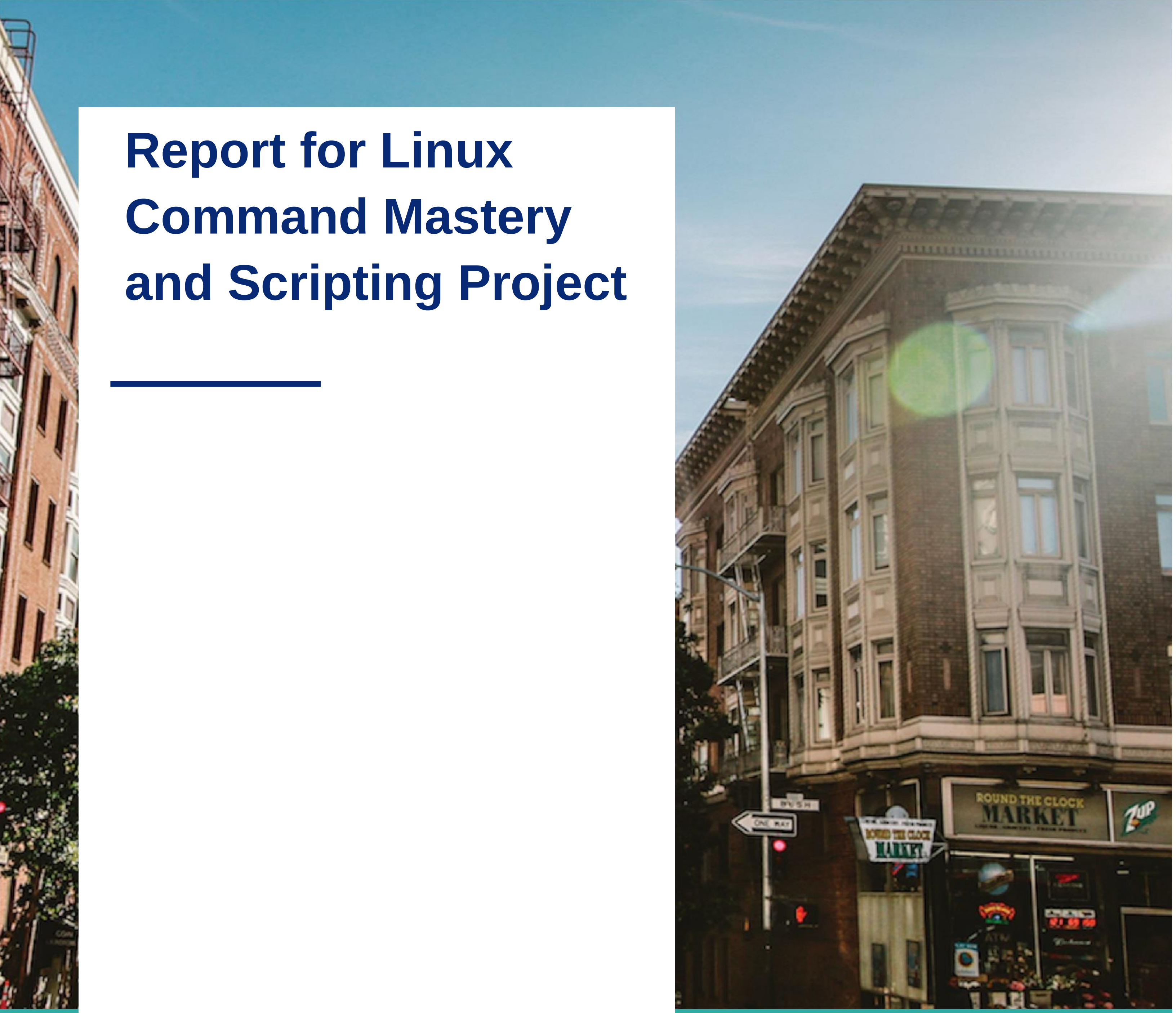
Report for Linux Command Mastery and Scripting Project



JULY 13

Authored by: Deya' Aldeen AL-Bettar

Table Content

Introduction

This project involves developing two shell scripts for common Linux system administration tasks. The first script, `backup_script.sh`, automates directory backups with compression and logging. The second script, `health_check.sh`, monitors system health by checking disk space, memory usage, running services, and recent system updates.

Backup Script

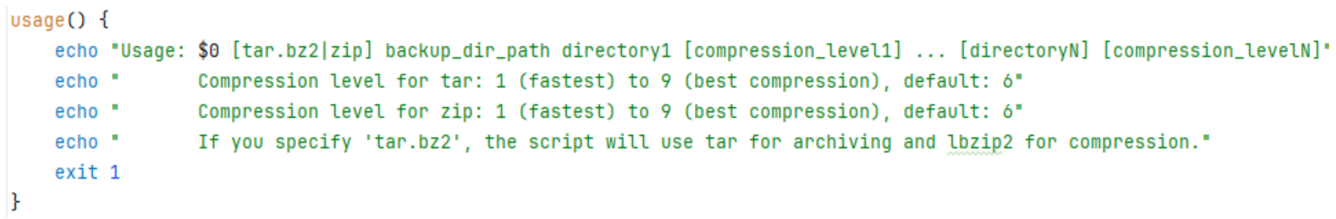
Purpose: This script automates the process of backing up specified directories, compressing them, and logging the backup process. It supports two compression methods: lbzip2 and zip.

Note:

- **Tar** is primarily an archiving tool that packages multiple files and directories into a single archive without compressing them. When used with **lbzip2**, it provides compression in addition to archiving.
- **Zip**, on the other hand, serves both as an archiving and compression tool, creating a compressed archive in a single step.

Functionality:

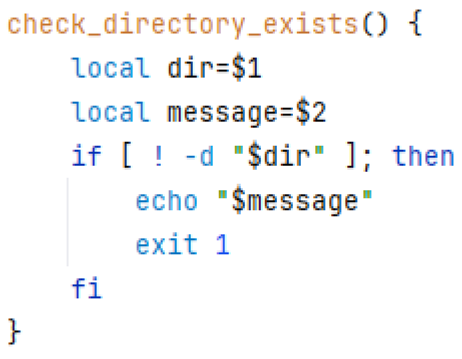
1. **Usage Information and Initial Checks:**



```
usage() {
    echo "Usage: $0 [tar.bz2|zip] backup_dir_path directory1 [compression_level1] ... [directoryN] [compression_levelN]"
    echo "       Compression level for tar: 1 (fastest) to 9 (best compression), default: 6"
    echo "       Compression level for zip: 1 (fastest) to 9 (best compression), default: 6"
    echo "       If you specify 'tar.bz2', the script will use tar for archiving and lbzip2 for compression."
    exit 1
}
```

Explanation: This function displays how to use the script and provides details on the arguments it accepts. It explains the compression methods and levels, helping users understand how to run the script correctly. If incorrect or insufficient arguments are provided, it exits the script.

