CSE 362: Operating Systems Lab

Fun with Linux Shell Scripts

"Automate the boring stuff with shell scripts."

Introduction

- Building on basic shell commands, shell scripts allow you to automate tasks by combining commands into executable programs.
- Scripts can handle input, perform logic, and output results, making complex operations repeatable and efficient.
- We'll focus on Bash scripting, assuming `/bin/bash` is your default shell.
- Practice writing, executing, and debugging scripts to master automation.

What is Shell Script?

- We've seen basic shell commands; now it's time to move on to scripts.
- There are two ways of writing shell programs:
- You can type a sequence of commands and allow the shell to execute them interactively.
- You can store those commands in a file that you can then invoke as a program. This is known as a Shell Script.
- We will use Bash shell assuming that the shell has been installed as `/bin/sh` and that it is the default shell for your login.

More Examples

- Interactive: Type commands directly in terminal, e.g., `echo "Hello"; ls`.
- Script file: Save commands in a .sh file for reuse.

Tasks

1. Explain in your own words the difference between interactive commands and a shell script.
2. Think of a repetitive task you do in the terminal that could be scripted.

Answers: What is Shell Script Tasks

1. Interactive: Commands run one-by-one in real-time. Script: Stored in file, executed as a program.

2. Example: Backing up files daily – script to copy files to a backup directory.

Why Shell Script?

- Shell script can take input from user, file and output them on screen.
- Useful to create own commands.
- Save lots of time.
- To automate some task of day to day life.
- System administration part can be also automated.

More Examples

- Input/Output: Read user name and greet them.
- Custom Commands: Script to clean temporary files.
- Automation: Backup script run via cron.
- Admin: Script to monitor disk usage and alert.

Tasks

1. List two personal tasks you could automate with scripts.
2. Describe how a script could save time in system admin.

Answers: Why Shell Script Tasks

1. Examples: Automate file renaming; daily report generation.

2. Example: Script to update and upgrade packages automatically.

How to Write and Execute?

- Use any editor to write shell script.
- The extension is .sh.
- After writing shell script set execute permission for your script: `chmod +x script_name`
- Execute your script: `./script_name`

```
nano hello.sh
# Write script content
chmod +x hello.sh
./hello.sh
```

More Examples

```
vim myscript.sh
# Edit and save
chmod u+x myscript.sh
bash myscript.sh  # Alternative
  execution
```

Tasks

1. Create an empty script file named `test.sh` using nano.
2. Make it executable and run it (even if empty).
3. Try running without chmod – note the error.

Answers: How to Write and Execute Tasks

1. `nano test.sh` (then Ctrl+O, Enter, Ctrl+X to save)
2. `chmod +x test.sh`
   `./test.sh`
3. `./test.sh` → Permission denied

Shell Script Format

- Every script starts with the line: #/bin/bash!
- This indicates that the script should be run in the bash shell regardless of which interactive shell the user has chosen.
- This is very important, since the syntax of different shells can vary greatly.
- # is used as the comment character.
- A word beginning with # causes that word and all remaining characters on that line to be ignored.

```
#!/bin/bash
# This is a comment
echo "Hello"  # Inline comment
```

More Examples

```
#!/bin/bash
# Author: You
# Date: Today
# Purpose: Demo
```

```
echo "Start"
# echo "Commented out"
echo "End"
```

Tasks

1. Write a script with shebang and three comments.
2. Add inline comments to explain echo commands.
3. Run and verify comments are ignored.

Answers: Shell Script Format Tasks

1.
```
#!/bin/bash
# Comment 1
# Comment 2
# Comment 3
```

```
#!/bin/bash
echo "Hello"  # Prints
  greeting
```

3. `./script.sh` – No comment output

A Sample Shell Script

```
#!/bin/bash
echo "Hello User"
echo "See the files in current
  directory"
ls
```

More Examples

```
#!/bin/bash
echo "Current directory:"
pwd
echo "User:"
whoami
```

Tasks

1. Write and run the sample script.
2. Modify to include `pwd` and `date`.
3. Add comments explaining each line.

Answers: Sample Shell Script Tasks

1.
```
#!/bin/bash
echo "Hello User"
echo "See the files in current
   directory"
ls
```

```
chmod +x sample.sh
./sample.sh
```

2.
```
#!/bin/bash
echo "Hello User"
echo "Current path:"; pwd
echo "Date:"; date
echo "Files:"; ls
```

3. Add `# Greet user`, etc.

Variables

- In Linux (Shell), there are two types of variable:
- System variables - created and maintained by Linux itself: `echo $USER`, `echo $PATH`
- User defined variables - created and maintained by user.
- All variables are considered and stored as strings, even when they are assigned numeric values.
- Variables are case sensitive.

```
echo $USER
echo $PATH
myvar=hello
echo $myvar
```

More Examples

```
echo $HOME
num=42
echo $num  # Outputs 42 as string
```

```
echo $((num + 1))  # Arithmetic
  expansion
```

Tasks

1. Print $USER and $PATH in a script.
2. Create user var `os=Linux`, echo it.
3. Assign numeric value, echo as is and +1.
4. Test case sensitivity: `Var=1`, `var=2`.

