Linux/Unix Shell Program/Script

Dr. Chokchai (Box) Leangsuksun

Louisiana Tech University

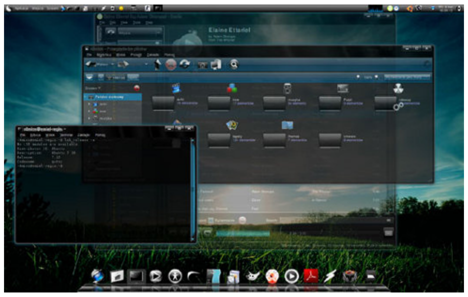
Original slides were created by Dr. Yao-Yuan Chuang www.math.ntu.edu.tw

Outline

- User Interface – shell commands
- Shell Script
  - Basic
  - Syntax
    - Lists
    - Functions
    - Command Execution
    - Here Documents
    - Debug
  - Regular Expression

USER Interface

- Command Line Interface
  - Shell commands
  - C-shell, tsh-shell, bourne shell etc..
- Graphic User Interface
  - GNOME, KDE etc..



CLI or Shell

- Command Line Interface
- The shell is a command interpreter
- It provides the interface between a user and the operating system via command line
- Shell commands. Eg. ls, cd, pwd etc
- Various shells: C-shell, tsh-shell, bourne shell etc..
- When you log in to a Unix system, a shell starts running. You interact with the default shell

Various shell programs

Shell name	Program (Command) name
rc	rc
Bourne Shell	sh
C Shell	csh
Bourne Again Shell	bash
Z shell	zsh
Korn Shell	ksh
TC	tcsh

Shell Script/Program

- A collection of user or system commands in an executable file
- For example,

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

- For routine tasks, such as sys admin or new tools/utilities, without writing programs
- May not be efficient, typically for prototyping the ideas

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 !)
```

Concept of a Program

- Variables to store value
- Control structure
  - Conditional statement
  - Loop
  - Decision making (case)
  - Functions
  - List

Syntax

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

Writing a Script

- Use text editor to generate the "first" file

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

First line is a must

exit code, 0 means successful

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

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!
```

Louisiana Tech University

Quoting

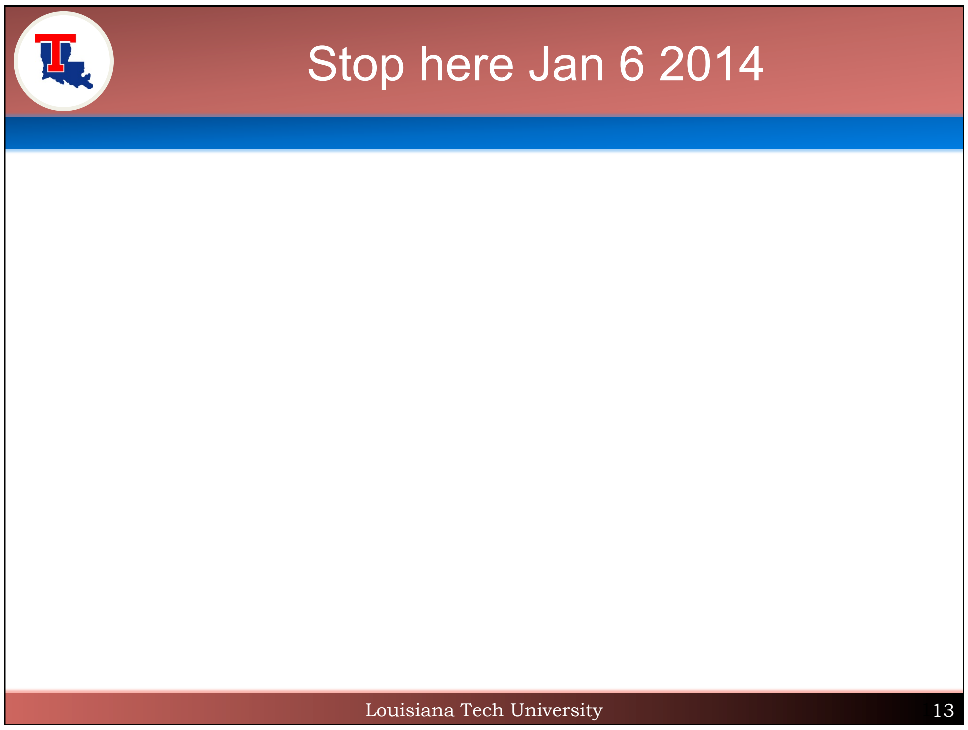
- Edit a "vartest.sh" file

```
#!/bin/sh

myvar="Hi there"

echo $myvar
echo "$myvar"

echo Enter some text
read myvar

echo '$myvar' now equals $myvar
exit 0
```

