Shell Scripting

ILG

Insight GNU/Linux & BSD Group

www.gnugroup.org
info@gnugroup.org

Outline

- What is shell?
- Basic
- Syntax
  - Lists
  - Functions
  - Command Execution
  - Here Documents
  - Debug
- Regular Expression
- Find

Why Shell?

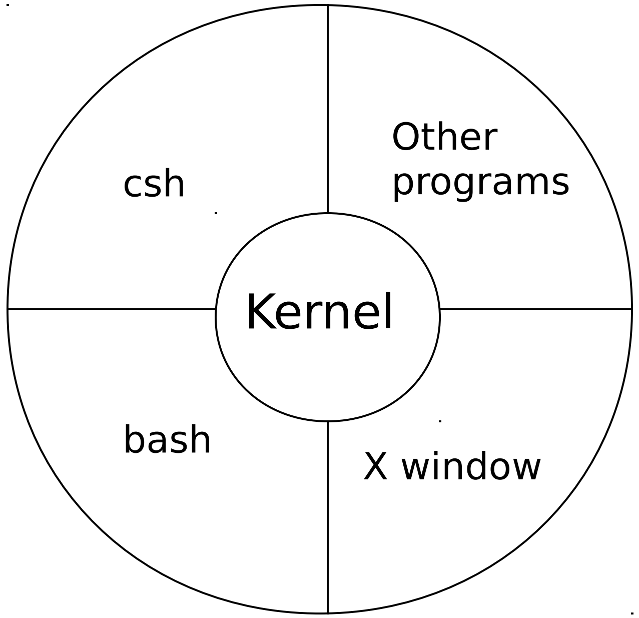
- The commercial UNIX used Korn Shell
- For Linux, the Bash is the default
- Why Shell?
  - For routing jobs, such as system administration, without w riting programs
  - However, the shell script is not efficient, therefore, can be used for prototyping the ideas
- For example,

```
% ls –al | more (better format of listing directory)
% man bash | col –b | lpr (print man page of man)
```

What is Shell?

- Shell is the interface between end user an d the Linux system, similar to the comman ds in Windows
- Bash is installed as in /bin/sh
- Check the version

```
% /bin/bash --version
```



Pipe and Redirection

- Redirection (< or >)
  - % **ls –l > lsoutput.txt** (save output to lsoutput.txt)
  - % **ps >> lsoutput.txt** (append to lsoutput.txt)
  - % **more < killout.txt** (use killout.txt as parameter to more)
  - % **kill -l 1234 > killouterr.txt 2 >&1** (redirect to the same file)
  - % **kill -l 1234 >/dev/null 2 >&1** (ignore std output)
- Pipe (|)
  - Process are executed *concurrently*
  - % **ps | sort | more**
  - % **ps –xo comm | sort | uniq | grep –v sh | more**
  - % **cat mydata.txt | sort | uniq | > mydata.txt** (generates an empty file !)

Shell as a Language

- We can write a script containing many shell commands
- Interactive Program:
  - grep files with POSIX string and print it

```
% for file in *
> do
> if grep –l POSIX $file
> then
> more $file
➢ fi
➢ done
Posix
There is a file with POSIX in it
```

  - ‘*’ is wildcard

```
% more `grep –l POSIX *`
% more $(grep –l POSIX *)
% more –l POSIX * | more
```

Writing a Script

- Use text editor to generate the “first” file

```
#!/bin/bash
# first
# this file looks for the files containing POSIX
# and print it
for file in *
do
   if grep –q POSIX $file
   then
      echo $file
   fi
done
exit 0
```

exit code, 0 means successful

```
% /bin/bash first
% chmod +x first
%./first (make sure . is include in PATH parameter)
```

Syntax

- Variables
- Conditions
- Control
- Lists
- Functions
- Shell Commands
- Result
- Document

Variables

- Variables needed to be declared, note it is case-sensitive (e. g. foo, FOO, Foo)
- Add ‘$’ for storing values

```
% salutation=Hello
% echo $salutation
Hello
% salutation=7+5
% echo $salutation
7+5
% salutation=“yes dear”
% echo $salutation
yes dear
% read salutation
Hola!
% echo $salutation
Hola!
```

Quoting

- Edit a “vartest.sh” file

```
#!/bin/bash

myvar=“Hi there”

echo $myvar
echo “$myvar”
echo `$myvar`
echo \$myvar

echo Enter some text
read myvar

echo ‘$myvar’ now equals $myvar
exit 0
```

Output

```
Hi there
Hi there
$myvar
$myvar
Enter some text
Hello world
$myvar now equals Hello world
```

Environment Variables

$HOME	home directory
$PATH	path
$PS1	(normally %)
$PS2	(normally >)
$$	process id of the script
$#	number of input parameters
$0	name of the script file
$IFS	separation character (white space)

- Use ‘env’ to check the value

Parameter

```
% IFS = ` `
% set foo bar bam
% echo “$@”
foo bar bam
% echo “$*”
foo bar bam
% unset IFS
% echo “$*”
foo bar bam
```

doesn’t matter IFS

Parameter

Edit file ‘try_var’

```
#!/bin/bash
salutation=“Hello”
echo $salutation
echo “The program $0 is now running”
echo “The parameter list was $*”
echo “The second parameter was $2”
echo “The first parameter was $1”
echo “The user’s home directory is $HOME”
echo “Please enter a new greeting”
read salutation
echo $salutation
echo “The script is now complete”
exit 0
```

```
%./try_var foo bar baz
Hello
The program ./try_var is now running
The second parameter was bar
The first parameter was foo
The parameter list was foo bar baz
The user’s home directory is /home/jai
Please enter a new greeting
Hola
Hola
The script is now complete
```

Condition

need space !