2. **Directory Existence Check:**



```
check_directory_exists() {
    local dir=$1
    local message=$2
    if [ ! -d "$dir" ]; then
        echo "$message"
        exit 1
    fi
}
```

Explanation: This function checks if a given directory exists. If it doesn't, it prints an error message and exits the script. This ensures the script only proceeds with valid directories.

3. **Backup Directory Creation:**

```
create_backup_dir() {
    local dir=$1
    mkdir -p "$dir"
}
```

Explanation: This function creates the backup directory if it doesn't already exist. The -p option ensures no error is thrown if the directory already exists, making the script more robust.

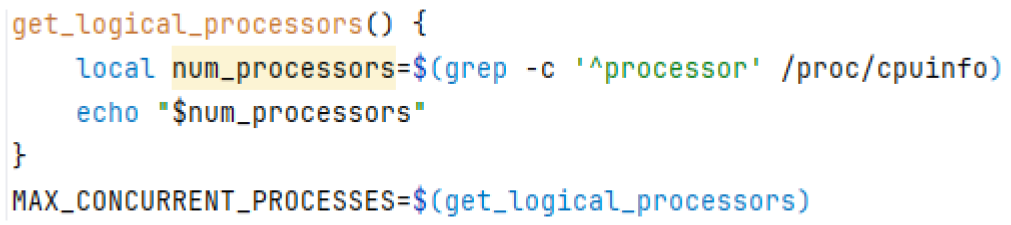
4. **Logging Functions:**

```
start_logging() {
    local log_file=$1
    local current_datetime=$(date +"%Y-%m-%d %H:%M:%S")
    echo "Backup started at $current_datetime" >> "$log_file"
}
log_message() {
    local log_file=$1
    local message=$2
    local current_datetime=$(date +"%Y-%m-%d %H:%M:%S")
    echo "[$current_datetime] $message" >> "$log_file"
}
log_error() {
    local log_file=$1
    local message=$2
    local current_datetime=$(date +"%Y-%m-%d %H:%M:%S")
    echo "[$current_datetime] ERROR: $message" >> "$log_file" >&2
}
```

Explanation: These functions handle logging:

- start_logging logs the start time of the backup.
- log_message logs informational messages with timestamps.
- log_error logs error messages with timestamps and directs them to standard error (stderr).

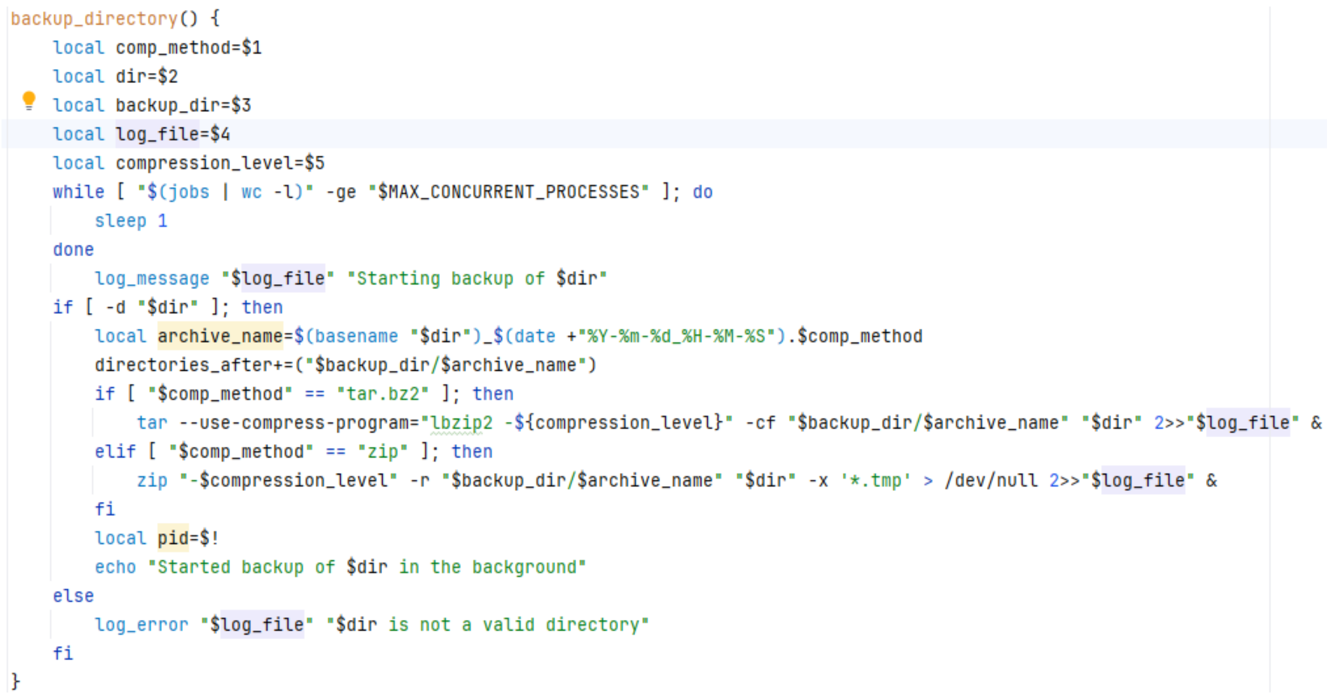
5. **Determine Logical Processors:**



```
get_logical_processors() {
    local num_processors=$(grep -c '^processor' /proc/cpuinfo)
    echo "$num_processors"
}
MAX_CONCURRENT_PROCESSES=$(get_logical_processors)
```

Explanation: This function determines the number of logical processors available on the system by counting the processor entries in /proc/cpuinfo. This information is used to limit the number of concurrent backup processes, optimizing performance.