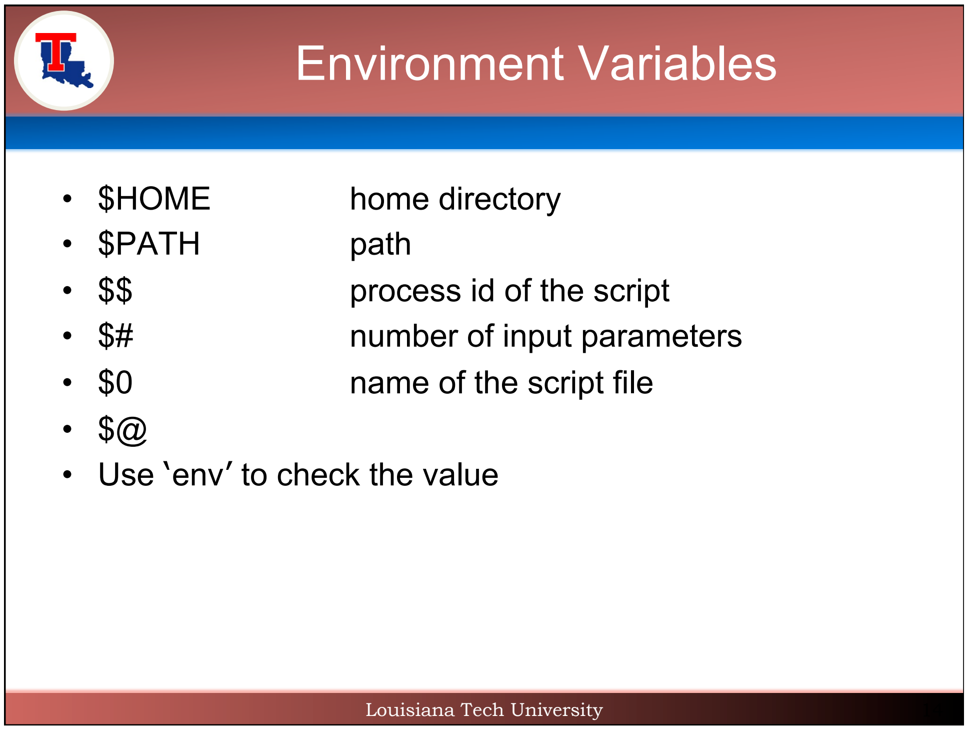
Output

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

Louisiana Tech University



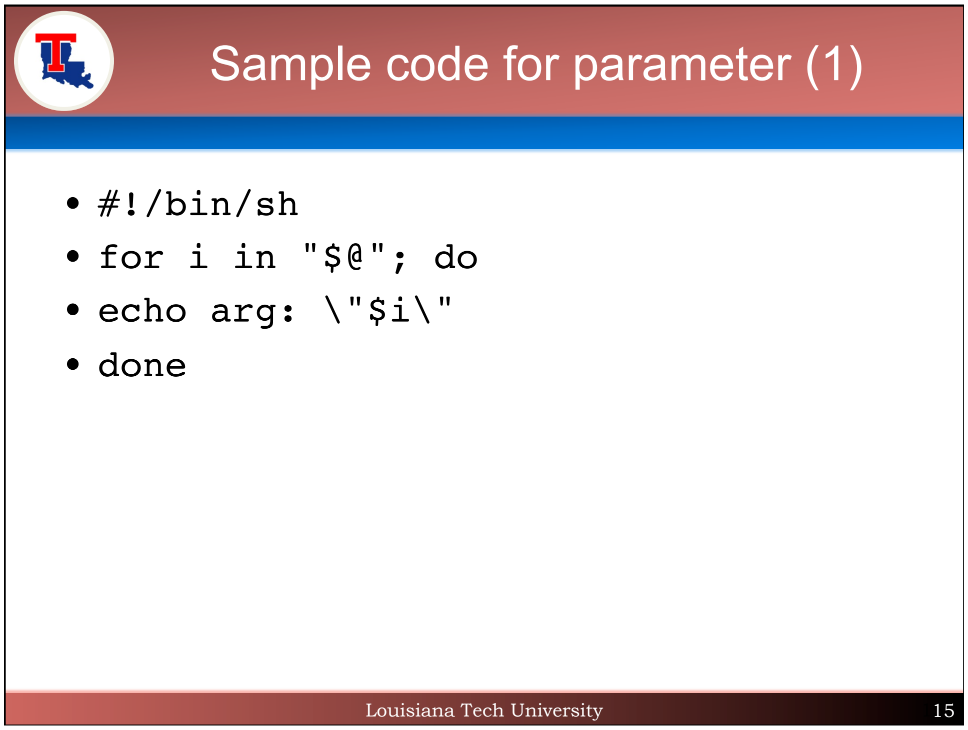
Stop here Jan 6 2014



Environment Variables

• $HOME	home directory
• $PATH	path
• $$	process id of the script
• $#	number of input parameters
• $0	name of the script file
• $@	

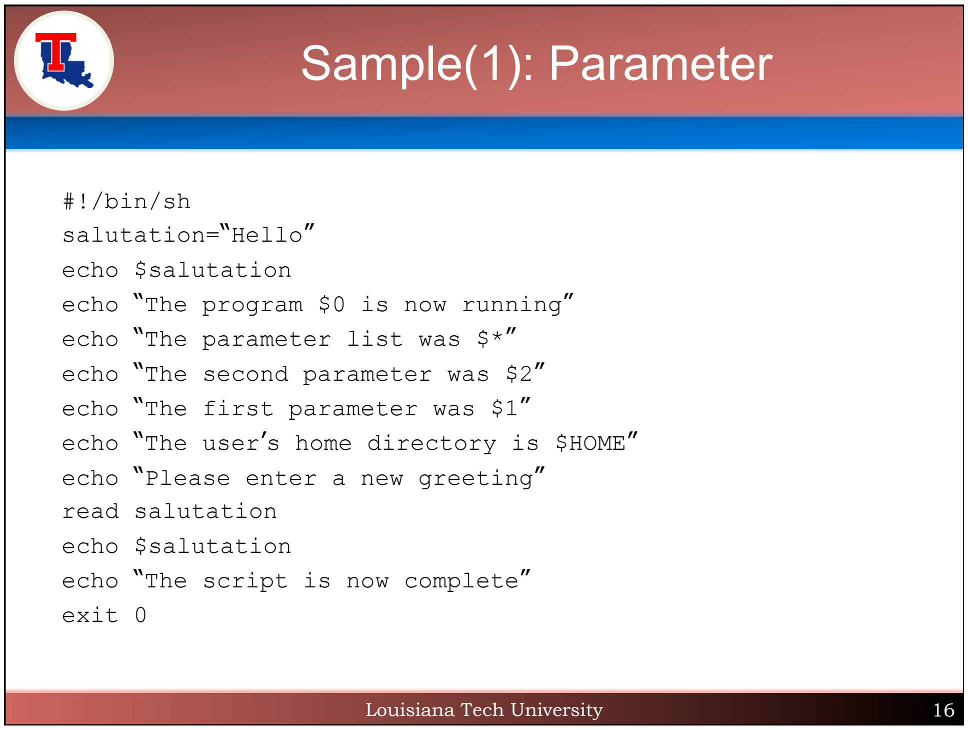
- Use ‘env’ to check the value



Sample code for parameter (1)

- `#!/bin/sh`
- `for i in "$@"; do`
- `echo arg: \"$i\"`
- `done`

Louisiana Tech University 15

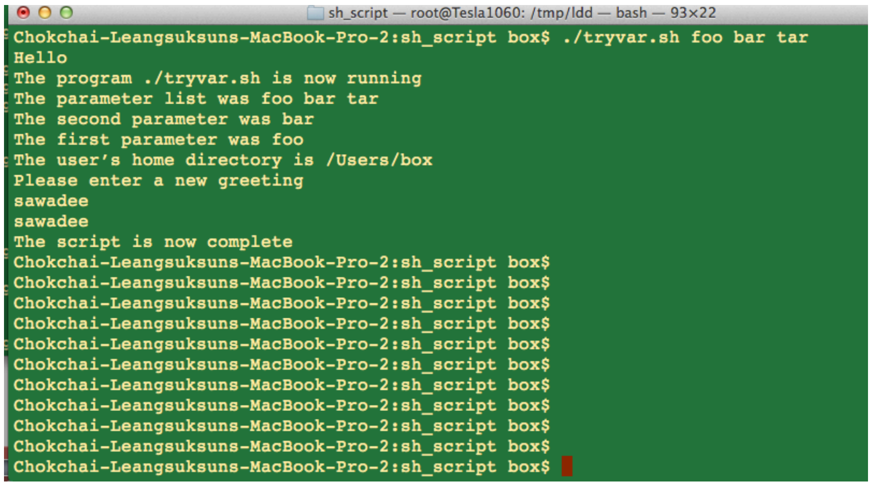


Sample(1): Parameter

```
#!/bin/sh
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
```

Louisiana Tech University 16

Run



sh_script — root@Tesla1060: /tmp/ldd — bash — 93×22

```
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$ ./tryvar.sh foo bar tar
Hello
The program ./tryvar.sh is now running
The parameter list was foo bar tar
The second parameter was bar
The first parameter was foo
The user's home directory is /Users/box
Please enter a new greeting
sawadee
sawadee
The script is now complete
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
Chokchai-Leangsuksuns-MacBook-Pro-2:sh_script box$
```

Louisiana Tech University 17

Conditional Statement

Syntax

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

Louisiana Tech University

Condition

- test or ‘ [’ … ‘]’

```
if test -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
```

Sample (1)

```
#!/bin/sh
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
```

Sample (2)