- test or ' ['

```
if test –f fred.c
then
...
fi
```

```
If [ -f fred.c
 ]
then
...
fi
```

```
if [ -f fred.c ];then
...
fi
```

```
expression1 –eq expression2
expression1 –ne expression2
expression1 –gt expression2
expression1 –ge expression2
expression1 -lt expression2
expression1 –le expression2
!expression
```

```
String1 = string2
String1 != string 2
-n string  (if not empty string)
-z string  (if empty string)
```

```
-d file   if directory
-e file   if exist
-f file   if file
-g file   if set-group-id
-r file   if readable
-s file   if size >0
-u file   if set-user-id
-w file   if writable
-x file   if executable
```

Control Structure

Syntax

```
if condition
then
  statement
else
  statement
fi
```

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
if [ $timeofday = “yes” ]; then
  echo “Good morning”
else
  echo “Good afternoon”
fi
exit 0
```

```
Is it morning? Please answer yes or no
yes
Good morning
```

Condition Structure

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
if [ $timeofday = “yes” ]; then
  echo “Good morning”
elif [ $timeofday = “no” ]; then
  echo “Good afternoon”
else
  echo “Sorry, $timeofday not recongnized. Enter yes or no”
   exit 1
fi
exit 0
```

Condition Structure

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
if [ “$timeofday” = “yes” ]; then
  echo “Good morning”
elif [ $timeofday = “no” ]; then
  echo “Good afternoon”
else
  echo “Sorry, $timeofday not recongnized. Enter yes or no”
   exit 1
fi
exit 0
```

If input “enter” still returns Good morning

Loop Structure

Syntax

```
    for variable
    do
      statement
    done
```

```
#!/bin/bash

for foo in bar fud 43
do
   echo $foo
done
exit 0
```

```
bar
fud
43
```

How to output as bar fud 43?

Try change for foo in “bar fud 43”

This is to have space in variable

Loop Structure

- Use wildcard ‘*’

```
#!/bin/bash

for file in $(ls f*.sh); do
  lpr $file
done
exit 0
```

Print all f*.sh files

Loop Structure

Syntax

```
while condition
do
  statement
done
```

Syntax

```
until condition
do
  statement
done
```

Note: condition is Reverse to while How to re-write previous sample?

```
#!/bin/bash
for foo in 1 2 3 4 5 6 7 8 9 10
do
  echo “here we go again”
done
exit 0
```

```
#!/bin/bash
foo = 1
while [ “$foo” –le 10 ]
do
  echo “here we go again”
  foo = $foo(($foo+1))
done
exit 0
```

Case Statement

Syntax

```
case variable in\
 pattern [ | pattern ] …) statement;;
 pattern [ | pattern ] …) statement;;
 …
esac
```

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
case “$timeofday” in
  yes) echo “Good Morning”;;
  y)   echo “Good Morning”;;
  no)  echo “Good Afternoon”;;
  n)   echo “Good Afternoon”;;
  * )  echo “Sorry, answer not recongnized”;;
esac
exit 0
```

Case Statement

- A much “cleaner” version

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
case “$timeofday” in
  yes | y | Yes | YES ) echo “Good Morning”;;
  n* | N* )     echo “Good Afternoon”;;
  * )           echo “Sorry, answer not recongnized”;;
esac
exit 0
```

But this has a problem, if we enter ‘never’ which obeys n* case and prints “Good Afternoon”

Case Statement

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
case “$timeofday” in
  yes | y | Yes | YES )
               echo “Good Morning”
               echo “Up bright and early this morning”
               ;;
  [nN]*)
               echo “Good Afternoon”;;
  *)
               echo “Sorry, answer not recongnized”
               echo “Please answer yes of no”
               exit 1
               ;;
esac
exit 0
```

List

- AND (&&)

  statement1 **&&** statement2 **&&** statement3 …

```
#!/bin/sh
touch file_one
rm –f file_two

if [ -f file_one ] && echo “Hello” && [-f file_two] && echo “ there”
then
  echo “in if”
else
  echo “in else”
fi
exit 0
```

Check if file exist if not then create one

Remove a file

Output

```
Hello
in else
```

List

- OR (||)

  statement1 || statement2 || statement3 …

```
#!/bin/bash

rm –f file_one
if [ -f file_one ] || echo “Hello” || echo “ there”
then
  echo “in if”
else
  echo “in else”
fi

exit 0
```

Output

```
Hello
in else
```

Statement Block

- Use multiple statements in the same place

```
get_comfirm && {
  grep –v “$cdcatnum” $stracks_file > $temp_file
  cat $temp_file > $tracks_file
  echo
  add_record_tracks
}
```

Function

- You can define functions for “structured” scripts

```
          function_name() {
            statements
          }
```

```
#!/bin/bash
foo() {
  echo “Function foo is executing”
}
echo “script starting”
foo
echo “script ended”
exit 0
```

Output

```
script starting
Function foo is executing
Script ended
```

You need to define a function before using it
The parameters $*,$@,$#,$1,$2 are replaced by local value
if function is called and return to previous after function is finished

Function

define local variable

```
#!/bin/bash
sample_text=“global variable”
foo() {
  local sample_text=“local variable”
  echo “Function foo is executing”
  echo $sample_text
}
echo “script starting”
echo $sample_text
foo

echo “script ended”
echo $sample_text

exit 0
```

Output?

