Task 1: File Management Script

write bash script that:

1. create backup directory named 'backup' in user home directory

2. copy all .txt files from current directory into 'backup' directory.

3. append current date and time to filenames of copied files

```
#!/bin/bash

backup_dir="backup"
mkdir  "$backup_dir"

echo "backup directory created at: $backup_dir"

for file in .txt; do
if [[ -f "$file" ]]; then


timestamp=$(date + "%Y%m%d_%H%M%S")
base_name=$(basename "$file".txt)
new_name="${base_name}_${timestamp}.txt"

cp "$file" "$backup_dir/$new_name"
echo "copied $file to $backup_dir/$new_name"
fi

done
```

Task 2: System health check

Create a script that

1. check the system's CPU and memory usage

2. reports if CPU usage is above 80% or if available memory is below 20%

3. logs the result to a file named system_health.log

```bash
#!/bin/bash

log_file="system_health.log"
timestamp=$(date "+%Y-%m-%d %H:%M:%S")

# --- Get CPU usage on macOS ---
cpu_usage=$(ps -A -o %cpu | awk '{s+=$1} END {print s}')
cpu_usage_int=${cpu_usage%.*}

# --- Get memory info on macOS ---
# vm_stat returns memory pages; each page is 4096 bytes
pages_free=$(vm_stat | grep "Pages free" | awk '{print $3}' | sed 's/\.//')
pages_inactive=$(vm_stat | grep "Pages inactive" | awk '{print $3}' | sed 's/\.//')
pages_speculative=$(vm_stat | grep "Pages speculative" | awk '{print $3}' | sed 's/\.//')
pages_total=$(sysctl -n hw.memsize)
pages_total=$((pages_total / 4096))

pages_available=$((pages_free + pages_inactive + pages_speculative))
memory_percent_free=$((100 * pages_available / pages_total))

# --- Write to log ---
echo "[$timestamp] System health check:" >> "$log_file"
echo "CPU usage: ${cpu_usage}% | Memory free: ${memory_percent_free}%" >> "$log_file"

# --- Check thresholds ---
if [ "$cpu_usage_int" -gt 80 ]; then
    echo "Warning: High CPU usage" >> "$log_file"
fi

if [ "$memory_percent_free" -lt 20 ]; then
    echo "Warning: Low available memory" >> "$log_file"
fi

echo "----------------------------------------" >> "$log_file"
```

Task 3: User Account Management

Write a bash script that

1. Reads a list of usernames from a file .

2. create a new user for each username.

3. generate random password for each user and save username and password to a file name credentials.txt

```
#!/bin/bash

input_file="usernames.txt"
output_file="credentials.txt"

if [[ $EUID -ne 0 ]]; then
    echo "Please run as root (use sudo)"
    exit 1
fi

> "$output_file"

while IFS= read -r username || [[ -n "$username" ]]; do
    if id "$username" &>/dev/null; then
        echo "User $username already exists. Skipping..."
    else
        password=$(openssl rand -base64 12)

        # Create user using dscl
        sudo dscl . -create /Users/$username
        sudo dscl . -create /Users/$username UserShell /bin/bash
        sudo dscl . -create /Users/$username RealName "$username"
        sudo dscl . -create /Users/$username UniqueID "$(dscl . -list /Users UniqueID | awk '{print $2}' | sort -n | tail -1 | awk '
        sudo dscl . -create /Users/$username PrimaryGroupID 20
        sudo dscl . -create /Users/$username NFSHomeDirectory /Users/$username
        sudo dscl . -passwd /Users/$username "$password"
        sudo createhomedir -c -u "$username" >/dev/null

        echo "$username:$password" >> "$output_file"
        echo "Created user: $username"
    fi
done < "$input_file"

echo "All credentials saved to $output_file"
```

Task 4: Automated Backup

Write a bash script that

1. Takes a directory path as input from user

2. Compress directory into a .tar.gz file

3. save the compressed file with a name that includes current date

```bash
#!/bin/bash

# Prompt user for directory path
read -p "Enter the path of the directory to compress: " dir_path

# Check if directory exists
if [[ ! -d "$dir_path" ]]; then
    echo "Directory does not exist."
    exit 1
fi

# Extract directory name
dir_name=$(basename "$dir_path")

# Generate output filename with date
timestamp=$(date +"%Y%m%d_%H%M%S")
archive_name="${dir_name}_${timestamp}.tar.gz"

# Compress the directory
tar -czf "$archive_name" -C "$(dirname "$dir_path")" "$dir_name"

echo "Directory compressed into: $archive_name"
```

