

Shell Scripting Lab Report

Lab #27 - Automating Tasks with Bash



Ahmad Mukhtar

**National University of Sciences and Technology (NUST)
Chip Design Centre (NCDC), Islamabad, Pakistan**

October 13, 2024

1 Task 1: Sorting an Array Using Bubble Sort

1.1 Algorithm

Bubble Sort Algorithm

Bubble Sort is a simple comparison-based sorting algorithm where each pair of adjacent elements is compared, and the elements are swapped if they are in the wrong order. This process is repeated for each element until the entire array is sorted.

1.1.1 Steps

- **Step 1:** Begin with an unsorted array.
- **Step 2:** Outer Loop: Repeat the sorting process for all elements except the last one in each pass.
- **Step 3:** Inner Loop: Compare adjacent elements in the array.
- **Step 4:** Swap: If the current element is greater than the next, swap them.
- **Step 5:** Repeat: The process continues until the array is fully sorted.
- **Step 6:** End: After all passes, the array will be sorted.

1.2 Bash Script and Syntax

Array Declaration: Declares the array to be sorted.

```
arr=(64 34 25 12 22 11 90)
```

Listing 1.1: Array Declaration

Get Array Length: Retrieves the number of elements in the array.

```
n=${#arr[@]}
```

Listing 1.2: Get Array Length

Outer Loop: Loops through the array for sorting, reducing the unsorted section on each pass.

```
for ((i = 0; i < n-1; i++))
```

Listing 1.3: Outer Loop

Inner Loop: Loops through adjacent elements within the unsorted section.

```
for ((j = 0; j < n-i-1; j++))
```

Listing 1.4: Inner Loop

Comparison and Swap: Compares two adjacent elements and swaps them if needed.

```
if [ ${arr[j]} -gt ${arr[$((j+1))]} ]; then
    temp=${arr[j]}
    arr[j]=${arr[$((j+1))]}
    arr[$((j+1))]=$temp
fi
```

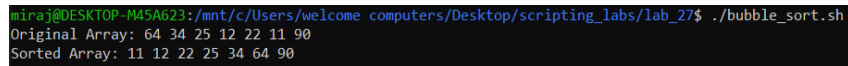
Listing 1.5: Comparison and Swap

Output the Sorted Array: After sorting, the final sorted array is printed.

```
echo "Sorted Array: ${arr[@]}"
```

Listing 1.6: Output the Sorted Array

1.3 Output



```
miraj@DESKTOP-M45A623: /mnt/c/Users/welcome_computers/Desktop/scripting_labs/lab_27$ ./bubble_sort.sh
Original Array: 64 34 25 12 22 11 90
Sorted Array: 11 12 22 25 34 64 90
```

Figure 1.1: Bubble Sort Output

2 Task 2: Random Password Generator

2.1 Algorithm

Random Password Generator Algorithm

This script generates a random password based on user input. It creates a password containing alphanumeric characters and special symbols.

2.1.1 Steps

- **Step 1:** Ask the user to specify the desired length of the password.
- **Step 2:** Validate the input to ensure it is a valid number (positive integer).
- **Step 3:** Generate a random password using alphanumeric characters and special symbols.
- **Step 4:** Display the generated password for the user.

2.2 Bash Script and Syntax

Prompt for Password Length: This command prompts the user to enter the password length.

```
read -p "Enter the desired password length: " password_length
```

Listing 2.1: Prompt for Password Length

Input Validation: A regular expression check to ensure the input is a valid number. If not, an error message is displayed.

```
if ! [[ "$password_length" =~ ^[0-9]+$ ]]; then
    echo "Error: Please enter a valid number."
    exit 1
fi
```

Listing 2.2: Input Validation

Random Password Generation: The `/dev/urandom` system file is used to generate random characters. Only alphanumeric characters and special symbols are filtered using `tr`, and the password is trimmed to the user-specified length.

```
password=$(< /dev/urandom tr -dc 'A-Za-z0-9@#$$%^&*()_+= ' | head  
-c "$password_length")
```

Listing 2.3: Random Password Generation

Output the Generated Password: The generated password is displayed using the ‘echo’ command.

```
echo "Generated Password: $password"
```