Check the scope of the variables

Function

- Use return to pass a result

```
#!/bin/bash
yes_or_no() {
  echo “Is your name $* ?”
  while true
  do
    echo –n “Enter yes or no:”
    read x
    case “$x” in
      y | yes ) return 0;;
      n | no ) return 1;;
      * ) echo “Answer yes or no”
    esac
    done
}
```

```
echo “Original parameters are $*”
if yes_or_no “$1”
then
  echo “Hi $1, nice name”
else
  echo “Never mind”
fi
exit 0
```

Output

```
./my_name Jai Phull
Original parameters are Jai Phull
Is your name Jai?
Enter yes or no: yes
Hi Jai, nice name.
```

Command

- External:
  use interactively
- Internal:
- only in script
- **break**
  skip loop

```
#!/bin/bash
rm –rf fred*
echo > fred1
echo > fred2
mkdir fred3
echo > fred4

for file in fred*
do
  if [ -d “$file” ] ; then
      break;
  fi
done
echo first directory starting fred was $file
rm –rf fred*
exit 0
```

Command

- :   treats it as true

```
#!/bin/bash

rm –f fred
if [ -f fred ]; then
    :
else
    echo file fred did not exist
fi

exit 0
```

Command

- continue		continues next iteration

```
#!/bin/bash
rm –rf fred*
echo > fred1
echo > fred2
mkdir fred3
echo > fred4
for file in fred*
do
  if [ -d “$file” ]; then
      echo “skipping directory $file”
      continue
  fi
    echo file is $file
done
rm –rf fred*
exit 0
```

Command

- . ./shell_script   execute shell_script

```
classic_set
#!/bin/bash
verion=classic
PATH=/usr/local/old_bin:/usr/bin:/bin:.
PS1=“classic> ”
latest_set
#!/bin/sh
verion=latest
PATH=/usr/local/new_bin:/usr/bin:/bin:.
PS1=“latest version> ”
```

```
% . ./classic_set
classic> echo $version
classic
Classic> . latest_set
latest
latest version>
```

Command

- echo print string
- -n do not output the trailing newline
- -e enable interpretation of backslash escapes
  - \0NNN the character whose ACSII code is NNN
  - \\ backslash
  - \a alert
  - \b backspace
  - \c suppress trailing newline
  - \f form feed
  - \n newline
  - \r carriage return
  - \t horizontal tab
  - \v vertical tab

Try these

```
% echo –n “string to \n output”

% echo –e “string to \n output”
```

Command

- eval evaluate the value of a parameter
  similar to an extra ‘$’

```
% foo=10
% x=foo
% y=‘$’$x
% echo $y
```

Output is $foo

```
% foo=10
% x=foo
% eval y=‘$’$x
% echo $y
```

Output is 10

Command

- exit *n*   ending the script
- 0 means success
- 1 to 255 means specific error code
- 126 means not executable file
- 127 means no such command
- 128 or >128 signal

```
#!/bin/bash
if [ -f .profile ]; then
  exit 0
fi
exit 1

Or % [ -f .profile ] && exit 0 || exit 1
```

Command

- export gives a value to a parameter

This is ‘export2’

```
#!/bin/bash
echo “$foo”
echo “$bar”
```

This is ‘export1’

```
#!/bin/bash
foo=“The first meta-syntactic variable”
export bar=“The second meta-syntactic variable”

export2
```

Output is

```
%export1

The second-syntactic variable
%
```

Command

- expr evaluate expressions

`%x=`expr $x + 1`` (Assign result value expr $x+1 to x)

Also can be written as

`%x=$(expr $x + 1)`

Expr1 | expr2 (or)	expr1 != expr2
Expr1 & expr2 (and)	expr1 + expr2
Expr1 = expr2	expr1 – expr2
Expr1 > expr2	expr1 * expr2
Expr1 >= expr2	expr1 / expr2
Expr1 < expr2	expr1 % expr2 (module)
Expr1 <= expr2	

Command

- printf format and print data
- Escape sequence
  - \\backslash
  - \a beep sound
  - \b backspace
  - \fform feed
  - \n newline
  - \r carriage return
  - \ttab
  - \v vertical tab
- Conversion specifier
  - %d decimal
  - %c character
  - %s string
  - %% print %

```
% printf “%s\n” hello
Hello
% printf “%s %d\t%s” “Hi There” 1
5 people
Hi There 15     people
```

Command

return	return a value
set	set parameter variable

```
#!/bin/bash

echo the date is $(date)
set $(date)
echo The month is $2

exit 0
```

Command

- Shift		shift parameter once, $2 to $1, $3 to $2, and so on

```
#!/bin/bash

while [ “$1” != “” ]; do
     echo “$1”
     shift
done

exit 0
```

Command

- trap action after receiving signal

```
trap command signal
```

- signal / explain

signal	explain
HUP (1)	hung up
INT (2)	interrupt (Crtl + C)
QUIT (3)	Quit (Crtl + \)
ABRT (6)	Abort
ALRM (14)	Alarm
TERM (15)	Terminate

Command

```
#!/bin/bash
trap ‘rm –f /tmp/my_tmp_file_$$’ INT
echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$
echo “press interrupt (CTRL-C) to interrupt …”
while [ -f /tmp/my_tmp_file_$$ ]; do
    echo File exists
    sleep 1
done
echo The file no longer exists
trap INT
echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$
echo “press interrupt (CTRL-C) to interrupt …”
while [ -f /tmp/my_tmp_file_$$ ]; do
    echo File exists
    sleep 1
done
echo we never get there
exit 0
```

Command

```
creating file /tmp/my_file_141
press interrupt (CTRL-C) to interrupt …
File exists
File exists
File exists
File exists
The file no longer exists
Creating file /tmp/my_file_141
Press interrupt (CTRL-C) to interrupt …
File exists
File exists
File exists
File exists
```

Command

Unset remove parameter or function

```
#!/bin/bash

foo=“Hello World”
echo $foo

unset $foo
echo $foo
```

