

Systems  
Software/Programming  
Shell Scripting

# Types of Shell

- Bourne Shell – with \$ as default prompt
  - Bourne Shell (sh)
  - Korn Shell (ksh)
  - Bourne Again Shell (bash)
  - POSIX Shell (sh)
- C-type Shell – with % as default prompt
  - C Shell (csh)
  - TENEX/TOPS C Shell (tcsh)

# Shell Scripts

- All shell scripts has extension .sh (e.g. test.sh)
- Use shebang (!) to indicate which shell you are using for shell script e.g. test.sh will contain

```
#!/bin/bash
```

```
pwd
```

```
ls
```

- Use # for single line comments

```
#!/bin/bash
```

```
# Author : Amit
```

```
# Copyright (c)
```

```
# Script follows here:
```

# Variables

- Shell Variables
  - Environment Variables
    - SHELL=/bin/bash
    - PATH=/usr/bin/
    - USER="logged in user"
    - HOME="path to home folder"
    - Many others...
  - Local Variables
    - Any variable that is defined in shell script by developer e.g. AUTHOR="Amit"
- Access variables using \$ sign e.g. echo \$PATH will display the value stored in environment variable PATH

# Special Variables

- \$\$ - process id (PID) of current shell
- \$0 - shell script filename being executed
- \$1, \$2, \$3...\$n – command line argument
- \$# - number of command line argument
- \$? – exit status of last command executed
  - 0 : success
  - 1 : error (all errors w/o specific code)
  - 130 terminated by Ctrl+C
- \$! – PID of last background command

- echo command is used to print value of any constant or variable

[ShellScripts\test.sh](#)

Now run: tesh.sh abc xyz

Output will be:

File name: test.sh

First command line argument is: abc

Second command line argument is: xyz

Execute C program and read the return status

[ShellScripts\retstat.sh](#)

# Arrays

## Definition

NAME[0]="Deepak"

NAME[1]="Renuka"

NAME[2]="Joe"

NAME[3]="Alex"

NAME[4]="Amir"

Or in bash

array\_name = (value1 ... valuen)

## Accessing

- To display value from a specific index

```
echo "First Index: ${NAME[0]}" echo  
"Second Index: ${NAME[1]}"
```

- To display all values

```
echo "All Index: ${NAME[*]}"
```

OR

```
echo "All Index: ${NAME[@]}"
```

[ShellScripts/array\\_test.sh](#)

# Operators

- Arithmetic Operators: + - \* / % = == !=
  - `c=`expr $a + $b`` add values from a and b and assign it to c
  - `a=$b` would assign value of b into a
  - `[ $a == $b ]` OR `[ $a != $b ]` would compare numeric values of a and b
- Relational Operators: -eq -ne -gt -lt -ge -le
- Boolean/Logical Operators: ! -o -a
- String Operators: -z -n
  - -z (or -n) returns true if string length is zero (or non-zero)

# Operators

- File Test Operators (assuming file variable holds the filename)
  - -d file: true if file is a directory
  - -f file: true if ordinary file instead of directory or special file
  - -r file: true if file is readable
  - -w file: true if file is writable
  - -x file: true if file is executable
  - -s file: true if file size > 0
  - -e file: true if file exists

Test if file exists

[ShellScripts/file\\_test.sh](#)

Test if file exist and display the content

[ShellScripts/file\\_read.sh](#)



# Decision Making Conditional Statements

- If...fi statement
- If...else...fi statement
- If...elif...else...fi statement

```
case word in
    pattern1)
        Statement(s) to be executed if
        pattern1 matches ;;
    pattern2)
        Statement(s) to be executed if
        pattern2 matches ;;
    pattern3)
        Statement(s) to be executed if
        pattern3 matches ;;
    *)
        Default condition to be
        executed ;;
esac
```

# Loops – Nested Loops are Allowed

- While Loop:

```
while command
```

```
do
```

```
    Statement(s) to be executed if command is  
true
```

```
done
```

- Until Loop:

```
until command
```

```
do
```

```
    Statement(s) to be executed until command is  
true
```

```
done
```

- For Loop:

```
for var in word1 word2 ... wordN
```

```
do
```

```
    Statement(s) to be executed for every word.
```

```
done
```

- Select Loop (Used for creating Menu):

```
select var in word1 word2 ... wordN
```

```
do
```

```
    Statement(s) to be executed for every word.
```

```
done
```

[ShellScripts\for select test.sh](#)

# Loop Control Statements

- break statement

```
#!/bin/sh a=0
while [ $a -lt 10 ]
do
    echo $a
    if [ $a -eq 5 ]
    then
        break
    fi
    a=`expr $a + 1`
done
```

- continue statement

```
#!/bin/sh
NUMS="1 2 3 4 5 6 7"
for NUM in $NUMS
do
    Q=`expr $NUM % 2`
    if [ $Q -eq 0 ]
    then
        echo "Number is an even
number!!"
        continue
    fi
    echo "Found odd number"
done
```

# Substitution

- Escape sequences with echo command
  - `\\` (backslash)
  - `\b` (backspace)
  - `\c` (suppress trailing newline)
  - `\f` (form feed)
  - `\n` (new line)
  - `\r` (carriage return)
  - `\t` (horizontal tab)
  - `\v` (vertical tab)
- For echo command, use `-e` (`-E`) option to enable (disable) interpretation of backslash escapes
- [ShellScripts\substitution.sh](#)

- Command Substitution using ``command``  
`DATE=`date``  
`echo "Date is $DATE"`  
`USERS=`who | wc -l``  
`echo "Logged in user are $USERS"`
- Variable Substitution  
`${var}`  
`${var:-defaultvalue}` → defaultvalue is NOT assigned to var  
`${var:=defaultval}` → defaultvalue is assigned to var  
`${var:?message}` → display a message on empty

# Input and Output Redirection

- Command/Program > file
  - Any output from command or program execution will be saved in file instead of displaying to STDOUT
  - New file will be created if does not exist or existing file will be erased first
- Command/Program >> file :
  - Any output from command or program execution will be appended to an existing file instead of displaying to STDOUT
  - New file will be created if does not exist but if file already exists then it is appended
- n >> file : output from stream with descriptor n is appended to a file
- n >& m : merges output from stream n with stream m  
[ShellScripts\merge stdout stderr.sh](#)
- Command/Program < file : Input to the command or program is fed from data in file
- | (called pipe) : Takes output from one process and feed into another process

# Functions

# Define your function here

```
Hello () {  
    if [ $# > 1 ]  
    then  
        for param in $*  
        do  
            echo $param  
        done  
    fi  
    return $#  
}
```

# Invoke your function

```
Hello $*
```

# Capture value returned by last command

```
ret=$?
```

[ShellScripts/function.sh](#)

**Note: Nested functions and recursive functions are allowed**

**Functions can be accessed in shell prompt by placing them in .sh file and executing that .sh file in a shell prompt**

# Exercises

- Print sum of all command line integer arguments
- Print the factorial of a given number using `fact()` function
- Print the day of the week for all the command line values provided between 1 and 7
- Given path (e.g. `/usr/local/lib` or `/etc/passwd`), first check if that path exists and then check if it is a file or a directory and print appropriate message

# Print sum of all command line arguments

- [ShellScripts\sum\\_args.sh](#)



# Quiz – Write a shell script

1. Multiple logs files with filenames \*.log are created while compilation of large application such as python from source. Task is to concatenate all the \*.log files and create a new file warnings.log with all the lines where warning has occurred (i.e. lines with 'warning' word)
2. User provided directory as command line argument has multiple program files with filenames \*.out. Task is to execute each program and store exit codes in an array.