6. **Backup Directory Function:**

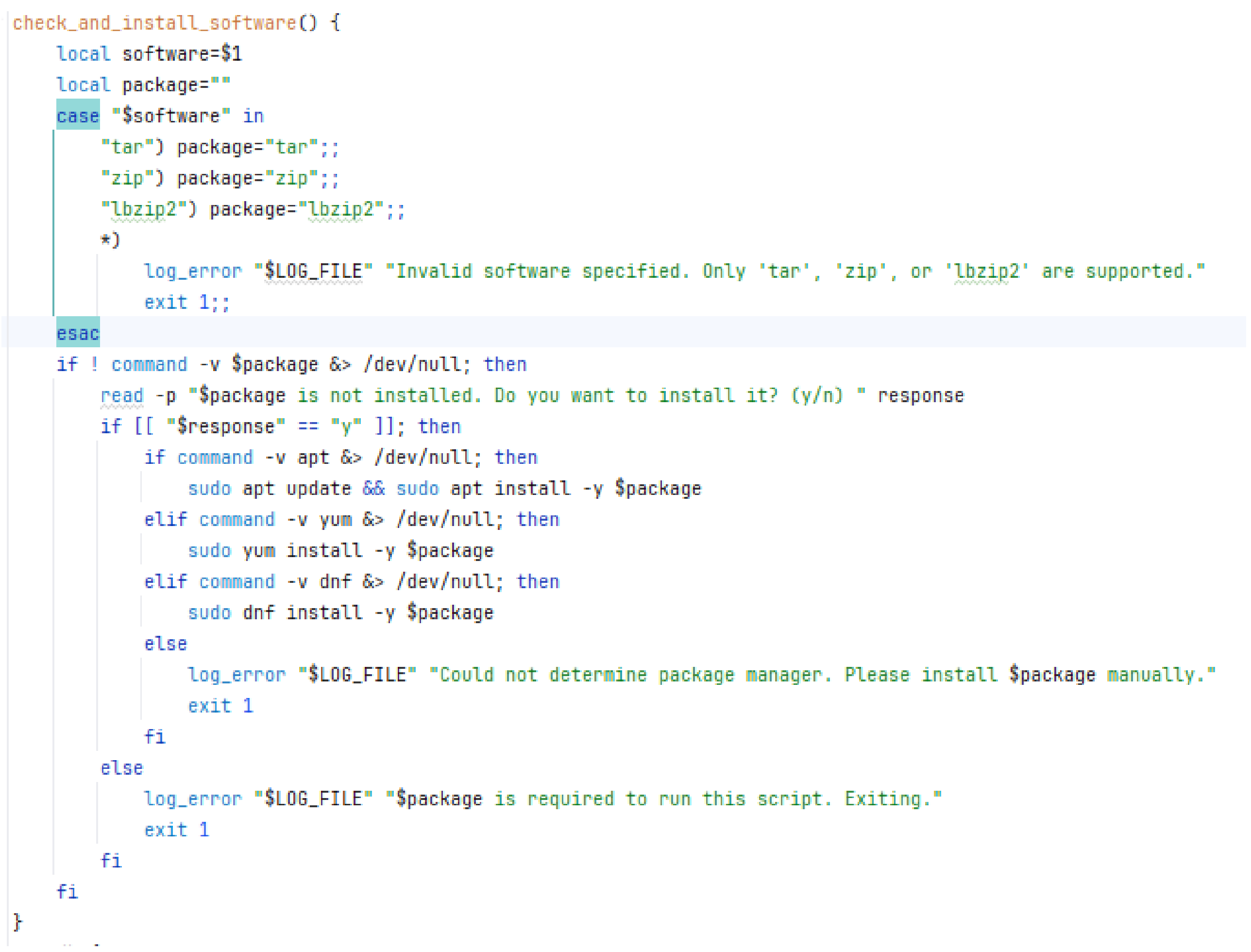


```
backup_directory() {
    local comp_method=$1
    local dir=$2
    local backup_dir=$3
    local log_file=$4
    local compression_level=$5
    while [ "$(jobs | wc -l)" -ge "$MAX_CONCURRENT_PROCESSES" ]; do
        sleep 1
    done
        log_message "$log_file" "Starting backup of $dir"
    if [ -d "$dir" ]; then
        local archive_name=$(basename "$dir")_$(date +"%Y-%m-%d_%H-%M-%S").$comp_method
        directories_after+=("$backup_dir/$archive_name")
        if [ "$comp_method" == "tar.bz2" ]; then
            tar --use-compress-program="lbzip2 -${compression_level}" -cf "$backup_dir/$archive_name" "$dir" 2>>"$log_file" &
        elif [ "$comp_method" == "zip" ]; then
            zip "-$compression_level" -r "$backup_dir/$archive_name" "$dir" -x '*.tmp' > /dev/null 2>>"$log_file" &
        fi
        local pid=$!
        echo "Started backup of $dir in the background"
    else
        log_error "$log_file" "$dir is not a valid directory"
    fi
}
```

Explanation: This function performs the backup:

- Waits if the number of background jobs exceeds the number of logical processors.
- Logs the start of the backup.
- Compresses the directory using either lbzip2 (with lbzip2 for parallel compression) or zip.
- Runs the compression in the background, allowing multiple backups to proceed
- concurrently **by appending & at the end.**