Listing 2.4: Output the Generated Password

2.3 Output

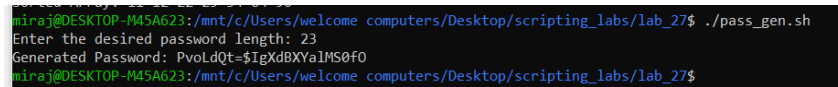
A terminal window screenshot showing the execution of a script. The prompt is 'miraj@DESKTOP-M45A623:/mnt/c/Users/welcome_computers/Desktop/scripting_labs/lab_27\$'. The user enters './pass_gen.sh'. The script prompts 'Enter the desired password length: 23'. The user enters '23'. The script outputs 'Generated Password: PvoLdQt=\$IgXdBXyAlMS0f0'. The prompt returns to 'miraj@DESKTOP-M45A623:/mnt/c/Users/welcome_computers/Desktop/scripting_labs/lab_27\$'.

Figure 2.1: Password Generator Output

3 Task 3: Git Init and Status Simulation

3.1 Algorithm

Git Simulation Algorithm

This script simulates basic Git commands such as ‘init’ and ‘status’, allowing users to back up and track changes to a directory.

3.1.1 Steps

- **Step 1:** Check if the directory has already been initialized.
- **Step 2:** If not initialized, create a backup of the directory, excluding the backup folder itself.
- **Step 3:** If initialized, update the backup with the latest files.
- **Step 4:** Compare the current directory with the backup to show differences such as untracked or removed files, and modified files and their differences.

3.2 Bash Script and Syntax

Backup Directory Declaration: This is the directory that simulates the ‘.git’ directory in Git.

```
backup_dir=".backup"
```

Listing 3.1: Backup Directory Declaration

Init Function: If the backup directory exists, it updates the backup using ‘rsync’ to synchronize files. If the backup directory doesn’t exist, it creates one and backs up the files.

```
init() {  
    if [ -d "$backup_dir" ]; then  
        echo "The directory has already been initialized.  
        Updating the backup..."  
        rsync -a --delete --exclude="$backup_dir" ./  
            "$backup_dir/"  
        echo "Backup updated."  
    fi  
}
```

```

else
    mkdir "$backup_dir"
    rsync -a --exclude="$backup_dir" ./ "$backup_dir/"
    echo "Initialized the directory and created a backup."
fi
}

```

Listing 3.2: Init Function

Status Function: If the directory is not initialized, it informs the user to run ‘init’ first. It compares the current directory with the backup using ‘diff’ and identifies untracked, removed, or modified files.

```

status() {
    if [ ! -d "$backup_dir" ]; then
        echo "The directory is not initialized. Please run
            'init' first."
        return
    fi

    echo "Checking the status of files..."
    diff -rq . "$backup_dir" --exclude="$backup_dir" | while
    read -r line; do
        if [[ $line == *"Only in"* ]]; then
            file=$(echo "$line" | sed 's/Only in //' | sed 's:/
                /\///')
            if [[ $file != "$backup_dir"* ]]; then
                echo "Untracked or removed file: $file"
            fi
        elif [[ $line == *"differ"* ]]; then
            file=$(echo "$line" | sed 's/Files //' | sed 's/
                and.* differ//')
            echo "Modified file: $file"
            diff "$file" "$backup_dir/$file"
        fi
    done
}

```

Listing 3.3: Status Function

Main Program Logic: Depending on the user’s input, either the ‘init’ or ‘status’ function is executed.

```

if [ "$1" == "init" ]; then
    init
elif [ "$1" == "status" ]; then
    status
else
    echo "Usage: $0 {init|status}"
fi

```

Listing 3.4: Main Program Logic

3.3 Output

```
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$ ./git_sim.sh status
The directory is not initialized. Please run 'init' first.
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$ ./git_sim.sh init
Initialized the directory and created a backup.
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$ ./git_sim.sh status
Checking the status of files...
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$ ./git_sim.sh init
The directory has already been initialized. Updating the backup...
Backup updated.
miraj@DESKTOP-M45A623:/mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$
```

Figure 3.1: Git Init and Status Simulation Output

4 Task 4: File Type Segregation and Archiving

4.1 Algorithm

File Segregation Algorithm

The script organizes files into different directories based on their extensions and compresses them into zip archives.

4.1.1 Steps

- **Step 1:** Define the base directory where files are located.
- **Step 2:** Create subdirectories for '.pdf', '.c', and '.h' files.
- **Step 3:** Move files to their respective subdirectories based on their extension.
- **Step 4:** Zip each subdirectory into a compressed file.