Pattern Matching

- find search for files in a directory hierarchy

  find [path] [options] [tests] [actions]

options

-depth	find content in the directory
-follow	follow symbolic links
-maxdepths N	fond N levels directories
-mount	do not find other directories

tests

-atime N	accessed N days ago
-mtime N	modified N days ago
-new otherfile	name of a file
-type X	file type X
-user username	belong to username

Pattern Matching

operator

!	-not	test reverse
-a	-and	test and
-o	-or	test or

action

-exec command	execute command
-ok command	confirm and exectute command
-print	print
-ls	ls –dils

Find files newer than while2 then print

```
% find . –newer while2 -print
```

Pattern Matching

Find files newer than while2 then print only files

```
% find . –newer while2 –type f –print
```

Find files either newer than while2, start with ‘_’

```
% find . \( -name “_*” –or –newer while2 \) –type f
  –print
```

Find files newer than while2 then list files

```
% find . –newer while2 –type f –exec ls –l {} \;
```

Pattern Matching

- grep print lines matching a pattern

  (General Regular Expression Parser)

```
grep [options] PATTERN [FILES]
```

option

-c	print number of output context
-E	Interpret PATTERN as an extended regular expression
-h	Supress the prefixing of filenames
-i	ignore case
-l	surpress normal output
-v	invert the sense of matching

```
% grep in words.txt
% grep –c in words.txt words2.txt
% grep –c –v in words.txt words2.txt
```

Regular Expressions

- a **regular expression** (abbreviated as **regexp** or **regex**, with plu ral forms **regexps**, **regexes**, or **regexen**) is a string that describe s or matches a set of strings, according to certain syntax rules.
- Syntax
  - ^ Matches the start of the line
  - $ Matches the end of the line
  - . Matches any single character
  - [] Matches a single character that is contained within the brack ets
  - [^] Matches a single character that is not contained within the brackets
  - () Defines a "marked subexpression”
  - {*x*,*y*}Match the last "block" at least *x* and not more than *y* time s

Regular Expressions

- Examples:
  - ".at" matches any three-character string like *ha t*, *cat* or *bat*
  - "[hc]at" matches *hat* and *cat*
  - "[^b]at" matches all the matched strings from the regex ".at" except *bat*
  - "^[hc]at" matches *hat* and *cat* but only at the beginning of a line
  - "[hc]at$" matches *hat* and *cat* but only at the e nd of a line

Regular Expressions

POSIX class	**similar to**	**meaning**
[:upper:]	[A-Z]	uppercase letters
[:lower:]	[a-z]	lowercase letters
[:alpha:]	[A-Za-z]	upper- and lowercase letters
[:alnum:]	[A-Za-z0-9]	digits, upper- and lowercase letters
[:digit:]	[0-9]	digits
[:xdigit:]	[0-9A-Fa-f]	hexadecimal digits
[:punct:]	[.,!?:...]	punctuation
[:blank:]	[\t]	space and TAB characters only
[:space:]	[\t\n\r\f\v]	blank (whitespace) characters
[:cntrl:]		control characters
[:graph:]	[^ \t\n\r\f\v]	printed characters
[:print:]	[^\t\n\r\f\v]	printed characters and space

- Example: [[:upper:]ab] should only match the uppercase letters and lowercase 'a' and 'b'.

Regular Expressions

- **POSIX modern (extended) regular expressions**
- The more modern "extended" regular expressions can often be used with modern Unix utilities by including the command line flag "-E".

+	Match one or more times
?	Match at most once
*	Match zero or more
{n}	Match n times
{n,}	Match n or more times
{n,m}	Match n to m times

Regular Expressions

- Search for lines ending with “e”

```
% grep e$ words2.txt
```

- Search for “a”

```
% grep a[[:blank:]] word2.txt
```

- Search for words starting with “Th.”

```
% grep Th.[[:blank:]] words2.txt
```

- Search for lines with 10 lower case characters

```
% grep –E [a-z]\{10\} words2.txt
```

Command

- $(command) to execute command in a script
- Old format used “`” but it can be confused with “’ ”

```
#!/bin/bash
echo The current directory is $PWD
echo the current users are $(who)
```

Arithmetic Expansion

- Use $((…)) instead of expr to evaluate arithmetic equation

```
#!/bin/bash
x=0
while [ “$x” –ne 10]; do
    echo $x
    x=$(($x+1))
done

exit 0
```

Parameter Expansion

- Parameter Assignment

```
foo=fred
echo  $foo
```

```
${param:-default}  set default if null
${#param} length of param
${param%word} remove smallest suffix pattern
${param%%word} remove largest suffix pattern
${param#word} remove smallest prefix pattern
${param##word} remove largest prefix pattern
```

```
#!/bin/bash
for i in 1 2
do
  my_secret_process $i_tmp
done
```

Gives result
“mu_secret_process:
too few arguments”

```
#!/bin/bash
for i in 1 2
do
  my_secret_process ${i}_tmp
done
```

Parameter Expansion

```
#!/bin/bash
unset foo
echo ${foo:-bar}

foo=fud
echo ${foo:-bar}

foo=/usr/bin/X11/startx
echo ${foo#*/}
echo ${foo##*/}

bar=/usr/local/etc/local/networks
echo ${bar%local*}
echo ${bar%%local*}

Exit 0
```

Output

```
bar
fud
usr/bin/X11/startx
startx
/usr/local/etc
/usr
```

Here Documents

- A here document is a special-purpose code block, starts with <<

```
#!/bin/bash
cat <<!FUNKY!
hello
this is a here
document
!FUNCKY!
exit 0
```

```
#!/bin/bash
ed a_text_file <<HERE
3
d
.,\$s/is/was/
w
q
HERE
exit 0
```

a_text_file

```
That is line 1
That is line 2
That is line 3
That is line 4
```

Output

```
That is line 1
That is line 2
That was line 4
```

Debug

- sh –n<script>   set -o noexec   check syntax
  set –n
- sh –v<script>   set -o verbose   echo command before
  set –v
- sh –x<script>   set –o trace   echo command after
  set –x
  set –o nounset   gives error if undefined
  set –x

```
set –o xtrace
set +o xtrace
trap ‘echo Exiting: critical variable =$critical_variable’
   EXIT
```

References

- Bash Beginners Guide (http://tldp.org/LDP/Bash-Beginners-Guide/)

Shell Scripting

ILG

Insight GNU/Linux & BSD Group

www.gnugroup.org
info@gnugroup.org

Outline

- What is shell?
- Basic
- Syntax
  - Lists
  - Functions
  - Command Execution
  - Here Documents
  - Debug
- Regular Expression
- Find

Why Shell?