Answers: Variables Tasks

1.

```
#!/bin/bash
echo $USER
echo $PATH
```

```
os=Linux
echo $os
```

2. 3.

```
num=10
echo $num
echo $((num + 1))
```

```
Var=1
var=2
echo $Var $var  # 1 2
```

Variables (Continued)

4. When assigning: no spaces around = → `var_name=value`

- Use quotes for spaces: `name="John Doe"`
- Double quotes allow substitution, single quotes prevent it.

```
myvar="Hello"
echo $myvar       # Hello
echo "$myvar"     # Hello
echo '$myvar'     # $myvar
echo \$myvar      # $myvar
```

Tasks

1. Assign a variable with spaces using quotes.
2. Demonstrate double vs single quotes.
3. Escape $ to print literally.

Answers: Variables (Continued) Tasks

1. ```
   phrase="Hello World"
   echo $phrase
   ```

2. `echo "$phrase"` → Hello World
   `echo '$phrase'` → $phrase

3. echo \$phrase → $phrase
   ```

Read User Input

- Use `read var1 var2 ...` to read from keyboard.

```
#!/bin/bash
echo -n "Enter your name: "
read name
echo -n "Enter your student no: "
read stdno
echo "Your Name: $name"
echo "Your Age: $stdno"
```

More Examples

```
read -p "Age: " age
echo "Age: $age"
read first last
echo "Full: $first $last"
```

Tasks

1. Write script to read name and greet.
2. Read two numbers and echo their sum using `expr`.
3. Use `read -p` for prompt.

Answers: Read User Input Tasks

1.
```
#!/bin/bash
echo -n "Name: "
read name
echo "Hi $name!"
```

```
echo "Enter two numbers:"
read a b
echo $(expr $a + $b)
```

2.
3. `read -p "Input: " var`

Shell Arithmetic

- Use `expr` for basic operations (escape `*` with \).

```
expr 10 + 5
expr 6 \* 4
var=‘expr 3 + 7‘
```

Operator	Meaning
|	logical OR
&	logical AND
=	equal
>	greater than
>=	greater or equal
<	less than
<=	less or equal
!=	not equal
+	addition
-	subtraction
	multiplication
/	division
%	modulo

Tasks

1. Compute 15 % 4 using expr.

2. Increment a variable by 10.

3. Check if $8 > 5$.

Answers: Shell Arithmetic Tasks

1. `expr 15 % 4` → 3
2.
```
x=5
x=‘expr $x + 10‘
```
3. `expr 8 > 5` → 1 (true)

If-Else

```
if [ condition ]; then
    statements
elif [ condition2 ]; then
    statements
else
    statements
fi
```

- Spaces around condition inside [ ] are mandatory.
- Same-line: `if [ cond ]; then echo yes ; fi`

String comparisons

Test	Meaning
str1 = str2	equal
str1 != str2	not equal
-n str	not empty
-z str	empty

Arithmetic comparisons

Test	Meaning
a -eq b	equal
a -ne b	not equal
a -gt b	greater
a -ge b	greater or equal
a -lt b	less
a -le b	less or equal

File tests

Test	Meaning
-d file	directory
-e file	exists
-f file	regular file
-r file	readable
-w file	writable
-x file	executable

Tasks

1. Write script to compare two numbers read from user.
2. Check if a file exists.
3. Create age classifier (child/teen/adult).

Answers: If-Else Tasks

1.
```
read a b
if [ $a -gt $b ]; then echo "
  $a > $b"
elif [ $a -lt $b ]; then echo
  "$a < $b"
else echo equal; fi
```

2. `if [ -e file.txt ]; then echo Exists ; fi`

3.
```
read age
if [ $age -lt 13 ]; then echo
  Child
elif [ $age -lt 20 ]; then
  echo Teen
else echo Adult; fi
```

Case Statement

```
case $var in
  pat1) commands ;;
  pat2|pat3) commands ;;
  *) default ;;
esac
```

Example

```
echo "Is it morning? (yes/no)"
read ans
case "$ans" in
  yes|y|Yes|YES) echo "Good
  Morning" ;;
  no|n*) echo "Good Afternoon" ;;
  *) echo "Not recognized" ;;
esac
```

Tasks

1. Create menu: 1→List files, 2→Show date, else→Invalid.
2. Handle file extensions (.txt, .sh, etc.).

Answers: Case Tasks

1.
```
read choice
case $choice in
  1) ls ;;
  2) date ;;
  *) echo Invalid ;;
esac
```

```
case "$file" in
  *.txt) echo Text ;;
  *.sh) echo Script ;;
  *) echo Other ;;
esac
```

Final Challenge: Simple Backup Script

Create `backup.sh` that:

1. Takes a directory name as argument.
2. Checks if argument is provided.
3. Creates a dated backup folder.
4. Copies all files from given directory.
5. Logs actions.