Task 5: Simple TO-DO List

Write a bash script that

1. Implements a simple command-line to-do-list.

2. Allows user to add tasks,view tasks, remove tasks.

3. Saves the tasks to a file.

```
 #!/bin/bash

TODO_FILE="todo.txt"

# Ensure the to-do file exists
touch "$TODO_FILE"

# Function: Display menu
show_menu() {
    echo "□ Simple To-Do List"
    echo "-------------------------"
    echo "1. View tasks"
    echo "2. Add task"
    echo "3. Remove task"
    echo "4. Exit"
    echo "-------------------------"
}

# Function: View tasks
view_tasks() {
    if [[ ! -s "$TODO_FILE" ]]; then
        echo "No tasks found."
    else
        echo " Your Tasks:"
        nl -w2 -s'. ' "$TODO_FILE"
    fi
}

# Function: Add task
add_task() {
    read -p "Enter new task: " task
    echo "$task" >> "$TODO_FILE"
    echo "Task added!"
}

# Function: Remove task
remove_task() {
    view_tasks
    read -p "Enter task number to remove: " task_no
    if [[ "$task_no" =~ ^[0-9]+$ ]]; then
        sed -i '' "${task_no}d" "$TODO_FILE"  # macOS syntax: sed -i ''
        echo "Task removed!"
    else
        echo " Invalid input. Please enter a number."
    fi
}

# Main loop
while true; do
    show_menu
    read -p "Choose an option [1-4]: " choice

    case $choice in
        1) view_tasks ;;
        2) add_task ;;
        3) remove_task ;;
        4) echo "Goodbye!"; exit 0 ;;
        *) echo "Invalid option. Try again." ;;
    esac

    echo ""  # spacer line
done
```

Task 6: Automated Software Installation

Write a bash script that:

1. Reads a list of software package names from a file (eg packages.txt)

2. Install each package using appropriate package manager (apt,yum,etc)

3. log installation status of each package

```
 #!/bin/bash

PACKAGE_FILE="packages.txt"
LOG_FILE="install_log.txt"

# Check if Homebrew is installed
if ! command -v brew &>/dev/null; then
    echo " Homebrew not found. Install it from https://brew.sh/"
    exit 1
fi

# Check if package list file exists
if [[ ! -f "$PACKAGE_FILE" ]]; then
    echo " File '$PACKAGE_FILE' not found."
    exit 1
fi

# Empty the log file
> "$LOG_FILE"

# Read and install packages
while IFS= read -r package || [[ -n "$package" ]]; do
    if [[ -z "$package" ]]; then
        continue
    fi

    echo "Installing: $package"

    if brew list "$package" &>/dev/null; then
        echo "$package is already installed." | tee -a "$LOG_FILE"
    else
        if brew install "$package" >>"$LOG_FILE" 2>&1; then

            echo " Installed: $package" | tee -a "$LOG_FILE"
        else
            echo "Failed to install: $package" | tee -a "$LOG_FILE"
        fi
    fi
done < "$PACKAGE_FILE"
```

Task 7: Text File Processing

Write a bash script that

1. Takes a text file as input.

2. count and displays number of lines, words and characters in a file.

3. find and display longest word in the file.

```
#!/bin/bash

# 1. Prompt user for input file
read -p "Enter the path to a text file: " input_file

# 2. Check if file exists
if [[ ! -f "$input_file" ]]; then
    echo "File not found!"
    exit 1
fi

# 3. Count lines, words, and characters
line_count=$(wc -l < "$input_file")
word_count=$(wc -w < "$input_file")
char_count=$(wc -m < "$input_file")

echo "File Analysis:"
echo "----------------------------"
echo "Lines      : $line_count"
echo "Words      : $word_count"
echo "Characters : $char_count"

# 4. Find the longest word
# Remove punctuation, split words, find longest
longest_word=$(tr -d '[:punct:]' < "$input_file" | \
               tr '[:space:]' '\n' | \
               awk '{ if (length > max) { max = length; word = $0 } } END { print word }')

echo "Longest word: $longest_word"
```