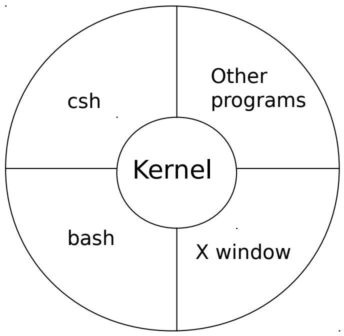
- The commercial UNIX used Korn Shell
- For Linux, the Bash is the default
- Why Shell?
  - For routing jobs, such as system administration, without w riting programs
  - However, the shell script is not efficient, therefore, can be used for prototyping the ideas
- For example,

```
% ls –al | more (better format of listing directory)
% man bash | col –b | lpr (print man page of man)
```

What is Shell?

- Shell is the interface between end user an d the Linux system, similar to the comman ds in Windows
- Bash is installed as in /bin/sh
- Check the version

```
% /bin/bash --version
```



Pipe and Redirection

- Redirection (< or >)

```
% ls –l > lsoutput.txt (save output to lsoutput.txt)
% ps >> lsoutput.txt (append to lsoutput.txt)
% more < killout.txt (use killout.txt as parameter to m
  ore)
% kill -l 1234 > killouterr.txt 2 >&1 (redirect to the
  same file)
% kill -l 1234 >/dev/null 2 >&1 (ignore std output)
```

- Pipe (|)
  - Process are executed *concurrently*

```
% ps | sort | more
% ps –xo comm | sort | uniq | grep –v sh | more
% cat mydata.txt | sort | uniq | > mydata.txt (generate
  s an empty file !)
```

Shell as a Language

- We can write a script containing many shell commands
- Interactive Program:
  - grep files with POSIX string and print it

```
% for file in *
> do
> if grep –l POSIX $file
> then
> more $file
➢ fi
➢ done
Posix
There is a file with POSIX in it
```

  - '*' is wildcard

```
% more `grep –l POSIX *`
% more $(grep –l POSIX *)
% more –l POSIX * | more
```

Writing a Script

- Use text editor to generate the “first” file

```
#!/bin/bash
# first
# this file looks for the files containing POSIX
# and print it
for file in *
do
   if grep –q POSIX $file
   then
      echo $file
   fi
done
exit 0
```

exit code, 0 means successful

```
% /bin/bash first
% chmod +x first
%./first (make sure . is include in PATH parameter)
```

Syntax

- Variables
- Conditions
- Control
- Lists
- Functions
- Shell Commands
- Result
- Document

Variables

- Variables needed to be declared, note it is case-sensitive (e.g. foo, FOO, Foo)
- Add ‘$’ for storing values

```
% salutation=Hello
% echo $salutation
Hello
% salutation=7+5
% echo $salutation
7+5
% salutation=“yes dear”
% echo $salutation
yes dear
% read salutation
Hola!
% echo $salutation
Hola!
```

Quoting

- Edit a “vartest.sh” file

```
#!/bin/bash

myvar=“Hi there”

echo $myvar
echo “$myvar”
echo `$myvar`
echo \$myvar

echo Enter some text
read myvar

echo ‘$myvar’ now equals $myvar
exit 0
```

Output

```
Hi there
Hi there
$myvar
$myvar
Enter some text
Hello world
$myvar now equals Hello world
```

Environment Variables

$HOME	home directory
$PATH	path
$PS1	(normally %)
$PS2	(normally >)
$$	process id of the script
$#	number of input parameters
$0	name of the script file
$IFS	separation character (white space)

- Use ‘env’ to check the value

Parameter

```
% IFS = ` `
% set foo bar bam
% echo "$@"
foo bar bam
% echo "$*"
foo bar bam
% unset IFS
% echo "$*"
foo bar bam
```

doesn't matter IFS

Parameter

Edit file ‘try_var’

```
#!/bin/bash
salutation=“Hello”
echo $salutation
echo “The program $0 is now running”
echo “The parameter list was $*”
echo “The second parameter was $2”
echo “The first parameter was $1”
echo “The user’s home directory is $HOME”
echo “Please enter a new greeting”
read salutation
echo $salutation
echo “The script is now complete”
exit 0
```

```
%./try_var foo bar baz
Hello
The program ./try_var is now running
The second parameter was bar
The first parameter was foo
The parameter list was foo bar baz
The user’s home directory is /home/jai
Please enter a new greeting
Hola
Hola
The script is now complete
```

Condition

need space !