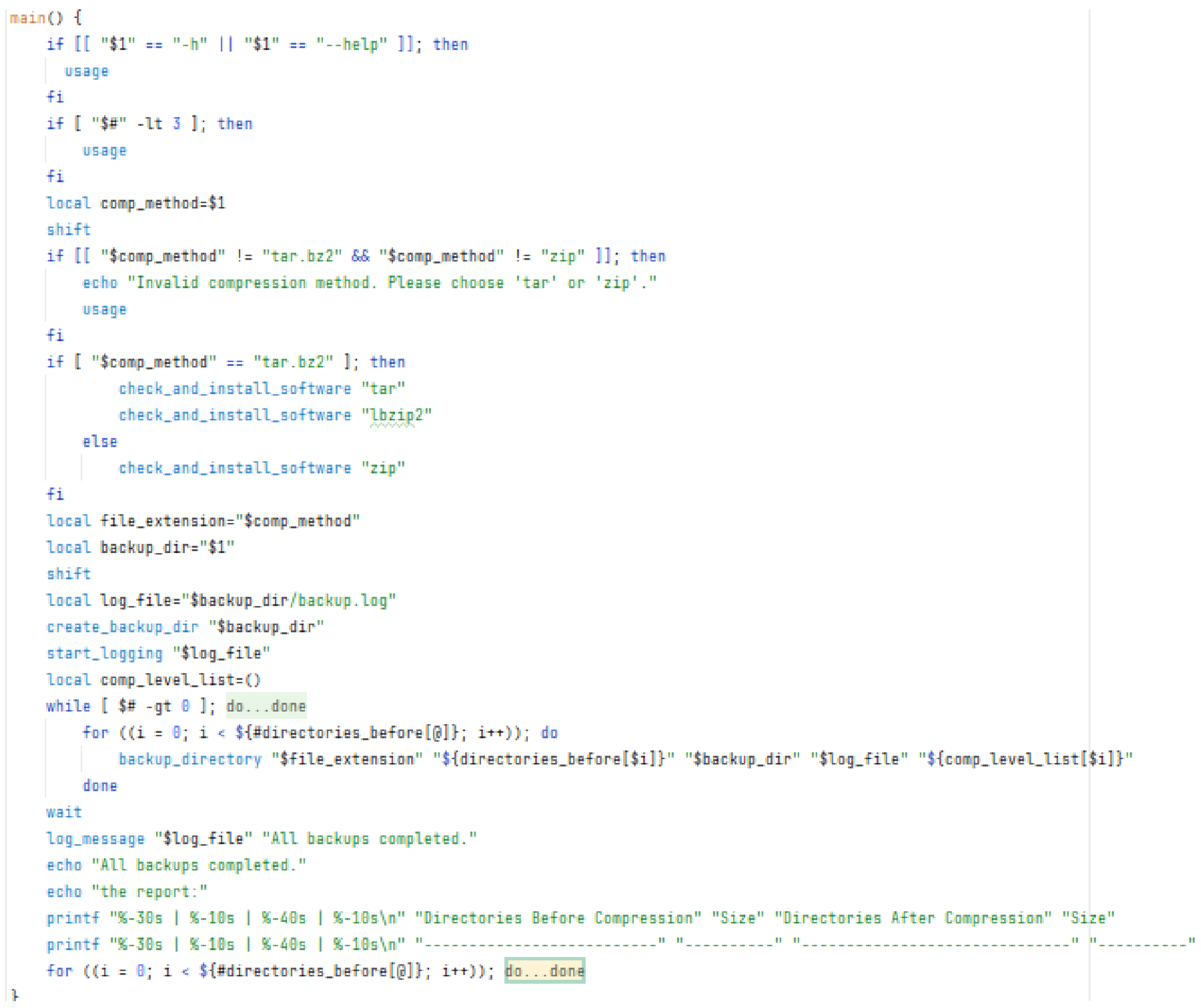
7. **Software Check and Installation:**



```
check_and_install_software() {
    local software=$1
    local package=""
    case "$software" in
        "tar") package="tar";;
        "zip") package="zip";;
        "lbzip2") package="lbzip2";;
        *)
            log_error "$LOG_FILE" "Invalid software specified. Only 'tar', 'zip', or 'lbzip2' are supported."
            exit 1;;
    esac
    if ! command -v $package &> /dev/null; then
        read -p "$package is not installed. Do you want to install it? (y/n) " response
        if [[ "$response" == "y" ]]; then
            if command -v apt &> /dev/null; then
                sudo apt update && sudo apt install -y $package
            elif command -v yum &> /dev/null; then
                sudo yum install -y $package
            elif command -v dnf &> /dev/null; then
                sudo dnf install -y $package
            else
                log_error "$LOG_FILE" "Could not determine package manager. Please install $package manually."
                exit 1
            fi
        else
            log_error "$LOG_FILE" "$package is required to run this script. Exiting."
            exit 1
        fi
    fi
}
```

Explanation: This function checks if the necessary software (tar, zip, lbzip2) is installed. If not, it prompts the user to install it and attempts to do so using the appropriate package manager (apt, yum, dnf). This ensures the script can perform its tasks without missing dependencies.

8. **Main Function:**



```bash
main() {
    if [[ "$1" == "-h" || "$1" == "--help" ]]; then
      usage
    fi
    if [ "$#" -lt 3 ]; then
        usage
    fi
    local comp_method=$1
    shift
    if [[ "$comp_method" != "tar.bz2" && "$comp_method" != "zip" ]]; then
        echo "Invalid compression method. Please choose 'tar' or 'zip'."
        usage
    fi
    if [ "$comp_method" == "tar.bz2" ]; then
            check_and_install_software "tar"
            check_and_install_software "lbzip2"
        else
            check_and_install_software "zip"
    fi
    local file_extension="$comp_method"
    local backup_dir="$1"
    shift
    local log_file="$backup_dir/backup.log"
    create_backup_dir "$backup_dir"
    start_logging "$log_file"
    local comp_level_list=()
    while [ $# -gt 0 ]; do...done
        for ((i = 0; i < ${#directories_before[@]}; i++)); do
            backup_directory "$file_extension" "${directories_before[$i]}" "$backup_dir" "$log_file" "${comp_level_list[$i]}"
        done
    wait
    log_message "$log_file" "All backups completed."
    echo "All backups completed."
    echo "the report:"
    printf "%-30s | %-10s | %-40s | %-10s\n" "Directories Before Compression" "Size" "Directories After Compression" "Size"
    printf "%-30s | %-10s | %-40s | %-10s\n" "------------------------------" "----------" "----------------------------------------" "----------"
    for ((i = 0; i < ${#directories_before[@]}; i++)); do...done
}
```

Explanation: The main function orchestrates the script's execution:

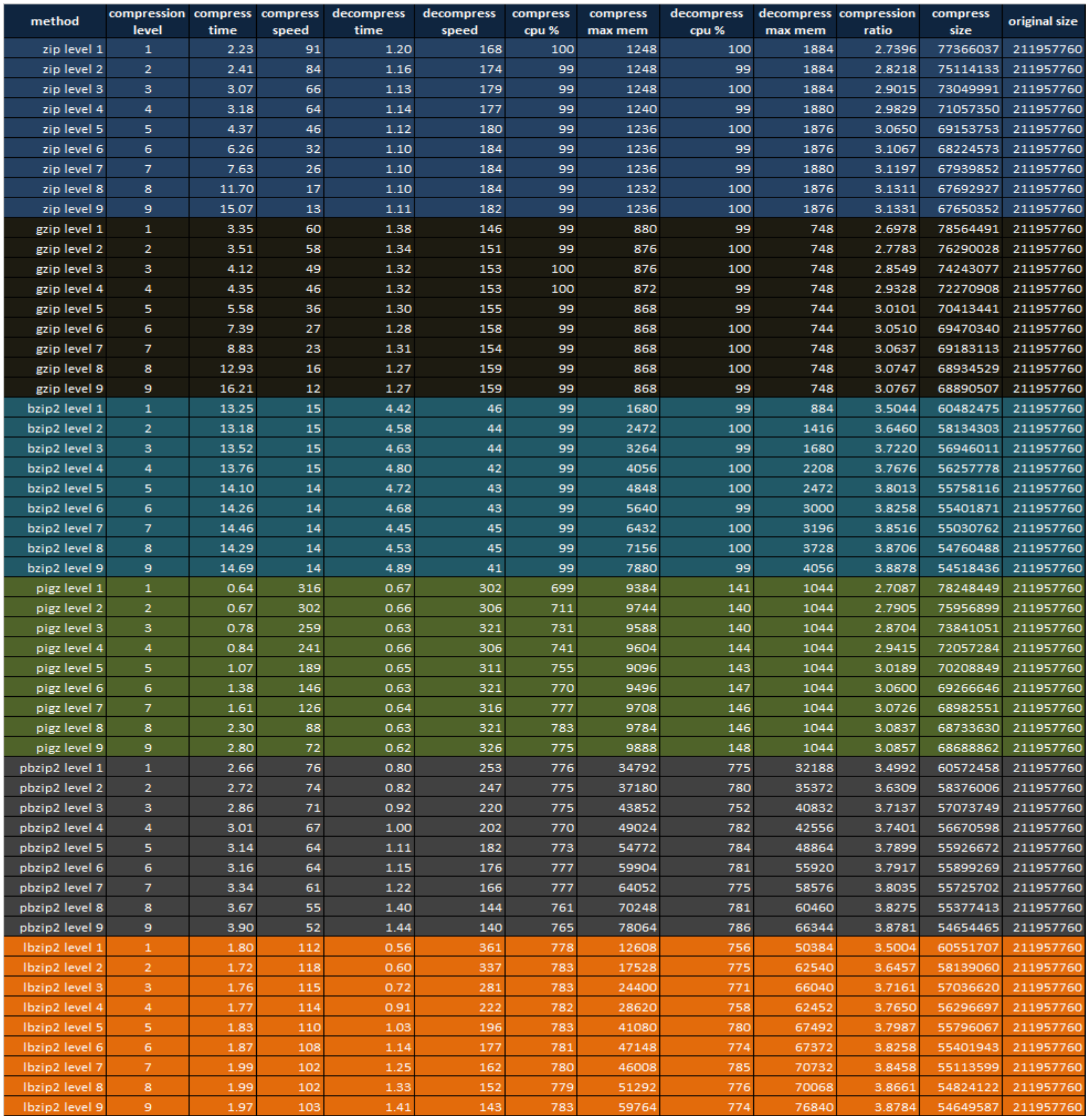
- Handles usage and argument validation.
- Installs required software if necessary.
- Initializes directories and logging.
- Iterates over the input directories, initiating the backup for each with the specified compression level.
- Waits for all background jobs to complete.
- Logs the completion of all backups and prints a summary report showing the directories before compression, their sizes, and the corresponding backup files with their sizes.

Performance Insights:

- Using lbzip2 for parallel compression improves performance by leveraging multiple CPU cores.
- Background processing allows concurrent backups, enhancing overall efficiency.

Why Use lbzip2?

- After searching and reviewing many benchmarks on compression tools, I chose lbzip2 because it offers a good balance between compression ratio and speed.



method	compression level	compress time	compress speed	decompress time	decompress speed	compress cpu %	compress max mem	decompress cpu %	decompress max mem	compression ratio	compress size	original size
zip level 1	1	2.23	91	1.20	168	100	1248	100	1884	2.7396	77366037	211957760
zip level 2	2	2.41	84	1.16	174	99	1248	99	1884	2.8218	75114133	211957760
zip level 3	3	3.07	66	1.13	179	99	1248	100	1884	2.9015	73049991	211957760
zip level 4	4	3.18	64	1.14	177	99	1240	99	1880	2.9829	71057350	211957760
zip level 5	5	4.37	46	1.12	180	99	1236	100	1876	3.0650	69153753	211957760
zip level 6	6	6.26	32	1.10	184	99	1236	99	1876	3.1067	68224573	211957760
zip level 7	7	7.63	26	1.10	184	99	1236	99	1880	3.1197	67939852	211957760
zip level 8	8	11.70	17	1.10	184	99	1232	100	1876	3.1311	67692927	211957760
zip level 9	9	15.07	13	1.11	182	99	1236	100	1876	3.1331	67650352	211957760
gzip level 1	1	3.35	60	1.38	146	99	880	99	748	2.6978	78564491	211957760
gzip level 2	2	3.51	58	1.34	151	99	876	100	748	2.7783	76290028	211957760
gzip level 3	3	4.12	49	1.32	153	100	876	100	748	2.8549	74243077	211957760
gzip level 4	4	4.35	46	1.32	153	100	872	99	748	2.9328	72270908	211957760
gzip level 5	5	5.58	36	1.30	155	99	868	99	744	3.0101	70413441	211957760
gzip level 6	6	7.39	27	1.28	158	99	868	100	744	3.0510	69470340	211957760
gzip level 7	7	8.83	23	1.31	154	99	868	100	748	3.0637	69183113	211957760
gzip level 8	8	12.93	16	1.27	159	99	868	100	748	3.0747	68934529	211957760
gzip level 9	9	16.21	12	1.27	159	99	868	99	748	3.0767	68890507	211957760
bzip2 level 1	1	13.25	15	4.42	46	99	1680	99	884	3.5044	60482475	211957760
bzip2 level 2	2	13.18	15	4.58	44	99	2472	100	1416	3.6460	58134303	211957760
bzip2 level 3	3	13.52	15	4.63	44	99	3264	99	1680	3.7220	56946011	211957760
bzip2 level 4	4	13.76	15	4.80	42	99	4056	100	2208	3.7676	56257778	211957760
bzip2 level 5	5	14.10	14	4.72	43	99	4848	100	2472	3.8013	55758116	211957760
bzip2 level 6	6	14.26	14	4.68	43	99	5640	99	3000	3.8258	55401871	211957760
bzip2 level 7	7	14.46	14	4.45	45	99	6432	100	3196	3.8516	55030762	211957760
bzip2 level 8	8	14.29	14	4.53	45	99	7156	100	3728	3.8706	54760488	211957760
bzip2 level 9	9	14.69	14	4.89	41	99	7880	99	4056	3.8878	54518436	211957760
pigz level 1	1	0.64	316	0.67	302	699	9384	141	1044	2.7087	78248449	211957760
pigz level 2	2	0.67	302	0.66	306	711	9744	140	1044	2.7905	75956899	211957760
pigz level 3	3	0.78	259	0.63	321	731	9588	140	1044	2.8704	73841051	211957760
pigz level 4	4	0.84	241	0.66	306	741	9604	144	1044	2.9415	72057284	211957760
pigz level 5	5	1.07	189	0.65	311	755	9096	143	1044	3.0189	70208849	211957760
pigz level 6	6	1.38	146	0.63	321	770	9496	147	1044	3.0600	69266646	211957760
pigz level 7	7	1.61	126	0.64	316	777	9708	146	1044	3.0726	68982551	211957760
pigz level 8	8	2.30	88	0.63	321	783	9784	146	1044	3.0837	68733630	211957760
pigz level 9	9	2.80	72	0.62	326	775	9888	148	1044	3.0857	68688862	211957760
pbzip2 level 1	1	2.66	76	0.80	253	776	34792	775	32188	3.4992	60572458	211957760
pbzip2 level 2	2	2.72	74	0.82	247	775	37180	780	35372	3.6309	58376006	211957760
pbzip2 level 3	3	2.86	71	0.92	220	775	43852	752	40832	3.7137	57073749	211957760
pbzip2 level 4	4	3.01	67	1.00	202	770	49024	782	42556	3.7401	56670598	211957760
pbzip2 level 5	5	3.14	64	1.11	182	773	54772	784	48864	3.7899	55926672	211957760
pbzip2 level 6	6	3.16	64	1.15	176	777	59904	781	55920	3.7917	55899269	211957760
pbzip2 level 7	7	3.34	61	1.22	166	777	64052	775	58576	3.8035	55725702	211957760
pbzip2 level 8	8	3.67	55	1.40	144	761	70248	781	60460	3.8275	55377413	211957760
pbzip2 level 9	9	3.90	52	1.44	140	765	78064	786	66344	3.8781	54654465	211957760
lbzip2 level 1	1	1.80	112	0.56	361	778	12608	756	50384	3.5004	60551707	211957760
lbzip2 level 2	2	1.72	118	0.60	337	783	17528	775	62540	3.6457	58139060	211957760
lbzip2 level 3	3	1.76	115	0.72	281	783	24400	771	66040	3.7161	57036620	211957760
lbzip2 level 4	4	1.77	114	0.91	222	782	28620	758	62452	3.7650	56296697	211957760
lbzip2 level 5	5	1.83	110	1.03	196	783	41080	780	67492	3.7987	55796067	211957760
lbzip2 level 6	6	1.87	108	1.14	177	781	47148	774	67372	3.8258	55401943	211957760
lbzip2 level 7	7	1.99	102	1.25	162	780	46008	785	70732	3.8458	55113599	211957760
lbzip2 level 8	8	1.99	102	1.33	152	779	51292	776	70068	3.8661	54824122	211957760
lbzip2 level 9	9	1.97	103	1.41	143	783	59764	774	76840	3.8784	54649587	211957760