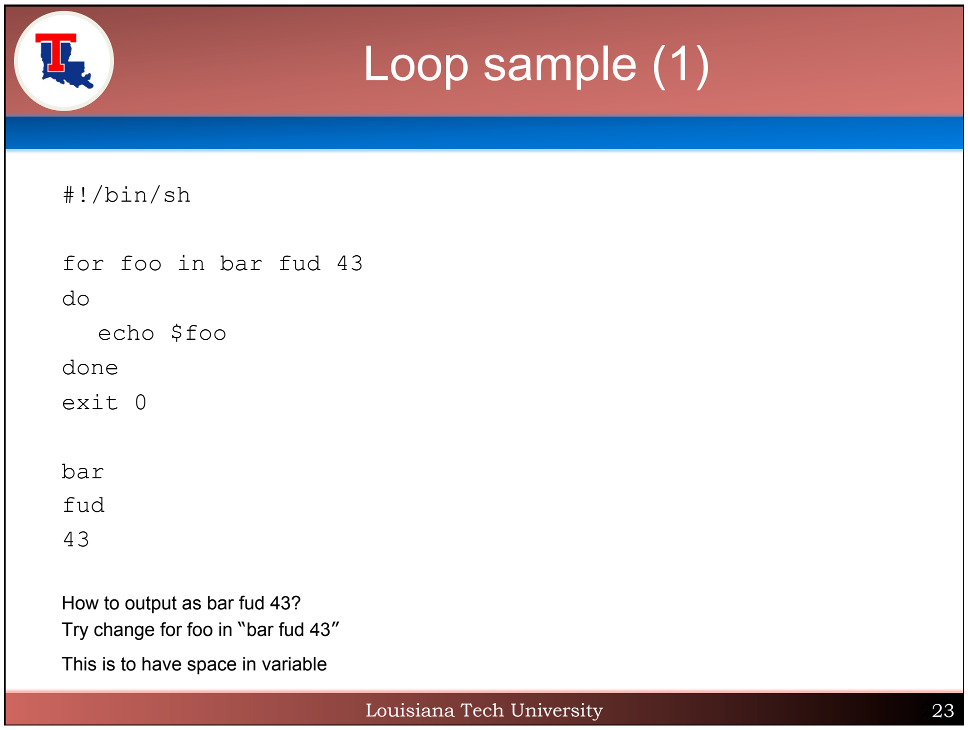
```
#!/bin/sh
echo “Is it morning? Please answer yes or no”
read timeofday
if [ “$timeofday” = “yes” ]; then
  echo “Good morning”
elif [ $timeofday = “no” ]; then
  echo “Good afternoon”
else
  echo “Sorry, $timeofday not recongnized. Enter yes or no”
   exit 1
fi
exit 0
```

If input “enter” still returns Good morning

Loop Structure

Syntax

```
for variable
do
  statement
done
```



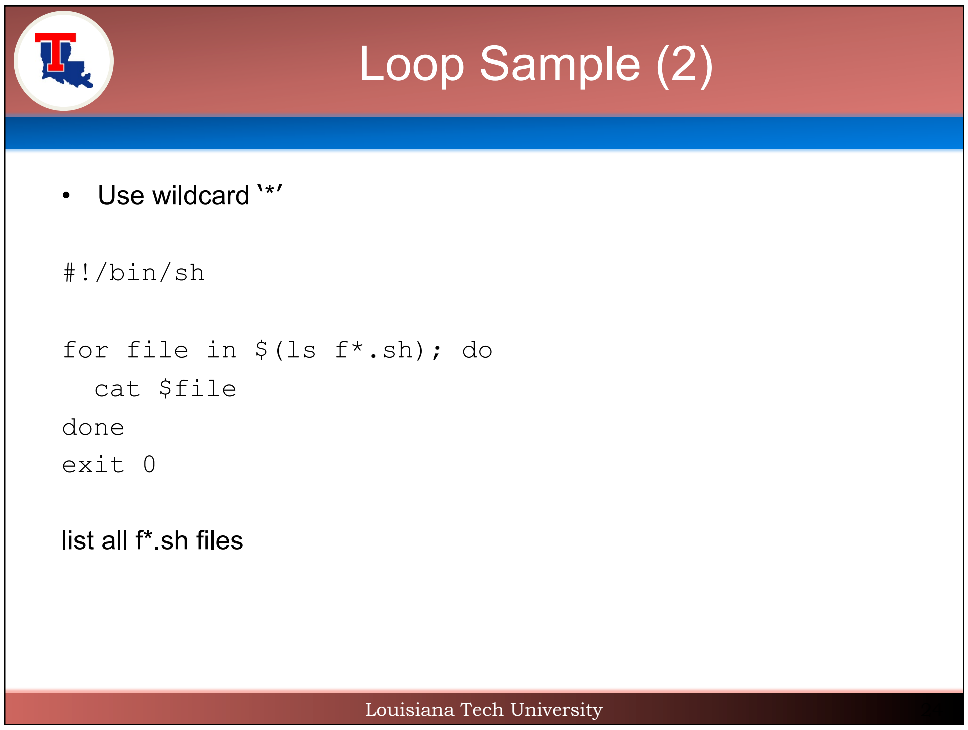
Loop sample (1)

```
#!/bin/sh

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

Louisiana Tech University 23



Loop Sample (2)

- Use wildcard '*'

```
#!/bin/sh

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

list all f*.sh files

Louisiana Tech University

Loop Structure Sample 3

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

More Loop Structures

Syntax

```
while condition
do
  statement
done
```

Syntax

```
until condition
do
 statement
done
```

Case Statement

Syntax

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

Case sample (1)

```
#!/bin/sh
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 recognized";;
esac
exit 0
```

Case sample (2)

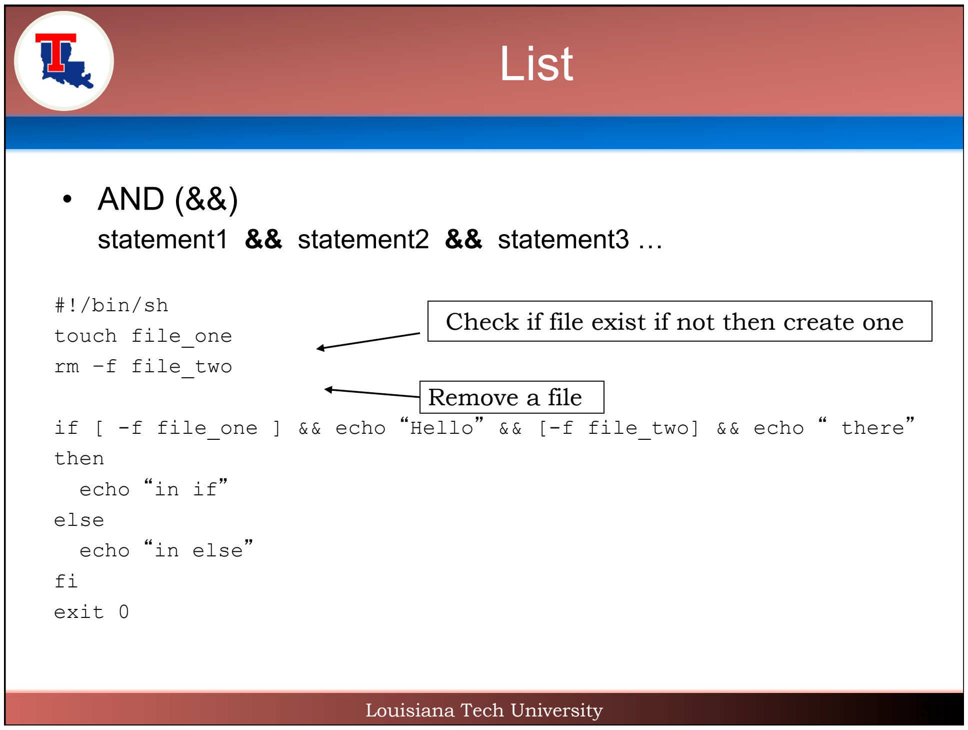
- A much "cleaner" version

```
#!/bin/sh
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 sample (3)

```
#!/bin/sh
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 (&&)

