CSE4510 Operating Systems Lab

Introduction to Shell Scripts

Salman Shamil

United International University (UIU)
Summer 2025

Lecture Topics

- **Recap Lab-02**
  - `find` and `grep`
  - Practice problem
- **Schell Scripting**
  - Writing and executing sctips
  - Solving basic programming problems
  - Practice problem

[Recap] The find tool

- Finding files or directories matching some criteria
- Recursively searches for files within the given directory

```
# By name (case-sensitive / insensitive), or path
find . -name '*.txt'
find . -iname '*.Txt'
find . -path '*/archive/*.txt'

# By type (file, directory, symlink...)
find . -type f
find . -type d

# By owner or group
find . -user alice
find . -group staff
```

- You can also filter by file size and/or modification time.
- **TODO**: Use `tldr` or `man` to see how to do it.

[Recap] More about find

- `find` is a really powerful tool. It can do a lot more.

```
# Depth control
find . -maxdepth 2 -type f

# Executing actions
find . -name '*.log' -exec rm {} \;
```

- You can combine multiple criteria.
- List all `.txt` files in your home directory that were modified in the last 7 days and are larger than 1KB. For each such file, show the number of lines in it.

```
find ~/ -name '*.txt' \
    -mtime -7 \
    -size +1k \
    -exec wc -l {} \;
```

- **TODO**: Use `tldr` for more examples.

[Recap] The grep tool

- Searching based on file conent.

```
# Look for the text "print" in cpu.c
grep print cpu.c
# Case-insensitie (-i), line number (-n)
grep -in linux README.md
# Match whole word
grep -n -w printf cpu.c
# Show count only (-c)
grep -c print cpu.c
# Invert match (-v)
grep -v include common.h
# Using regular expressions
grep -inw 'p[a-z]*t' README.md
# Match either pattern, using Extended Regex
grep -E 'double|int' common.h
```

Here you can find a useful and compact grep cheatsheet.

[Recap] More about grep

```
# Recursively search files in a directory
grep -rn "common\.h" .
# Combine with find for finer control
find . -name "*.c" -exec grep -nH "common\.h" {} \;
# grep with piping and context (after)
man grep | grep -n -A5 "filename"
# Show context around matches
# 2 lines before, 4 lines after
grep -n -B2 -A4 "main" cpu.c
# 3 lines before & after
grep -n -C3 "main" cpu.c
```

Practice

Task: Secure all C source files that include “common.h”

- Identify all C source files that include the header `common.h`.
  - Use `find` and `grep`
- Remove group-write permission from each of those files.
  - `-exec` can be used multiple times consecutively.
- Combine the search and permission change into a single operation.

Verify that the files with common.h include no longer grant write access to the group.

Shell Scripting: Variables and Functions

Scripts are useful for executing a **sequence of commands** and implementing **control flow** (*e.g.*, conditionals and loops).

Shell Scripting: Variables and Functions

Scripts are useful for executing a **sequence of commands** and implementing **control flow** (*e.g.*, conditionals and loops).

- **Variables**
  - assigning variables: *e.g.*, `a=3`
    - NOT `a = 3`, becomes argument of `a`!
    - `<space>` for argument splitting.
  - strings: within `'_'` vs `"_"`
    - literal (no expansion) *vs* variable expansion
    - `echo '$a'` *vs* `echo "$a"`

Shell Scripting: Variables and Functions

Scripts are useful for executing a **sequence of commands** and implementing **control flow** (*e.g.*, conditionals and loops).

- **Variables**
  - assigning variables: *e.g.*, `a=3`
    - NOT `a = 3`, becomes argument of `a`!
    - `<space>` for argument splitting.
  - strings: within `'_'` vs `"_"`
    - literal (no expansion) *vs* variable expansion
    - `echo '$a'` *vs* `echo "$a"`
- **Functions**
  - Function / script arguments
  - `$0` – name of the function/script
  - `$1` to `$9` – positional arguments
  - `$@` – all arguments as separate words
  - `$#` – number of arguments

```
touch_perm () {
    touch "$1"
    chmod "$2" "$1"
}

touch_perm file1 764
```

Shell Scripting: Conditional Statement with `if`

- String comparison

```
name="$1"
if [[ $name == "admin" ]]; then
    echo "Welcome!"
fi
```

- Numeric comparison

```
age="$2"
if [[ $age -ge 18 ]]; then
    echo "Adult"
fi
```

- Combine with OR

```
if [[ $user == "admin" && $age -lt 18 ]]; then
    echo "Access granted: privileged young user!"
fi
```

- Note: Arithmetic operatiosn can be done with (( ))

```
result=$(( 5 + 3 ))        # addition: result=8
result=$(( 4 * 2 ))        # multiplication: result=8
result=$(( 5 % 3 ))        # modulus: result=2
```

Shell Scripting: Conditional Statement with `if` (cont.)