- As you see in this image, lbzip2 seems to have the best speed to achieve a compression ratio of 3.73-3.80, followed by pbzip2, then plzip, then pzstd then pxz.
- You can visit the following links for more information on benchmarking and explanations about compression tools:
    - https://community.centminmod.com/threads/compression-comparison-benchmarks-zstd-vs-brotli-vs-pigz-vs-bzip2-vs-xz-etc.12764/
    - https://www.baeldung.com/linux/parallel-archiving

- **Multi-threading**: lbzip2 utilizes multiple cores to perform compression, making it significantly faster than single-threaded tools, especially on modern multi-core processors.
- **Compression Ratio**: While being fast, lbzip2 also achieves a high compression ratio, reducing the size of the compressed files effectively.
- **Compatibility**: lbzip2 is compatible with the bzip2 file format, ensuring that the compressed files can be decompressed with standard bzip2 tools if needed.
- **Resource Efficiency**: By using multiple threads, lbzip2 makes efficient use of available system resources, reducing the time needed for large compression tasks.

Why Not Use Non-Parallel Compression Tools?

Non-parallel compression tools such as standard bzip2, gzip, and others, are limited to using a single CPU core for compression. This limitation leads to several drawbacks:

1. **Slower Performance**: Single-threaded compression tools take significantly longer to compress large files, as they cannot leverage the full processing power of modern multi-core systems.
2. **Inefficient Resource Use**: In systems with multiple cores, non-parallel tools fail to utilize available resources efficiently, leaving many CPU cores idle during the compression process.
3. **Scalability**: As file sizes increase, the time required for compression with single-threaded tools grows linearly, making them impractical for large-scale tasks in high-performance computing environments.

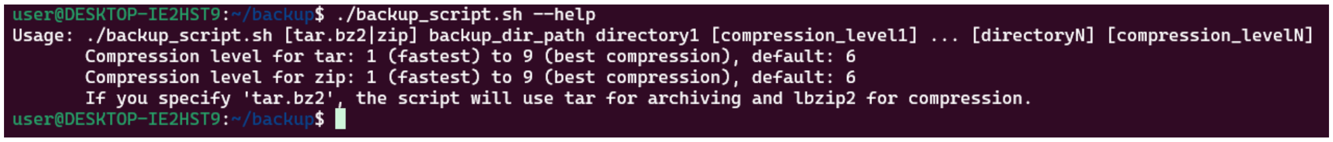
Why Use zip?

Zip is another popular compression tool, and it is chosen for different reasons:

1. **Wide Compatibility**: The zip format is widely supported across different operating systems and platforms, making it ideal for sharing compressed files.
2. **Directory Compression**: zip can compress entire directories while preserving the directory structure, making it convenient for packaging multiple files and directories.
3. **Exclusion of Files**: zip supports excluding specific file types (e.g., *.tmp), providing more control over the contents of the compressed archive.
4. **Ease of Use**: The zip command is straightforward and user-friendly, making it accessible for users with varying levels of technical expertise.