4.2 Bash Script and Syntax

Define the Base Directory: Set up the directory where files are located.

```
base_dir="files"
```

Listing 4.1: Define the Base Directory

Create Subdirectories for Each File Type: Using 'mkdir -p' ensures that the subdirectories are created if they don't exist.

```
pdf_dir="$base_dir/pdf_files"  
c_dir="$base_dir/c_files"  
h_dir="$base_dir/h_files"  
  
mkdir -p "$pdf_dir"  
mkdir -p "$c_dir"  
mkdir -p "$h_dir"
```

Listing 4.2: Create Subdirectories for Each File Type

Move Files Based on Extension: Loop through each file in the base directory and move files based on their extension using a case statement.

```

for file in "$base_dir"/*; do
    if [[ -f "$file" ]]; then
        case "$file" in
            *.pdf)
                mv "$file" "$pdf_dir/"
                ;;
            *.c)
                mv "$file" "$c_dir/"
                ;;
            *.h)
                mv "$file" "$h_dir/"
                ;;
        esac
    fi
done

```

Listing 4.3: Move Files Based on Extension

Zip Subdirectories: After segregating the files, compress each subdirectory into a '.zip' archive and place it in the base directory.

```

zip -r "$base_dir/pdf_files.zip" "$pdf_dir" > /dev/null 2>&1
zip -r "$base_dir/c_files.zip" "$c_dir" > /dev/null 2>&1
zip -r "$base_dir/h_files.zip" "$h_dir" > /dev/null 2>&1

```

Listing 4.4: Zip Subdirectories

Completion Message: After successful segregation and archiving, a confirmation message is displayed.

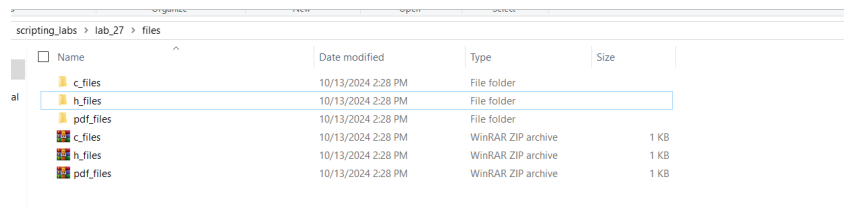
```
echo "Files segregated and zipped successfully."
```

Listing 4.5: Completion Message

4.3 Before and After Segregation

ng_labs > lab_27 > files				
Name	Date modified	Type	Size	
ahmad	10/12/2024 1:37 PM	C Source File	1 KB	
ahmad	10/12/2024 1:42 PM	H File	1 KB	
ahmad	10/12/2024 1:37 PM	Microsoft Edge PDF Doc...	1 KB	
ahmad_two	10/12/2024 1:38 PM	Microsoft Edge PDF Doc...	1 KB	
ali	10/12/2024 1:38 PM	C Source File	1 KB	
ali	10/12/2024 1:43 PM	H File	1 KB	

Figure 4.1: Before Segregation



<input type="checkbox"/>	Name	Date modified	Type	Size
<input type="checkbox"/>	c_files	10/13/2024 2:28 PM	File folder	
<input type="checkbox"/>	h_files	10/13/2024 2:28 PM	File folder	
<input type="checkbox"/>	pdf_files	10/13/2024 2:28 PM	File folder	
<input checked="" type="checkbox"/>	c_files	10/13/2024 2:28 PM	WinRAR ZIP archive	1 KB
<input checked="" type="checkbox"/>	h_files	10/13/2024 2:28 PM	WinRAR ZIP archive	1 KB
<input checked="" type="checkbox"/>	pdf_files	10/13/2024 2:28 PM	WinRAR ZIP archive	1 KB

Figure 4.2: After Segregation

5 Task 5: Executable Finder and Logger

5.1 Algorithm

Executable Finder Algorithm

This script finds all executable files in subdirectories and logs the results in ‘executables.log’.

5.1.1 Steps

- **Step 1:** A log file named ‘executables.log’ is created or cleared at the start.
- **Step 2:** Search all subdirectories within the current directory.
- **Step 3:** Search for executable files in each subdirectory.
- **Step 4:** Log and display the found executables.
- **Step 5:** Display a completion message when all executables have been logged.