Answer: Final Challenge

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Usage: $0 <directory>"
    exit 1
fi

src=$1
backup="backup_$(date +%Y%m%d_%H%M
  %S)"
mkdir -p "$backup"

for file in "$src"/*; do
    if [ -f "$file" ]; then
        cp "$file" "$backup/"
        echo "$(date): Copied
  $file" >> backup.log
    fi
done
```

```
echo "Backup completed in $backup"
```

Command Line arguments

- Command line arguments can be passed to the shell scripts.
- There exists a number of built in variables
- `$*` - command line arguments
- `$#` - number of arguments
- `$n` - nth argument in $*
- `./script_name arg1 arg2 .... argn`

```
echo "Total args: $#"
echo "First arg: $1"
echo "All args: $*"
```

More Examples

```
#!/bin/bash
echo "the number of args is $#"
a=1
```

```
for i in $*
do
  echo "The $a No arg is $i"
  a=‘expr $a + 1‘
done
```

Tasks

1. Print number of arguments.
2. Print first and second argument.
3. Loop through all arguments and number them.

Answers: Command Line arguments Tasks

1. `echo $#`
2. `echo $1 $2`
3. As in more examples.

For

```
for variable in list
do
  statement
done

for (( expr1; expr2; expr3 ))
do
  statement
done
```

```
#!/bin/bash
for i in `ls`
do
  echo $i
done

for((i=0;i<=50;i++))
do
  echo $i
```

```
done
```

Tasks

1. Loop over `ls` output.
2. Print numbers 1 to 20 using C-style for.
3. Loop over command line arguments.

Answers: For Tasks

1. As first example.
2. `for ((i=1;i<=20;i++)); do echo $i; done`
3. `for arg in $*; do echo $arg; done`

While

```
while condition
do
  statements
done
```

```
#!/bin/bash
password="abc"
echo "Enter password"
read pass
while [ $pass != $password ]
do
  echo "Wrong Password, Try again"
  read pass
done
echo "Right Password"
```

Tasks

1. Write password checker with while.

2. Count from 1 to 10 using while.

Answers: While Tasks

1. As example.
2.

```
i=1
while [ $i -le 10 ]; do
  echo $i
  i=‘expr $i + 1‘
done
```

Until

```
until condition
do
  statements
done
```

```
#!/bin/bash
password="abc"
echo "Enter password"
read pass
until [ $pass = $password ]
do
  echo "Wrong Password, Try again"
  read pass
done
echo "Right Password"
```

Tasks

1. Rewrite password checker using until.

2. Countdown from 10 to 1.

Answers: Until Tasks

1. As example.
2.

```
i=10
until [ $i -lt 1 ]; do
  echo $i
  i=‘expr $i - 1‘
done
```

Functions

- Functions can be defined in the shell and it is very useful to structure the code.
- To define: `function_name()  statements`
- Must be defined before calling.

```
#!/bin/sh

foo() {
  echo "Function foo is executing"
}

echo "script starting"
foo
echo "script ending"
```

More Examples

```
greet() {
```

```
  echo "Hello $1"
}
greet Alice
```

Tasks

1. Define and call simple function.
2. Pass argument to function.

Answers: Functions Tasks

1.
```
hi() { echo Hello; }
hi
```

2. As more examples.

Functions

```
#!/bin/bash

showarg()
{
  a=1
  for i in $*
  do
    echo "The $a No arg is $i"
    a=‘expr $a + 1‘
  done
}

showarg $*
```

- Return numeric: `return n`
- Return string: echo or global var

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

```
{
  echo "Is your name $* ?"
  read x
  case "$x" in
    y|yes) return 0;;
    *) return 1;;
  esac
}

if yes_or_no "$1"
then
  echo "Hi $1, nice name"
else
  echo "Never mind"
fi
```

Tasks

1. Create function to add two numbers.
2. Use yes_or_no function.

Answers: Functions Tasks

1.
```
add() { echo `expr $1 + $2`; }
add 5 7
```

2. Run script with name as argument.

Final Challenge: Simple Backup Script

Create `backup.sh` that:

1. Takes one directory as argument.
2. Checks if argument is given.
3. Creates dated backup folder.
4. Copies all files from source.
5. Logs actions.

```
chmod +x backup.sh
./backup.sh myfolder
```

Answer: Final Challenge — backup.sh

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo "Usage: $0 <directory>"
    exit 1
fi

src="$1"
backup="backup_$(date +%Y%m%d_%H%M
  %S)"
mkdir -p "$backup"

for file in "$src"/*; do
    if [ -f "$file" ]; then
        cp "$file" "$backup/"
        echo "$(date): Copied
  $file" >> backup.log
    fi
done
```

```
echo "Backup completed: $backup"
echo "Log saved in backup.log"
```

Thank you

Now go open your terminal and start scripting!