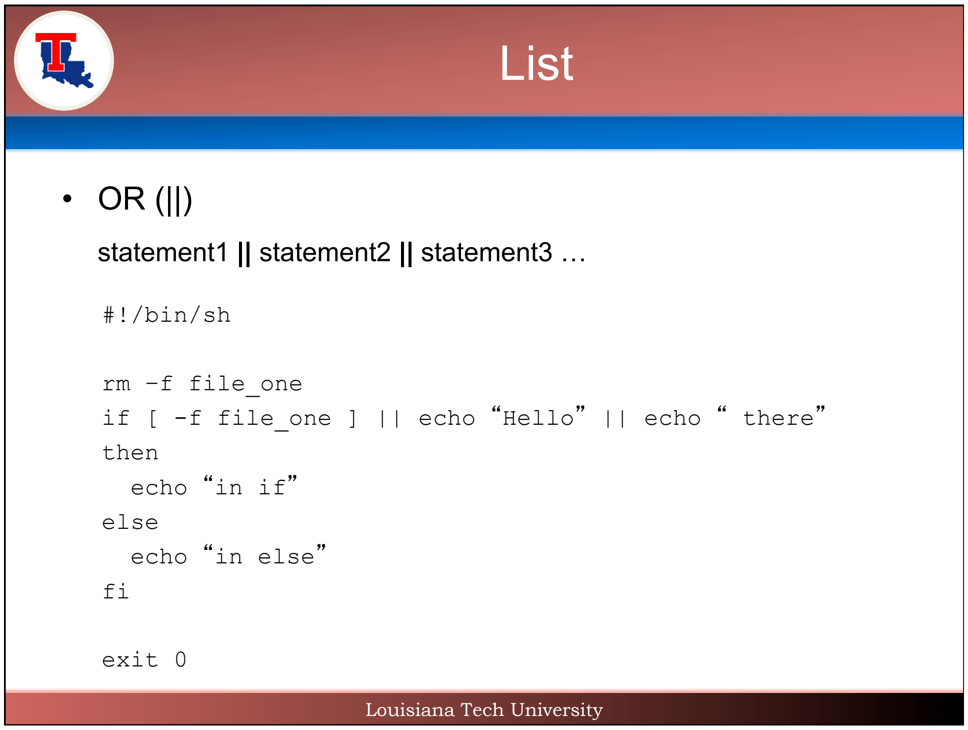
  statement1 **&&** statement2 **&&** statement3 …

```
#!/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

Louisiana Tech University



List

- OR (||)

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

```
#!/bin/sh

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

exit 0
```

Louisiana Tech University

Statement Block

- Use multiple statements in the same place

```
get_comfirm && {
  grep -v "$cdcatnum" $stracks_file > $temp_file
  cat $temp_file > $tracks_file
}
```

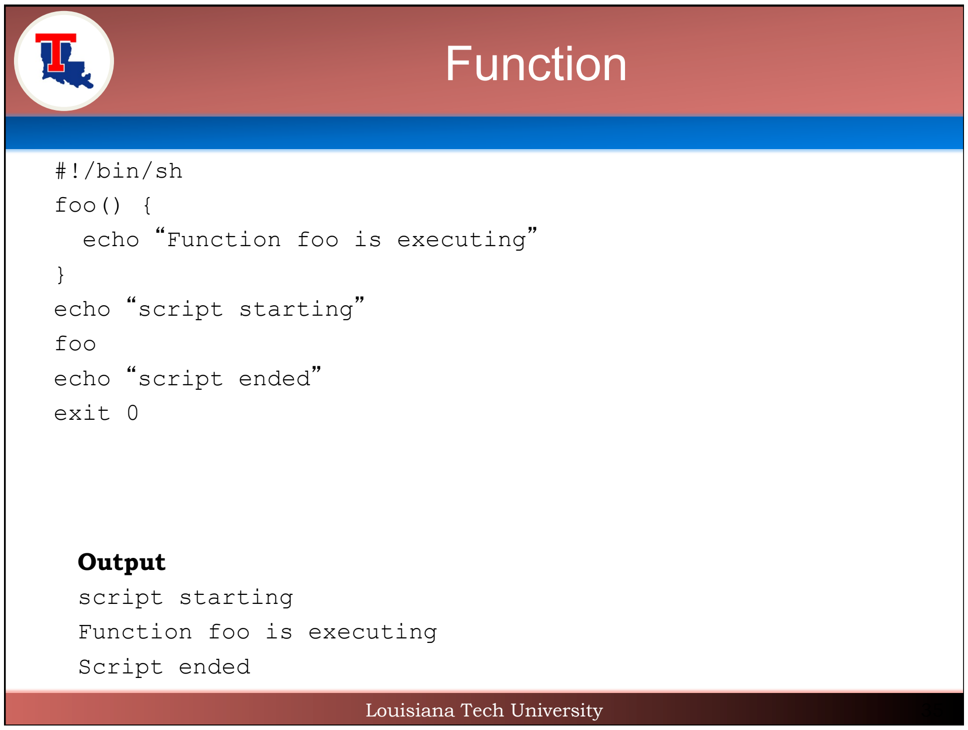
Function

- functions for "structured" scripts

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

- Must define a function before using it

- $@,$#,$1,$2 are replaced by local value, if function is called and return to previous after function is finished



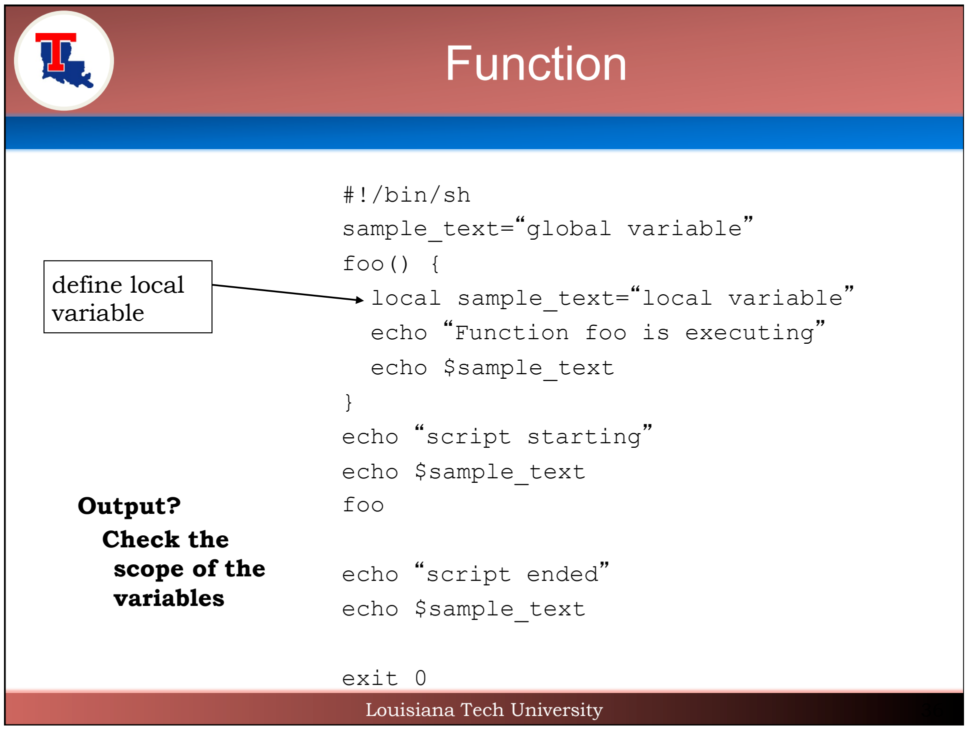
Function

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

Output

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

Louisiana Tech University



Function

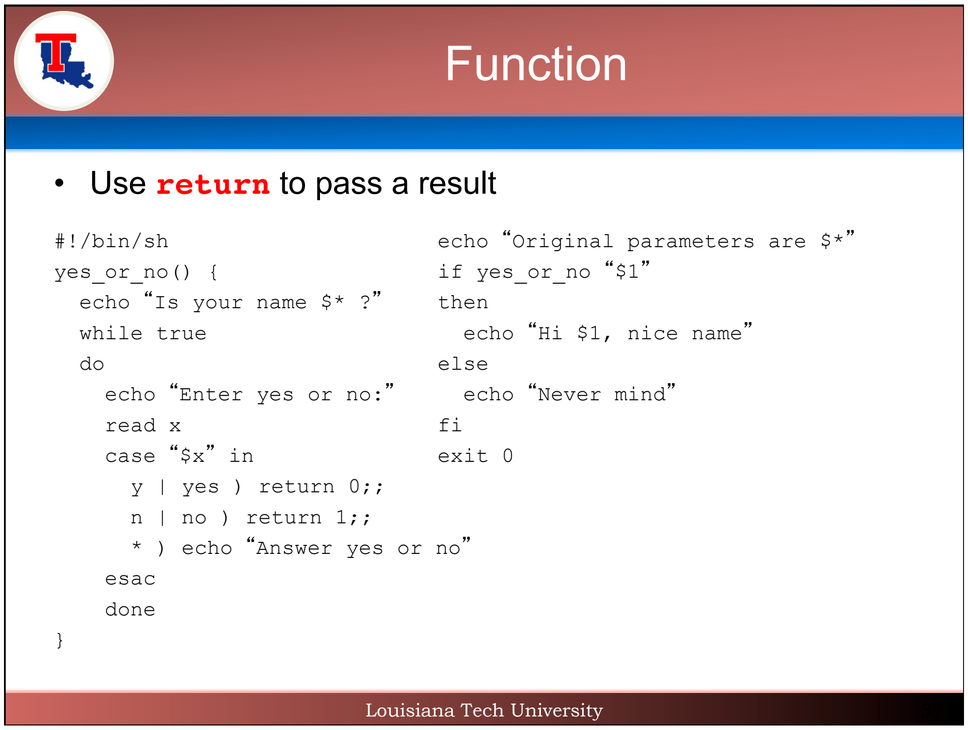
define local variable

