAlive.sh host

Usage="[Usage] $0 host"
temp="$1.ping"
Admin="liuyh"
count="20"

if [ $# != 1 ] ; then
  echo $Usage
else
  /sbin/ping -c ${count} $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
```

Appendix A: Regular Expression

pattern matching

Regular Expression (1) - Intro.

- ❑ 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) - Intro.

- Union: A | B
- Concatenation: AB
- Kleene closure: A*

Regular Expression (3) - operators

- ❑ Pattern-matching
  - Special operators

operator	Description
.	Any single character (usually except newline)
[]	Any character in []
[^]	Any character not in []
^	Start of a line
$	End of a line
*	Match zero or more
?	Match zero or one
+	Match one or more
{m,n}	At least m times and at most n times
{m,}	At least m times.
{m}	Exact m times.
\	Escape character

Regular Expression (4) - operators

- ❑ Character classes

class	perl	ASCII
[:alnum:]	\w([A-Za-z0-9_])	[A-Za-z0-9]
[:alpha:]	\a	[A-Za-z]
[:blank:]		[\t]
[:digit:]	\d	[0-9]
[:lower:]	\l	[a-z]
[:upper:]	\u	[A-Z]
[:space:]	\s	[\t\r\n\v\f]
[:xdigit:]		[A-Fa-f0-9]
[:punct:]		[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]

Regular Expression (5) - grouping

- () and \n
- Example:
  - ([A|B])\1
    - match AA, BB
  - ([0-9])([0-9])\2\1
    - match 5665, 1221

Regular Expression (7)

- ❑ 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

Regular Expression (8) -Practice

- ❑ https://regexcrossword.com/
- ❑ Finish at least intermediate level

Regular Expression (9) -Example

- 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 (10) -Example

- ^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
- [Ee][Nn][Dd]$
  - 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
- [+-][1-9] [0-9]*
  - Integer with a preceding + or -1

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 (11) -Example

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

- "Shell Patterns" in sh(1)
- "REGULAR EXPRESSIONS" in grep(1)
- …

Appendix B: sed and awk

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]
- Address format
  - n or $ ➔ 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 - substitution (1)

- substitution
  - Syntax

    [2addr] 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
    - I:Match the regular expression in a case-insensitive way

sed – Stream EDitor
- substitution (2)

❑ Ex:

- sed –e ‘s/liuyh/LIUYH/2’ file
- sed –e ‘s/liuyh/LIUYH/g’ file
- sed –e ‘s/liuyh/LIUYH/p’ file
- sed –n –e ‘s/liuyh/LIUYH/p’ file
- sed –e ‘s/liuyh/LIUYH/w wfile’ file

file

```
I am jon
I am john
I am liuyh
I am liuyh
I am nothing
```

sed – Stream EDitor
- delete

❑ delete

- Syntax:

  [2addr]d

❑ Ex:

- sed –e 10d
- sed –e /man/d
- sed –e 10,100d
- sed –e 10,/man/d

sed – Stream EDitor

- append, insert, change

- ❑ append, insert, change
  - Syntax:

```
[1addr]a\
text
```

```
[1addr]i\
text
```

```
[2addr]c\
text
```

- insert → insert before the line
- change → replace whole line

- ❑ Ex:
  - sed –f sed.src file

sed.src

```
/liuyh/i \
Meet liuyh, Hello
```

file

```
I am jon
I am john
I am liuyh
I am liuyh
I am nothing
```

Results:

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

sed – Stream EDitor - transform

- ❑ transform One-by-one transformation
  - Syntax:

    [add1,addr2] y/xyz…/abc…/
- ❑ Ex:
  - sed –e ‘y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/’ file
    - ➢ Lowercase to uppercase

sed – Stream EDitor - print

- ❑ print
  - Syntax:

    [addr1, addr2]p
- ❑ Ex:
  - sed -n -e '/^liuyh/p' Print out the lines that begins with liuyh

-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.

awk

- awk(1)
  - `awk [-F fs] [ 'prog' | -f prog_file ][ file ... ]`
    - awk will read the file line by line and evaluate the pattern, then do the action if the test is true
- Program structure
  - `pattern { action }`
  - A missing `{ action }` means print the line
  - A missing pattern always matches

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 line
    - `awk 'BEGIN { print "Nice to meet you" }'`
- END
  - after the last line is read
    - `awk 'END { print "Bye Bye" }'`

awk – action format

❑ Actions

```
if( expression ) statement [ else statement ]
while( expression ) statement
for( expression ; expression ; expression ) statement
for( var in array ) statement
do statement while( expression )
break
continue
{ [ statement ... ] }
expression              # commonly var = expression
print [ expression-list ] [ > expression ]
printf format [ , expression-list ] [ > expression ]
return [ expression ]
next                    # skip remaining patterns on this input line
nextfile                # skip rest of this file, open next, start at top delete
array[ expression ]     # delete an array element
delete array            # delete all elements of array
exit [ expression ]     # exit immediately; status is expression
```

awk

- Example

```
BEGIN {
         name = "Doraemon"                     # typeless variables
         height = 129.3
         weight = 129.3

         print "Hello, I'm " name
         print "H: " height
         print "W: " weight
         print "BMI: " weight / (height*0.01)**2        # floating point

         for (i = 1; i < 100; i++)
                   if ( i ~ 3 )                          # the 'match' operator
                            print i " Threeeee!!!!!"

         while ( i --> 0 ) {                            # secret goes-to operator!
                   if ( i !~ 2 && i !~ 3 )
                            c[i] = i * 10
         }

         for (r in c)                                   # associate array
                   print r " " c[r]
}
```

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

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
- http://www.vectorsite.net/tsawk.html
- https://www.gnu.org/software/sed/manual/sed.html
- https://www.gnu.org/software/gawk/manual/gawk.html