- test or ‘ [‘

```
if test –f fred.c
then
...
fi
```

```
If [ -f fred.c
 ]
then
...
fi
```

```
if [ -f fred.c ];then
...
fi
```

```
expression1 –eq expression2
expression1 –ne expression2
expression1 –gt expression2
expression1 –ge expression2
expression1 -lt expression2
expression1 –le expression2
!expression

String1 = string2
String1 != string 2
-n string  (if not empty string)
-z string  (if empty string)
```

```
-d file   if directory
-e file   if exist
-f file   if file
-g file   if set-group-id
-r file   if readable
-s file   if size >0
-u file   if set-user-id
-w file   if writable
-x file   if executable
```

Control Structure

Syntax

```
if condition
then
  statement
else
  statement
fi
```

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
if [ $timeofday = “yes” ]; then
  echo “Good morning”
else
  echo “Good afternoon”
fi
exit 0
```

```
Is it morning? Please answer yes or no
yes
Good morning
```

Condition Structure

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
if [ $timeofday = “yes” ]; then
  echo “Good morning”
elif [ $timeofday = “no” ]; then
  echo “Good afternoon”
else
  echo “Sorry, $timeofday not recongnized. Enter yes or no”
   exit 1
fi
exit 0
```

Condition Structure

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
if [ “$timeofday” = “yes” ]; then
  echo “Good morning”
elif [ $timeofday = “no” ]; then
  echo “Good afternoon”
else
  echo “Sorry, $timeofday not recongnized. Enter yes or no”
   exit 1
fi
exit 0
```

If input “enter” still returns Good morning

Loop Structure

Syntax

```
for variable
do
  statement
done
```

```
#!/bin/bash

for foo in bar fud 43
do
   echo $foo
done
exit 0

bar
fud
43
```

How to output as bar fud 43?
Try change for foo in “bar fud 43”
This is to have space in variable

Loop Structure

- Use wildcard ‘*’

```
#!/bin/bash

for file in $(ls f*.sh); do
  lpr $file
done
exit 0
```

Print all f*.sh files

Loop Structure

Syntax

```
while condition
do
  statement
done
```

Syntax

```
until condition
do
  statement
done
```

Note: condition is
Reverse to while
How to re-write
previous sample?

```
#!/bin/bash
for foo in 1 2 3 4 5 6 7 8 9 10
do
  echo “here we go again”
done
exit 0
```

```
#!/bin/bash
foo = 1
while [ “$foo” –le 10 ]
do
  echo “here we go again”
  foo = $foo(($foo+1))
done
exit 0
```

Case Statement

Syntax

```
case variable in\
 pattern [ | pattern ] …) statement;;
 pattern [ | pattern ] …) statement;;
 …
esac
```

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
case “$timeofday” in
  yes) echo “Good Morning”;;
  y)   echo “Good Morning”;;
  no)  echo “Good Afternoon”;;
  n)   echo “Good Afternoon”;;
  * )  echo “Sorry, answer not recongnized”;;
esac
exit 0
```

Case Statement

- A much “cleaner” version

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
case “$timeofday” in
  yes | y | Yes | YES ) echo “Good Morning”;;
  n* | N* )     echo “Good Afternoon”;;
  * )           echo “Sorry, answer not recongnized”;;
esac
exit 0
```

But this has a problem, if we enter ‘never’ which obeys n* case and prints “Good Afternoon”

Case Statement

```
#!/bin/bash
echo “Is it morning? Please answer yes or no”
read timeofday
case “$timeofday” in
  yes | y | Yes | YES )
               echo “Good Morning”
               echo “Up bright and early this morning”
               ;;
  [nN]*)
               echo “Good Afternoon”;;
  *)
               echo “Sorry, answer not recongnized”
               echo “Please answer yes of no”
               exit 1
               ;;
esac
exit 0
```

List

- AND (&&)
  statement1 **&&** statement2 **&&** statement3 …

```
#!/bin/sh
touch file_one
rm –f file_two

if [ -f file_one ] && echo “Hello” && [-f file_two] && echo “ there”
then
  echo “in if”
else
  echo “in else”
fi
exit 0
```

Check if file exist if not then create one

Remove a file

Output

```
Hello
in else
```

List

- OR (||)

  statement1 **||** statement2 **||** statement3 …

```
#!/bin/bash

rm –f file_one
if [ -f file_one ] || echo “Hello” || echo “ there”
then
  echo “in if”
else
  echo “in else”
fi

exit 0
```

Output

```
Hello
in else
```

Statement Block

- Use multiple statements in the same place

```
get_comfirm && {
  grep –v “$cdcatnum” $stracks_file > $temp_file
  cat $temp_file > $tracks_file
  echo
  add_record_tracks
}
```

Function

- You can define functions for “structured” scripts

```
function_name() {
  statements
}
```

```
#!/bin/bash
foo() {
  echo “Function foo is executing”
}
echo “script starting”
foo
echo “script ended”
exit 0
```

Output

```
script starting
Function foo is executing
Script ended
```

You need to define a function before using it
The parameters $*,$@,$#,$1,$2 are replaced by local value
if function is called and return to previous after function is finished

Function

define local variable

```
#!/bin/bash
sample_text=“global variable”
foo() {
  local sample_text=“local variable”
  echo “Function foo is executing”
  echo $sample_text
}
echo “script starting”
echo $sample_text
foo

echo “script ended”
echo $sample_text

exit 0
```

Output?

Check the scope of the variables

Function

- Use return to pass a result

```
#!/bin/bash
yes_or_no() {
  echo “Is your name $* ?”
  while true
  do
    echo –n “Enter yes or no:”
    read x
    case “$x” in
      y | yes ) return 0;;
      n | no ) return 1;;
      * ) echo “Answer yes or no”
    esac
    done
}
```

```
echo “Original parameters are $*”
if yes_or_no “$1”
then
  echo “Hi $1, nice name”
else
  echo “Never mind”
fi
exit 0
```

Output

```
./my_name Jai Phull
Original parameters are Jai Phull
Is your name Jai?
Enter yes or no: yes
Hi Jai, nice name.
```

Command

- External:
  use interactively
- Internal:
- only in script
- **break**
  skip loop