```
#!/bin/sh
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

Louisiana Tech University



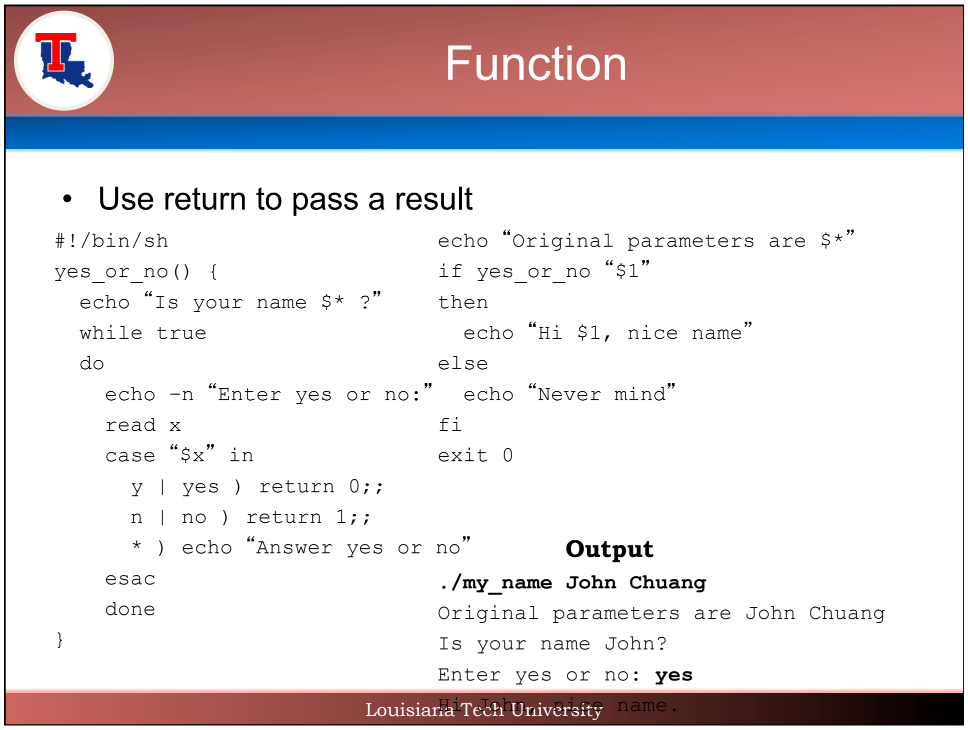
Function

- Use **return** to pass a result

```
#!/bin/sh
yes_or_no() {
  echo "Is your name $* ?"
  while true
  do
    echo "Enter yes or no:"
    read x
    case "$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
```

Louisiana Tech University



Function

- Use return to pass a result

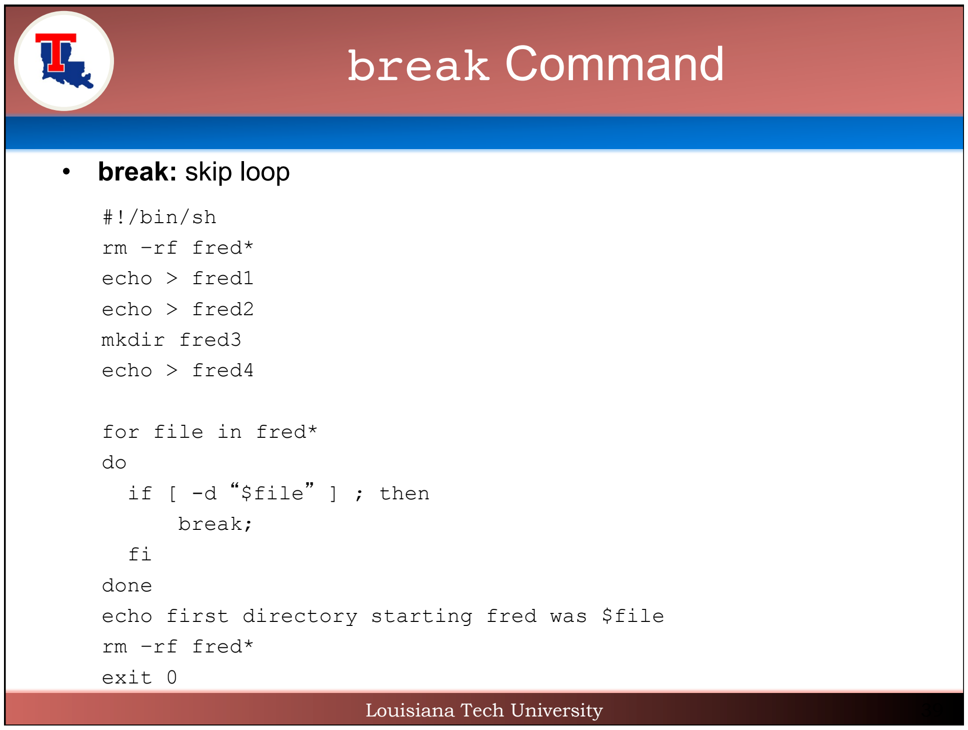
```
#!/bin/sh
yes_or_no() {
  echo "Is your name $* ?"
  while true
  do
    echo -n "Enter yes or no:"
    read x
    case "$x" in
      y | yes ) return 0;;
      n | no ) return 1;;
      * ) echo "Answer yes or no"
    esac
    done
}
```

```
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 John Chuang
Original parameters are John Chuang
Is your name John?
Enter yes or no: yes
Hi John, nice name.
```