- Common Bash Comparison and Logical Operators

String	**Numeric**	**Logical**
`==`, `!=`	`-eq`, `-ne`	`&&`
`-z`, `-n`	`-gt`, `-lt`	`||`
	`-ge`, `-le`	`!`

Shell Scripting: Conditional Statement with if (cont.)

- Common Bash Comparison and Logical Operators

String	Numeric	Logical
==, !=	-eq, -ne	&&
-z, -n	-gt, -lt	||
	-ge, -le	!

- if-else if . . .

```
#!/bin/bash

read -p "Enter your username: " username

if [[ -z "$username" ]]; then
    echo "Error: Username cannot be empty."
elif [[ "$username" == "admin" ]]; then
    echo "Welcome, admin! You have full access."
else
    echo "Hello, $username! Limited Access Granted."
fi
```

. . . **Notice how we took input from the user!**

Shell Scripting: Looping with `for`

- Iterate over a list of values

```
for name in Alice Bob Carol; do
    echo "Hello, $name"
done
```

- Iterate over files in a directory

```
for file in *.txt; do
    echo "Found text file: $file"
done
```

- Use a C-style loop (numeric range)

```
for (( i=1; i<=10; i+=2 )); do
    echo "Number $i"
done
```

- Note: `$(( ))` *vs* `$( )`
  - `$(( ))` evaluates expression; returns numeric value
  - `$( )` executes command; returns standard output

Offline Assignment-01: Task A

TASK A: File Length Checker

Write a bash script named `file_length` that interacts with the user to check the number of lines in multiple files and categorizes each file as **empty**, **small**, or **large**.

Subtasks:

1. **Query Count:** Prompt the user to enter the total number of files to check.
2. **Iterations:** Use a for loop to iterate exactly `num_queries` times.
3. **Remaining Queries:** At the start of each iteration, display how many queries remain.
4. **Filename Input:** Prompt for a filename on each iteration:
5. **Line Count:** Count the number of lines in the specified file.
    - You may assume that the file exists. Use `wc -l`.
    - Extracting the number only can be tricky. But there are multiple ways.
6. **Output:** Print the line count and based on that print its category.
    - **Empty:** `num_lines == 0`
    - **Small:** `1 <= num_lines < 10`
    - **Large:** `num_lines >= 10`

Offline Assignment-01: Task A (cont.)

Deliverable: A single executable script named `file_length` that reproduces the sample output shown below. Lines beginning with > asks for user input (prompts), while the remaining lines are program output.

```
$ ./file_length
> Number of queries: 3
You have 3 queries remaining
> Filename: dummy.sh
Number of lines: 0
dummy.sh is empty!
You have 2 queries remaining
> Filename: if_checker
Number of lines: 23
if_checker is large!
You have 1 queries remaining
> Filename: loop_checker
Number of lines: 5
loop_checker is small!
```

Offline Assignment-01: Task B

TASK B: Shell-Script Audit

Write a program `audit_scripts` that discovers shell scripts and counts occurrences of a user-supplied keyword.

Subtasks:

1. **Directory Prompt:** Prompt the user for a directory path.
2. **Find Scripts:** Use `find` to locate **all** files ending in `.sh` under the given directory. Store them into an array. Here's a sample code snippet for creating such a list and accessing the first item.

```
files=($(find . -type f))
echo ${files[0]}
```

3. **Keyword Prompt:** Prompt the user for a keyword that will be searched for.
4. **Count Matches:** For each `.sh` file returned by `find`, run a case-insensitive search to count the number of matching lines using `grep`.
5. **Report Results:** Print one line per script file in the format:

```
./path/to/script.sh :  X occurrences
```

Offline Assignment-01: Task B (cont.)

Deliverable: A single executable script named `audit_scripts` that reproduces the sample output shown below. Lines beginning with > asks for user input (prompts), while the remaining lines are program output.

```
$ ./audit_scripts
> Enter a directory to scan for shell scripts: ./projects
> Enter a keyword to count in found scripts: TODO
./projects/install.sh : 3 occurrences
./projects/tests/run_tests.sh : 0 occurrences
./projects/utils/helpers.sh : 1 occurrences
```

Submission Instructions: Create two bash scripts named `file_length` and `audit_scripts`, and place them both in a single directory named with your student ID. Compress this directory into a ZIP archive and upload it to eLMS.