```
#!/bin/bash
rm –rf fred*
echo > fred1
echo > fred2
mkdir fred3
echo > fred4

for file in fred*
do
  if [ -d “$file” ] ; then
      break;
  fi
done
echo first directory starting fred was $file
rm –rf fred*
exit 0
```

Command

- :   treats it as true

```
#!/bin/bash

rm –f fred
if [ -f fred ]; then
    :
else
    echo file fred did not exist
fi

exit 0
```

Command

- continue   continues next iteration

```
#!/bin/bash
rm –rf fred*
echo > fred1
echo > fred2
mkdir fred3
echo > fred4
for file in fred*
do
  if [ -d “$file” ]; then
      echo “skipping directory $file”
      continue
  fi
    echo file is $file
done
rm –rf fred*
exit 0
```

Command

- . ./shell_script   execute shell_script

```
classic_set
#!/bin/bash
verion=classic
PATH=/usr/local/old_bin:/usr/bin:/bin:.
PS1=“classic> ”
latest_set
#!/bin/sh
verion=latest
PATH=/usr/local/new_bin:/usr/bin:/bin:.
PS1=“latest version> ”
```

```
% . ./classic_set
classic> echo $version
classic
Classic> . latest_set
latest
latest version>
```

Command

- echo   print string
- -n do not output the trailing newline
- -e enable interpretation of backslash escapes
  - \0NNN the character whose ACSII code is NNN
  - \\ backslash
  - \a alert
  - \b backspace
  - \c suppress trailing newline
  - \f form feed
  - \n newline
  - \r carriage return
  - \t horizontal tab
  - \v vertical tab

Try these

```
% echo –n “string to \n output”

% echo –e “string to \n output”
```

Command

- eval	evaluate the value of a parameter
  similar to an extra ‘$’

```
% foo=10
% x=foo
% y=‘$’$x
% echo $y
```

Output is $foo

```
% foo=10
% x=foo
% eval y=‘$’$x
% echo $y
```

Output is 10

Command

- exit *n*   ending the script
- 0 means success
- 1 to 255 means specific error code
- 126 means not executable file
- 127 means no such command
- 128 or >128 signal

```
#!/bin/bash
if [ -f .profile ]; then
  exit 0
fi
exit 1

Or % [ -f .profile ] && exit 0 || exit 1
```

Command

- export gives a value to a parameter

This is ‘export2’

```
#!/bin/bash
echo “$foo”
echo “$bar”
```

This is ‘export1’

```
#!/bin/bash
foo=“The first meta-syntactic variable”
export bar=“The second meta-syntactic variable”

export2
```

Output is

```
%export1

The second-syntactic variable
%
```

Command

- expr   evaluate expressions

```
%x=`expr $x + 1`
```
(Assign result value expr $x+1 to x)

Also can be written as

```
%x=$(expr $x + 1)
```

Expr1 | expr2 (or)	expr1 != expr2
Expr1 & expr2 (and)	expr1 + expr2
Expr1 = expr2	expr1 – expr2
Expr1 > expr2	expr1 * expr2
Expr1 >= expr2	expr1 / expr2
Expr1 < expr2	expr1 % expr2 (module)
Expr1 <= expr2	

Command

- printf format and print data
- Escape sequence

\\	backslash
\a	beep sound
\b	backspace
\f	form feed
\n	newline
\r	carriage return
\t	tab
\v	vertical tab

- Conversion specifier

%d	decimal
%c	character
%s	string
%%	print %

```
% printf “%s\n” hello
Hello
% printf “%s %d\t%s” “Hi There” 1
5 people
Hi There 15     people
```

Command

return	return a value
set	set parameter variable

```
#!/bin/bash

echo the date is $(date)
set $(date)
echo The month is $2

exit 0
```

Command

- Shift shift parameter once, $2 to $1, $3 to $2, and so on

```
#!/bin/bash

while [ “$1” != “” ]; do
     echo “$1”
     shift
done

exit 0
```

Command

- trap action after receiving signal

```
trap command signal
```

- signal / explain

signal	explain
HUP (1)	hung up
INT (2)	interrupt (Crtl + C)
QUIT (3)	Quit (Crtl + \)
ABRT (6)	Abort
ALRM (14)	Alarm
TERM (15)	Terminate

Command

```
#!/bin/bash
trap ‘rm –f /tmp/my_tmp_file_$$’ INT
echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$
echo “press interrupt (CTRL-C) to interrupt …”
while [ -f /tmp/my_tmp_file_$$ ]; do
    echo File exists
    sleep 1
done
echo The file no longer exists
trap INT
echo creating file /tmp/my_tmp_file_$$
date > /tmp/my_tmp_file_$$
echo “press interrupt (CTRL-C) to interrupt …”
while [ -f /tmp/my_tmp_file_$$ ]; do
    echo File exists
    sleep 1
done
echo we never get there
exit 0
```

Command

```
creating file /tmp/my_file_141
press interrupt (CTRL-C) to interrupt …
File exists
File exists
File exists
File exists
The file no longer exists
Creating file /tmp/my_file_141
Press interrupt (CTRL-C) to interrupt …
File exists
File exists
File exists
File exists
```

Command

Unset remove parameter or function

```
#!/bin/bash

foo=“Hello World”
echo $foo

unset $foo
echo $foo
```

Pattern Matching

- find search for files in a directory hierarchy

  find `[path] [options] [tests] [actions]`

options

-depth	find content in the directory
-follow	follow symbolic links
-maxdepths N	fond N levels directories
-mount	do not find other directories

tests

-atime N	accessed N days ago
-mtime N	modified N days ago
-new otherfile	name of a file
-type X	file type X
-user username	belong to username

Pattern Matching

operator

!	-not	test reverse
-a	-and	test and
-o	-or	test or

action

-exec command	execute command
-ok command	confirm and exectute command
-print	print
-ls	ls –dils

Find files newer than while2 then print