Louisiana Tech University

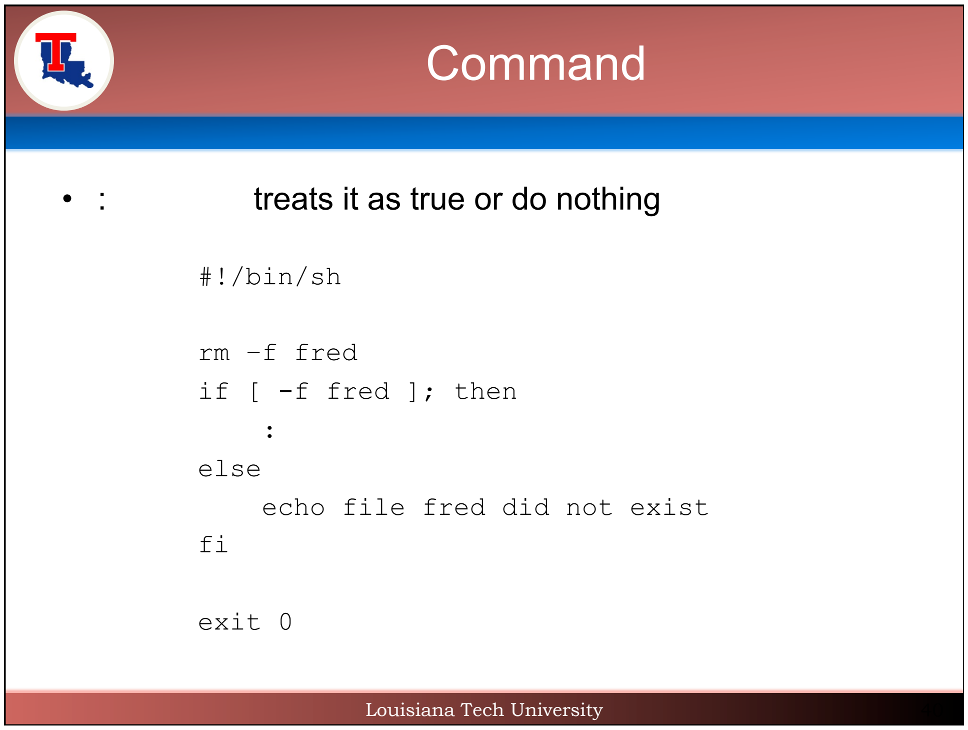


break Command

- **break:** skip loop

```
#!/bin/sh
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
```

Louisiana Tech University



Command

- :   treats it as true or do nothing

```
#!/bin/sh

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

exit 0
```

Louisiana Tech University

`continue` Command

- continue continues next iteration

```
#!/bin/sh
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
```

Louisiana Tech University

Command

- . ./shell_script execute shell_script

classic_set

```
#!/bin/sh
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>
```

Louisiana Tech University

echo Command

- **echo :** print string
  - \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

eval Command

- eval : evaluate the value of a parameter

```
% foo=10
% x=foo
% y='$'$x
% echo $y
```

Output is $foo

```
% foo=10
% x=foo
% eval y='$'$x
% echo $y
```

Output is 10

exit 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/sh
if [ -f .profile ]; then
  exit 0
fi
exit 1

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

Louisiana Tech University

export Command

- export gives a value to a parameter

This is 'export2' Output is

```
#!/bin/sh
echo "$foo"
echo "$bar"
```

```
%export1

The second-syntactic variable
%
```

This is 'export1'

```
#!/bin/sh
foo="The first meta-syntactic variable"
export bar="The second meta-syntactic variable"

export2
```

Louisiana Tech University

expr Command

- expr evaluate expressions

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

Also can be written as

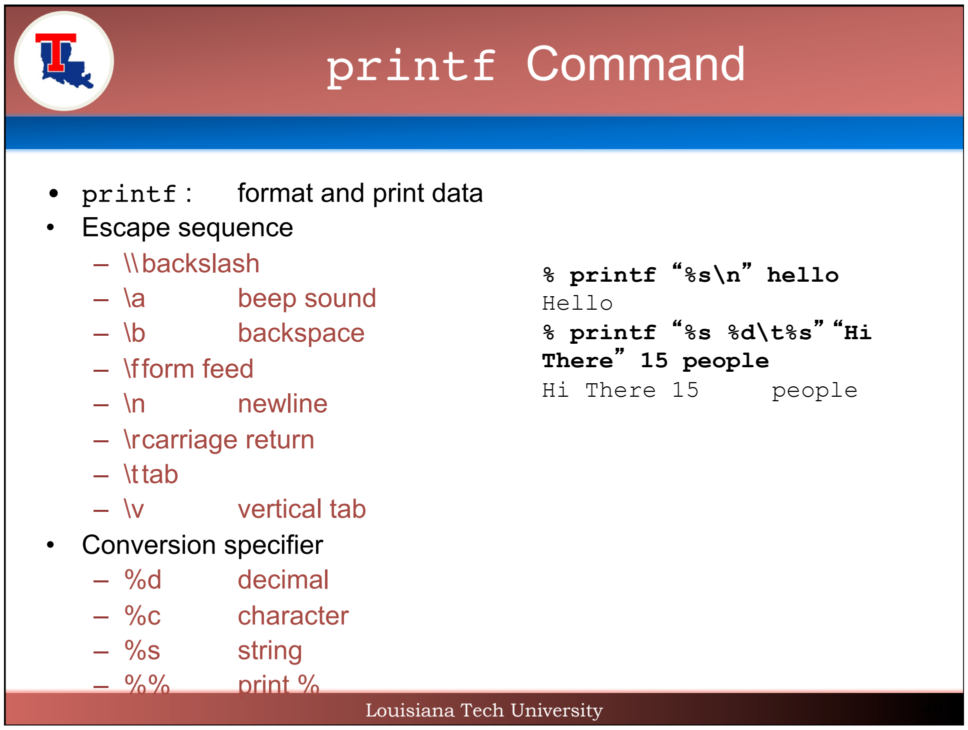
`%x=$(expr $x + 1)`

%let *val*=3*4

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	

Sample of Arithmetic/Assignment Statements

- Integer
  - **expr** 3 / 5
  - **expr** 3 * 5
  - **let** *val*=3*4 # see caution below!
  - ((*val* = 3 * 5))
  - ((*val* = 3 ** 5))