5.2 Bash Script and Syntax

Log File Creation: The log file is either created or cleared at the start of the script using the following syntax:

```
log_file="executables.log"  
> "$log_file"
```

Listing 5.1: Log File Creation

Find Subdirectories: The script uses ‘find’ to search for subdirectories at a minimum depth of 2, ensuring that it searches within the current directory’s subfolders.

```
find . -mindepth 2 -type d | while read -r dir; do
```

Listing 5.2: Find Subdirectories

Find Executable Files: For each subdirectory found, the script searches for files with the executable bit set. It only checks the first level of files (‘-maxdepth 1’) within each subdirectory.

```
executables=$(find "$dir" -maxdepth 1 -type f -executable)
```

Listing 5.3: Find Executable Files

Print and Log Executable Files: If executables are found, they are both printed to the console and appended to the log file.

```
if [ -n "$executables" ]; then
    echo "Directory: $dir"
    echo "Directory: $dir" >> "$log_file"

    for exe in $executables; do
        echo "    $exe"
        echo "    $exe" >> "$log_file"
    done
fi
```

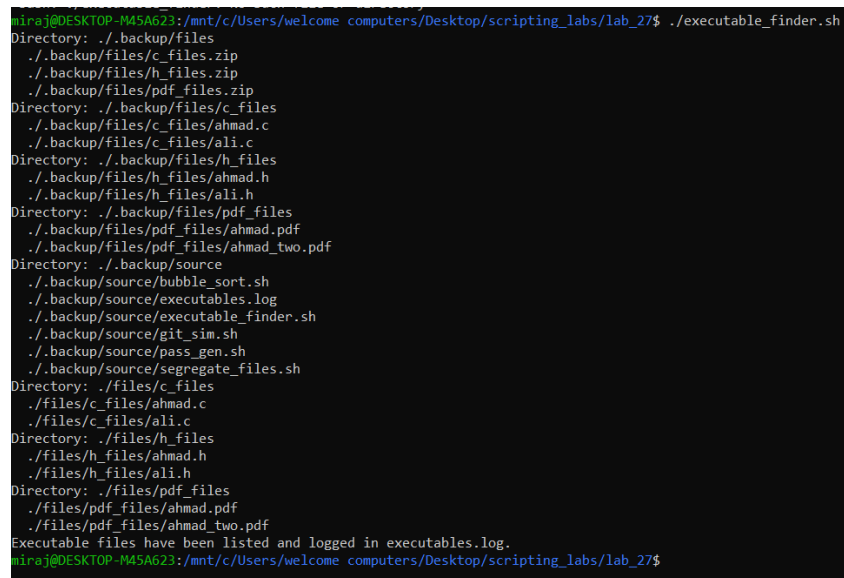
Listing 5.4: Print and Log Executable Files

Completion Message: The script ends with a message confirming that all executable files have been logged.

```
echo "Executable files have been listed and logged in $log_file."
```

Listing 5.5: Completion Message

5.3 Output



```
miraj@DESKTOP-M45A623: /mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$ ./executable_finder.sh
Directory: ./backup/files
./backup/files/c_files.zip
./backup/files/h_files.zip
./backup/files/pdf_files.zip
Directory: ./backup/files/c_files
./backup/files/c_files/ahmad.c
./backup/files/c_files/ali.c
Directory: ./backup/files/h_files
./backup/files/h_files/ahmad.h
./backup/files/h_files/ali.h
Directory: ./backup/files/pdf_files
./backup/files/pdf_files/ahmad.pdf
./backup/files/pdf_files/ahmad_two.pdf
Directory: ./backup/source
./backup/source/bubble_sort.sh
./backup/source/executables.log
./backup/source/executable_finder.sh
./backup/source/git_sim.sh
./backup/source/pass_gen.sh
./backup/source/segregate_files.sh
Directory: ./files/c_files
./files/c_files/ahmad.c
./files/c_files/ali.c
Directory: ./files/h_files
./files/h_files/ahmad.h
./files/h_files/ali.h
Directory: ./files/pdf_files
./files/pdf_files/ahmad.pdf
./files/pdf_files/ahmad_two.pdf
Executable files have been listed and logged in executables.log.
miraj@DESKTOP-M45A623: /mnt/c/Users/welcome computers/Desktop/scripting_labs/lab_27$
```

Figure 5.1: Executable Finder Output