```
% find . –newer while2 -print
```

Pattern Matching

Find files newer than while2 then print only files

```
% find . –newer while2 –type f –print
```

Find files either newer than while2, start with ‘_’

```
% find . \( -name “_*” –or –newer while2 \) –type f
  –print
```

Find files newer than while2 then list files

```
% find . –newer while2 –type f –exec ls –l {} \;
```

Pattern Matching

- grep print lines matching a pattern
  (General Regular Expression Parser)

  grep [options] PATTERN [FILES]

option

-c	print number of output context
-E	Interpret PATTERN as an extended regular expression
-h	Supress the prefixing of filenames
-i	ignore case
-l	surpress normal output
-v	invert the sense of matching

```
% grep in words.txt
% grep –c in words.txt words2.txt
% grep –c –v in words.txt words2.txt
```

Regular Expressions

- a **regular expression** (abbreviated as **regexp** or **regex**, with plu ral forms **regexps**, **regexes**, or **regexen**) is a string that describe s or matches a set of strings, according to certain syntax rules.
- Syntax
  - ^ Matches the start of the line
  - $ Matches the end of the line
  - . Matches any single character
  - [] Matches a single character that is contained within the brack ets
  - [^] Matches a single character that is not contained within the brackets
  - () Defines a "marked subexpression”
  - {*x*,*y*}Match the last "block" at least *x* and not more than *y* time s

Regular Expressions

- Examples:
  - ".at" matches any three-character string like *ha t*, *cat* or *bat*
  - "[hc]at" matches *hat* and *cat*
  - "[^b]at" matches all the matched strings from the regex ".at" except *bat*
  - "^[hc]at" matches *hat* and *cat* but only at the beginning of a line
  - "[hc]at$" matches *hat* and *cat* but only at the e nd of a line

Regular Expressions

POSIX class	**similar to**	**meaning**
[:upper:]	[A-Z]	uppercase letters
[:lower:]	[a-z]	lowercase letters
[:alpha:]	[A-Za-z]	upper- and lowercase letters
[:alnum:]	[A-Za-z0-9]	digits, upper- and lowercase letters
[:digit:]	[0-9]	digits
[:xdigit:]	[0-9A-Fa-f]	hexadecimal digits
[:punct:]	[.,!?:...]	punctuation
[:blank:]	[\t]	space and TAB characters only
[:space:]	[\t\n\r\f\v]	blank (whitespace) characters
[:cntrl:]		control characters
[:graph:]	[^ \t\n\r\f\v]	printed characters
[:print:]	[^\t\n\r\f\v]	printed characters and space

- Example: [[:upper:]ab] should only match the uppercase letters an d lowercase 'a' and 'b'.

Regular Expressions

- **POSIX modern (extended) regular expressions**
- The more modern "extended" regular expressions can often be used with modern Unix utilities by including the command line flag "-E".

+	Match one or more times
?	Match at most once
*	Match zero or more
{n}	Match n times
{n,}	Match n or more times
{n,m}	Match n to m times

Regular Expressions

- Search for lines ending with “e”

```
% grep e$ words2.txt
```

- Search for “a”

```
% grep a[[:blank:]] word2.txt
```

- Search for words starting with “Th.”

```
% grep Th.[[:blank:]] words2.txt
```

- Search for lines with 10 lower case characters

```
% grep –E [a-z]\{10\} words2.txt
```

Command

- $(command) to execute command in a script
- Old format used “`” but it can be confused with “’ ”

```
#!/bin/bash
echo The current directory is $PWD
echo the current users are $(who)
```

Arithmetic Expansion

- Use $((…)) instead of expr to evaluate arithmetic equation

```
#!/bin/bash
x=0
while [ “$x” –ne 10]; do
    echo $x
    x=$(($x+1))
done

exit 0
```

Parameter Expansion

- Parameter Assignment

```
foo=fred
echo $foo
```

${param:-default} set default if null
${#param} length of param
${param%word} remove smallest suffix pattern
${param%%word} remove largest suffix pattern
${param#word} remove smallest prefix pattern
${param##word} remove largest prefix pattern

```
#!/bin/bash
for i in 1 2
do
  my_secret_process $i_tmp
done
```

Gives result
“mu_secret_process:
too few arguments”

```
#!/bin/bash
for i in 1 2
do
  my_secret_process ${i}_tmp
done
```

Parameter Expansion

```
#!/bin/bash
unset foo
echo ${foo:-bar}

foo=fud
echo ${foo:-bar}

foo=/usr/bin/X11/startx
echo ${foo#*/}
echo ${foo##*/}

bar=/usr/local/etc/local/networks
echo ${bar%local*}
echo ${bar%%local*}

Exit 0
```

Output

```
bar
fud
usr/bin/X11/startx
startx
/usr/local/etc
/usr
```

Here Documents

- A here document is a special-purpose code block, starts with <<

```
#!/bin/bash
cat <<!FUNKY!
hello
this is a here
document
!FUNCKY!
exit 0
```

```
#!/bin/bash
ed a_text_file <<HERE
3
d
.,\$s/is/was/
w
q
HERE
exit 0
```

a_text_file

```
That is line 1
That is line 2
That is line 3
That is line 4
```

Output

```
That is line 1
That is line 2
That was line 4
```

Debug

- sh –n<script> set -o noexec check syntax
  set –n
- sh –v<script> set -o verbose echo command before
  set –v
- sh –x<script> set –o trace echo command after
  set –x
  set –o nounset gives error if undefined
  set –x

```
set –o xtrace
set +o xtrace
trap ‘echo Exiting: critical variable =$critical_variable’
  EXIT
```

References

- Bash Beginners Guide (http://tldp.org/LDP/Bash-Beginners-Guide/)