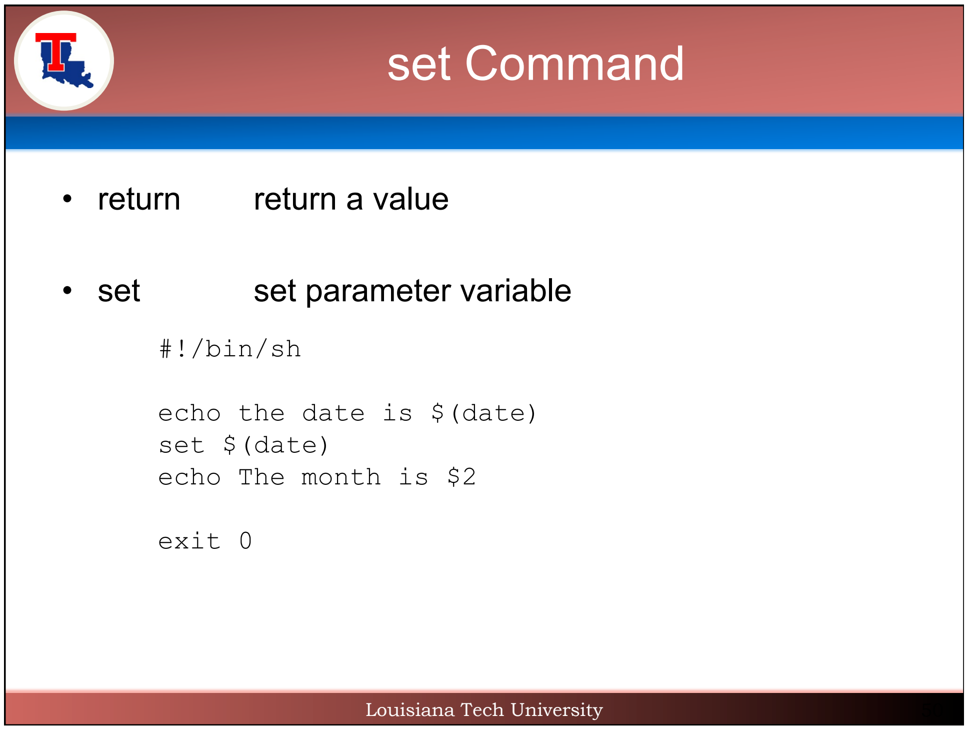


`printf` Command

- `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” 15 people
Hi There 15     people
```

Louisiana Tech University

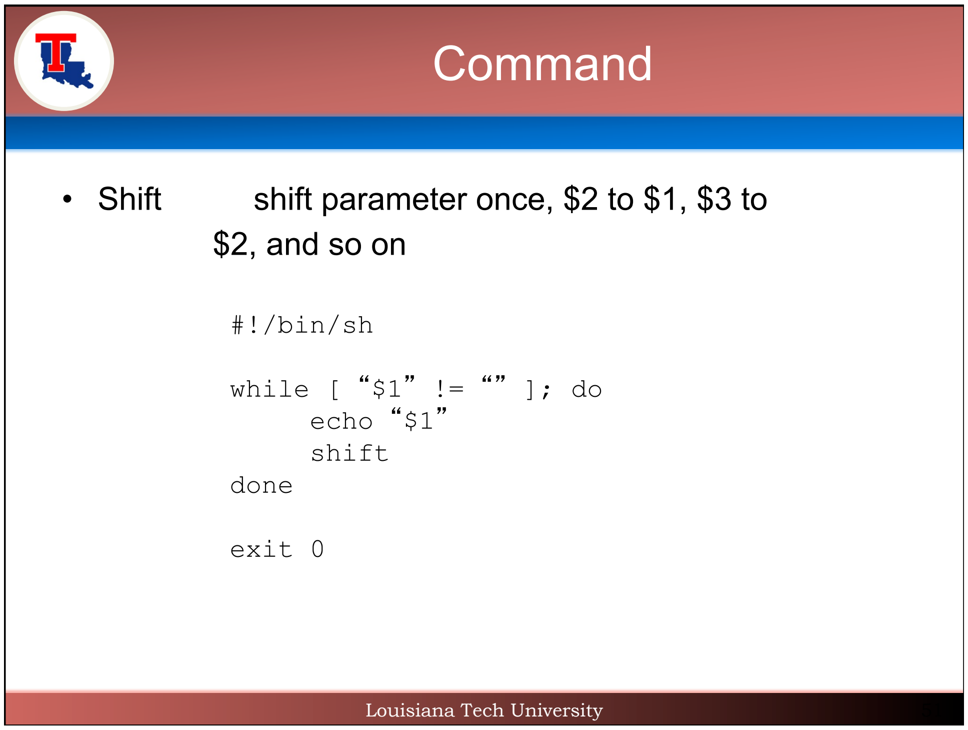


set Command

- return return a value
- set set parameter variable

```
#!/bin/sh

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

exit 0
```

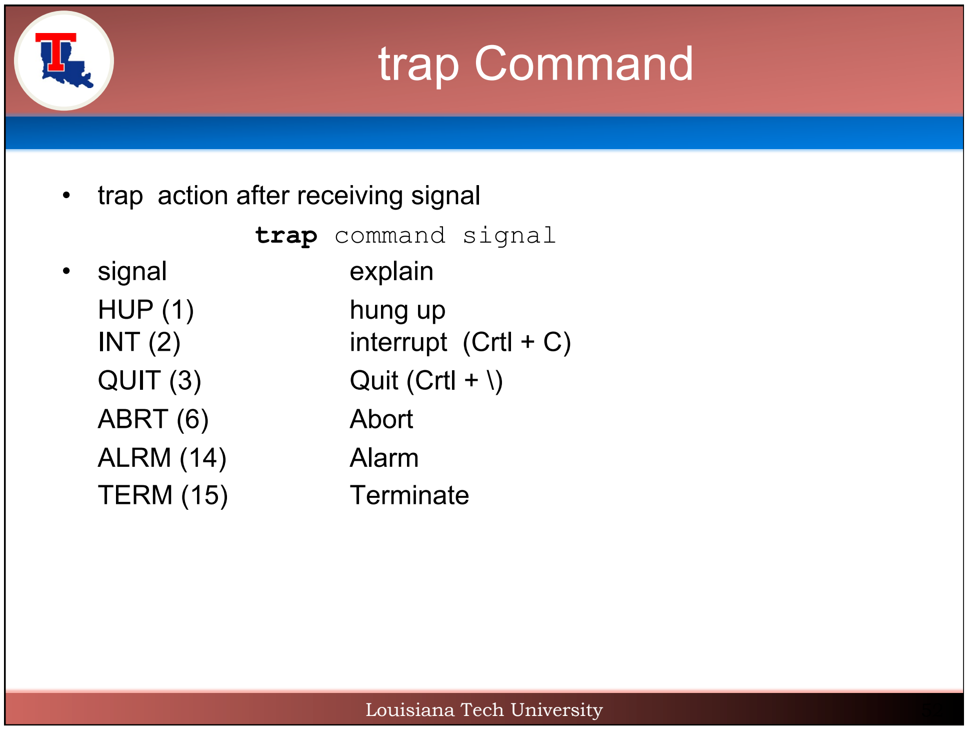
Louisiana Tech University



Command

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

```
#!/bin/sh

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

exit 0
```



trap Command

- trap action after receiving signal

```
trap command signal
```

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

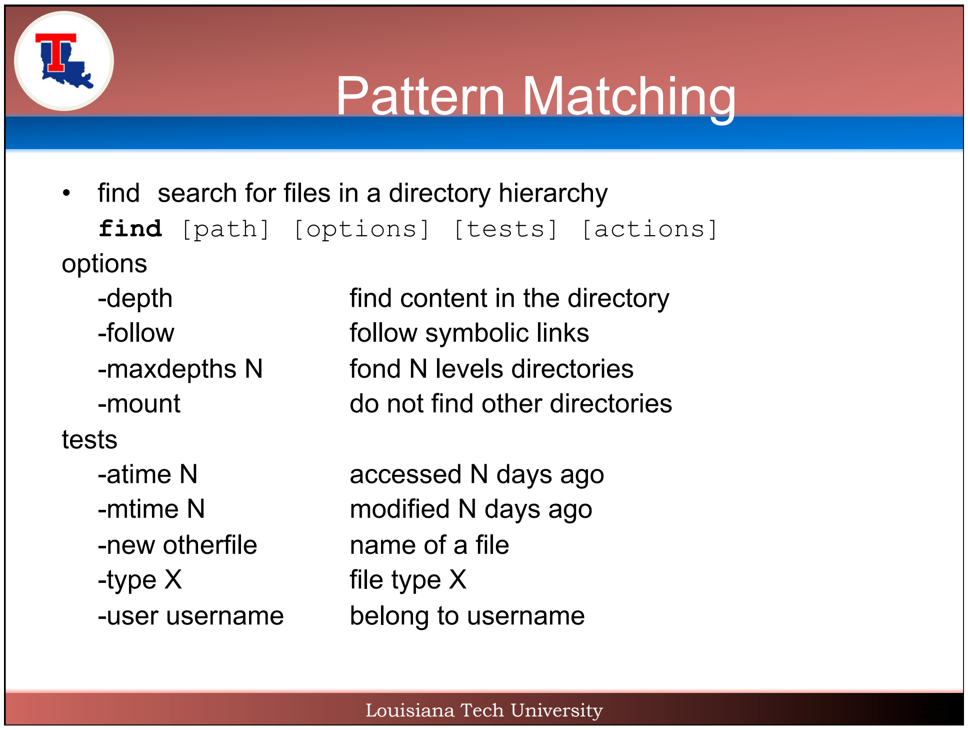
samples

```
#!/bin/sh
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
```

unset Command

unset	remove parameter or function

```
#!/bin/sh

foo="Hello World"
echo $foo

unset $foo
echo $foo
```



Pattern Matching

- find search for files in a directory hierarchy
  find [path] [options] [tests] [actions]

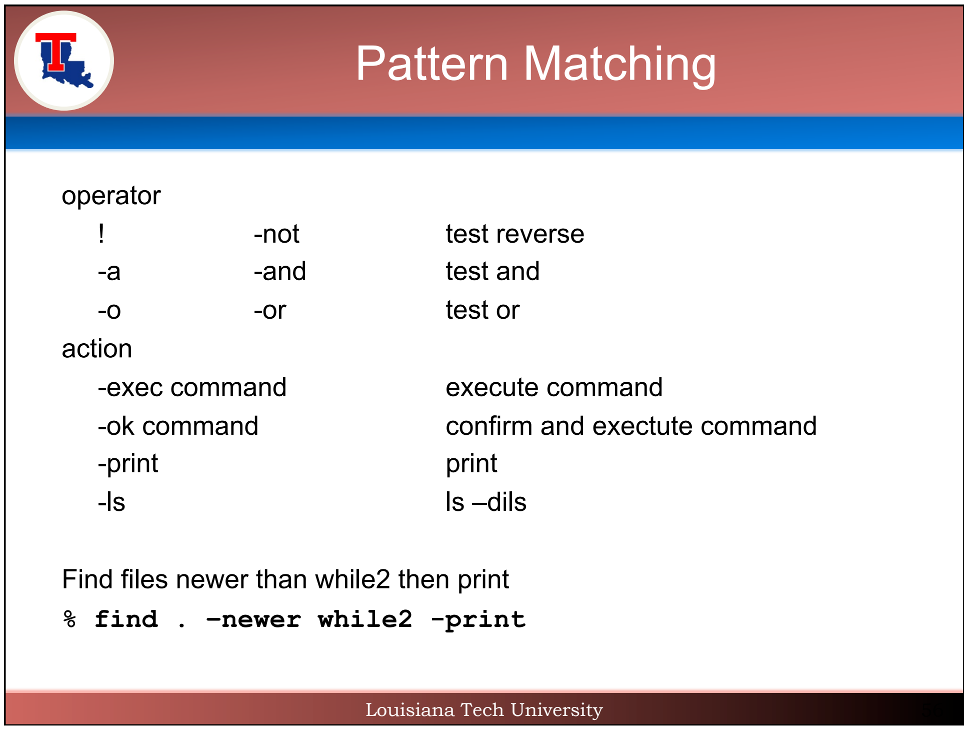
options

-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

Louisiana Tech University



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
```

Louisiana Tech University

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 plural forms **regexps**, **regexes**, or **regexen**) is a string that describes 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 brackets
  - [^] 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* times

Regular Expressions

- Examples:
  - ".at" matches any three-character string like *hat*, *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 end 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
```

Louisiana Tech University

Command

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

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

Louisiana Tech University

Arithmetic Expansion

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

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

Gives result "mu_secret_process: too few arguments"

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

Parameter Expansion

```
#!/bin/sh
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
```

Louisiana Tech University

Here Documents

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

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

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

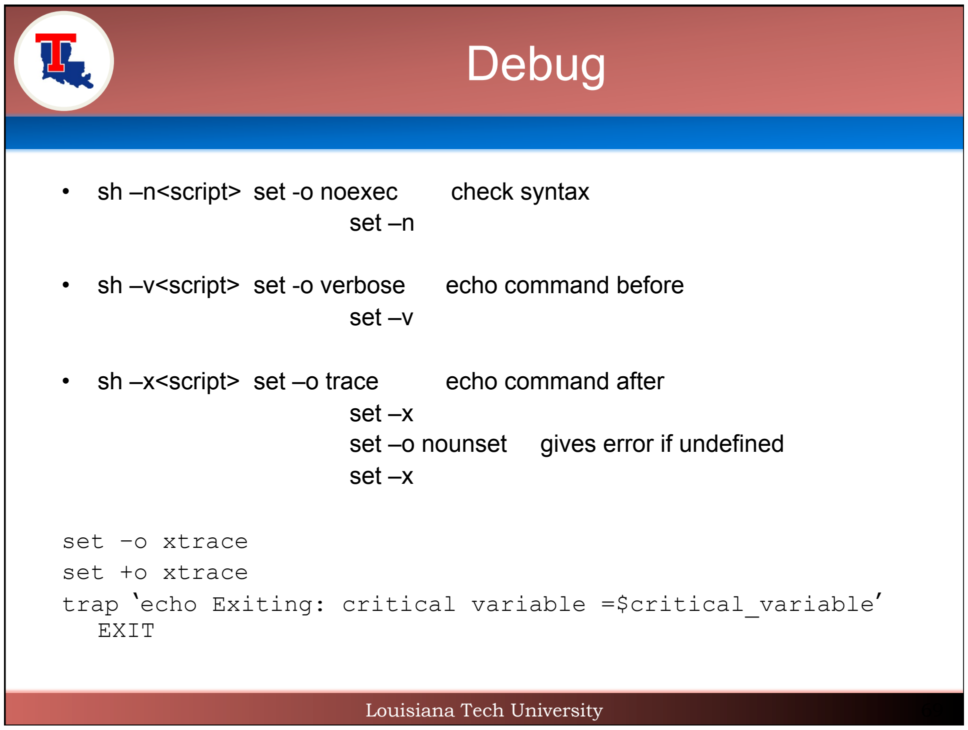
a_text_file

```
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
```

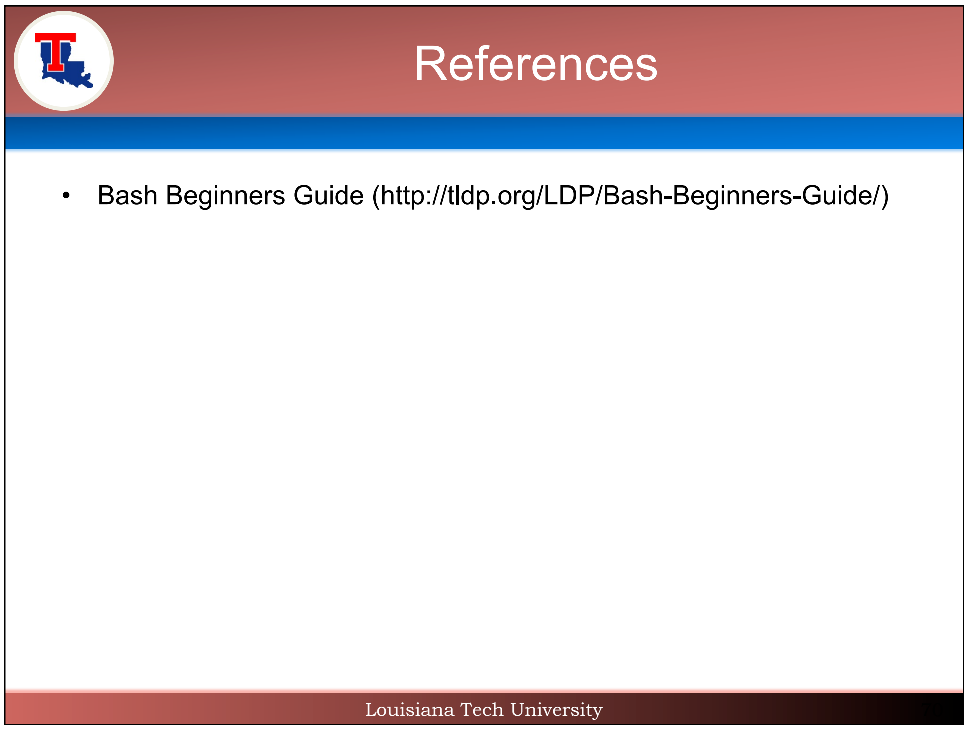
Louisiana Tech University



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/)