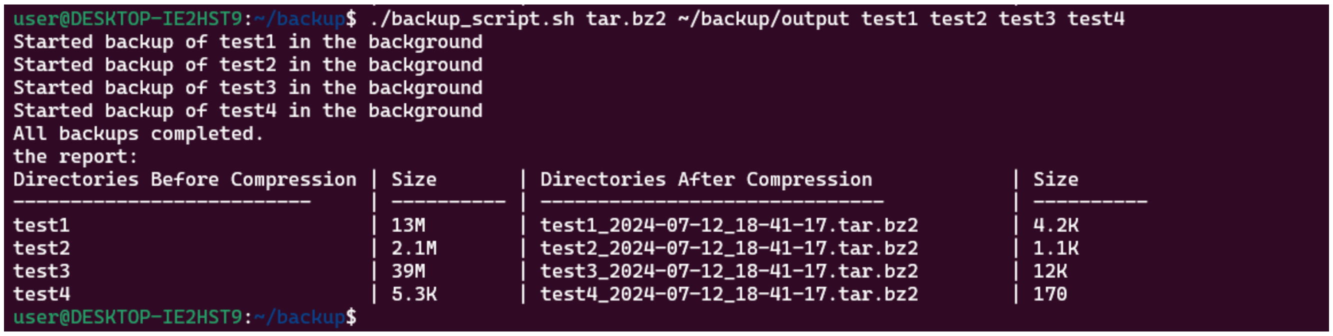
Sample Execution and Output:

Usage Information: ./backup_script.sh --help



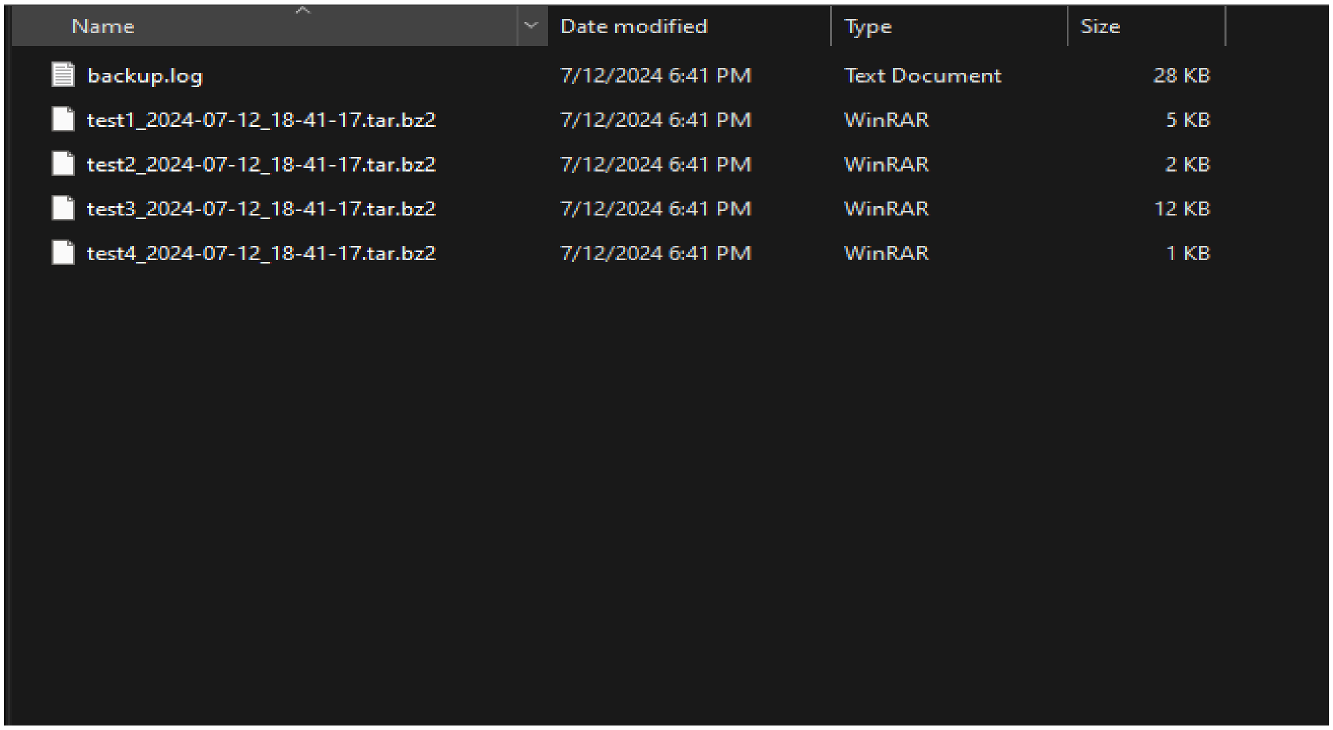
```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh --help
Usage: ./backup_script.sh [tar.bz2|zip] backup_dir_path directory1 [compression_level1] ... [directoryN] [compression_levelN]
       Compression level for tar: 1 (fastest) to 9 (best compression), default: 6
       Compression level for zip: 1 (fastest) to 9 (best compression), default: 6
       If you specify 'tar.bz2', the script will use tar for archiving and lbzip2 for compression.
user@DESKTOP-IE2HST9:~/backup$
```

Tar and lbzip2: ./backup_script.sh tar.bz2 ~/backup/output test1 test2 test3 test4

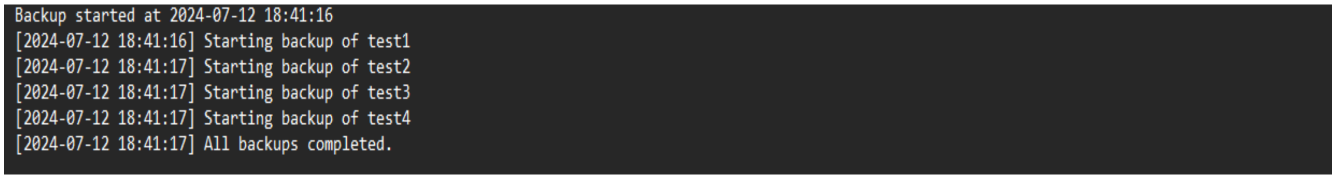


```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh tar.bz2 ~/backup/output test1 test2 test3 test4
Started backup of test1 in the background
Started backup of test2 in the background
Started backup of test3 in the background
Started backup of test4 in the background
All backups completed.
the report:
Directories Before Compression | Size       | Directories After Compression          | Size
------------------------------ | ---------- | -------------------------------        | ----------
test1                          | 13M        | test1_2024-07-12_18-41-17.tar.bz2      | 4.2K
test2                          | 2.1M       | test2_2024-07-12_18-41-17.tar.bz2      | 1.1K
test3                          | 39M        | test3_2024-07-12_18-41-17.tar.bz2      | 12K
test4                          | 5.3K       | test4_2024-07-12_18-41-17.tar.bz2      | 170
user@DESKTOP-IE2HST9:~/backup$
```

Output and Log File:

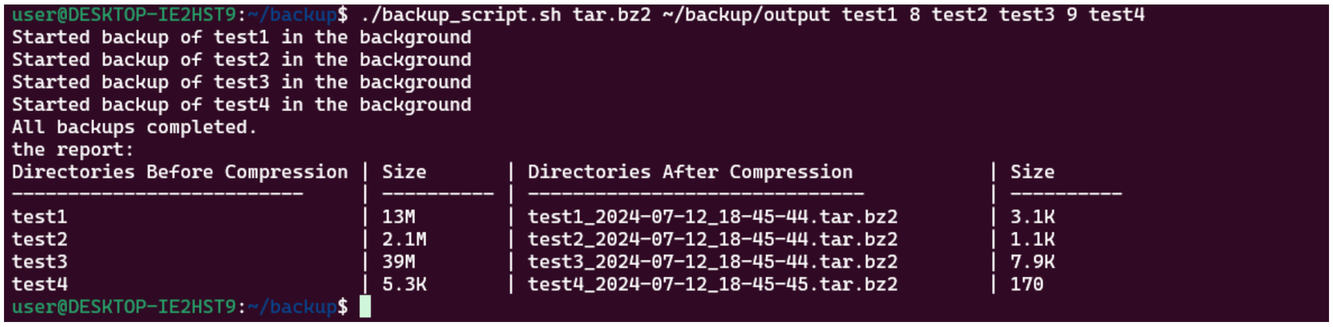


Name	Date modified	Type	Size
backup.log	7/12/2024 6:41 PM	Text Document	28 KB
test1_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	5 KB
test2_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	2 KB
test3_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	12 KB
test4_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	1 KB



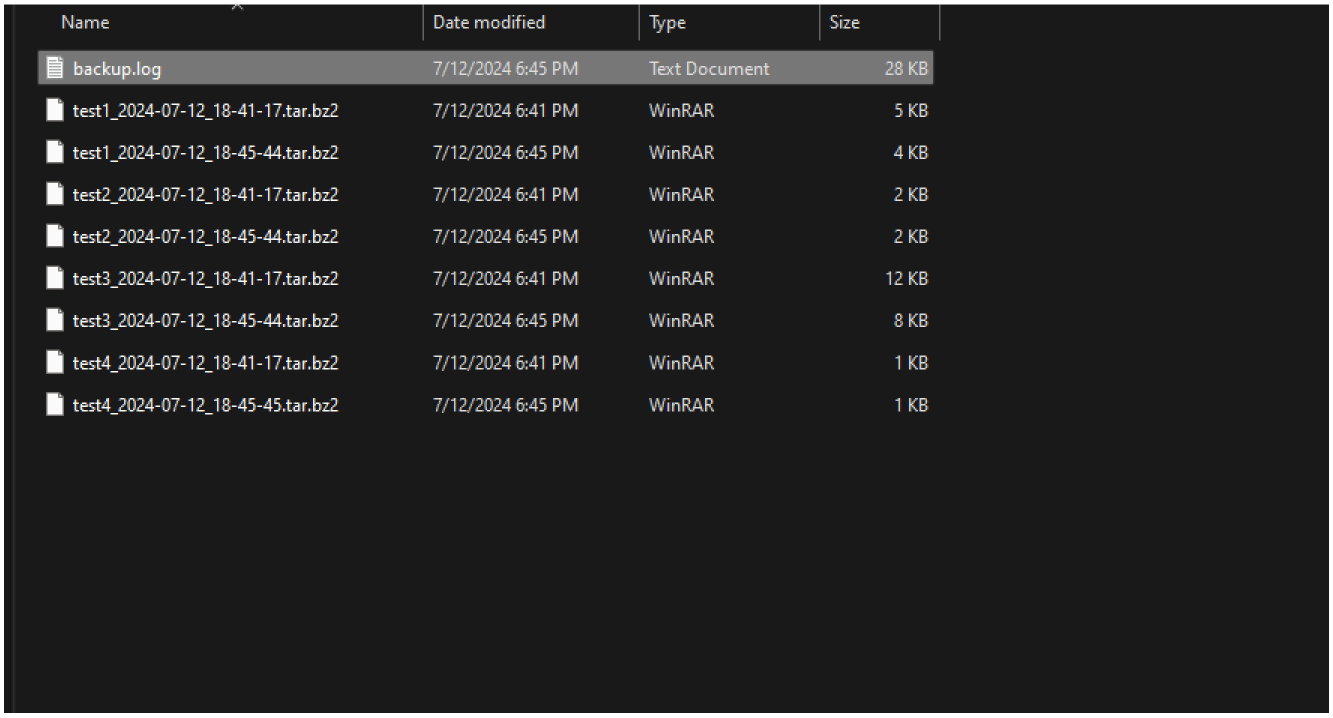
```
Backup started at 2024-07-12 18:41:16
[2024-07-12 18:41:16] Starting backup of test1
[2024-07-12 18:41:17] Starting backup of test2
[2024-07-12 18:41:17] Starting backup of test3
[2024-07-12 18:41:17] Starting backup of test4
[2024-07-12 18:41:17] All backups completed.
```

Tar and lbzip2 with compression level: ./backup_script.sh tar.bz2 ~/backup/output test1 8 test2 test3 9 test4

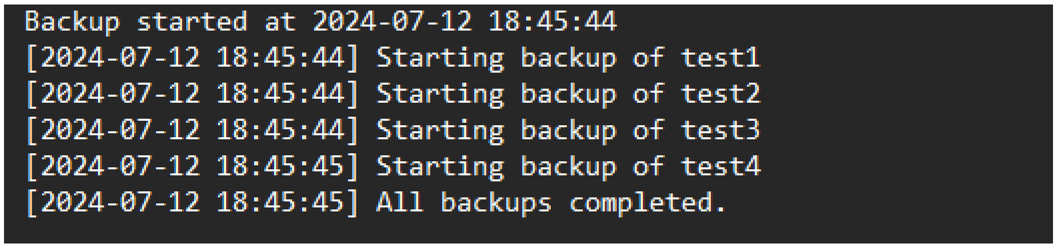


```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh tar.bz2 ~/backup/output test1 8 test2 test3 9 test4
Started backup of test1 in the background
Started backup of test2 in the background
Started backup of test3 in the background
Started backup of test4 in the background
All backups completed.
the report:
Directories Before Compression | Size       | Directories After Compression          | Size
------------------------------ | ---------- | ------------------------------         | ----------
test1                          | 13M        | test1_2024-07-12_18-45-44.tar.bz2      | 3.1K
test2                          | 2.1M       | test2_2024-07-12_18-45-44.tar.bz2      | 1.1K
test3                          | 39M        | test3_2024-07-12_18-45-44.tar.bz2      | 7.9K
test4                          | 5.3K       | test4_2024-07-12_18-45-45.tar.bz2      | 170
user@DESKTOP-IE2HST9:~/backup$
```

Output and Log File:

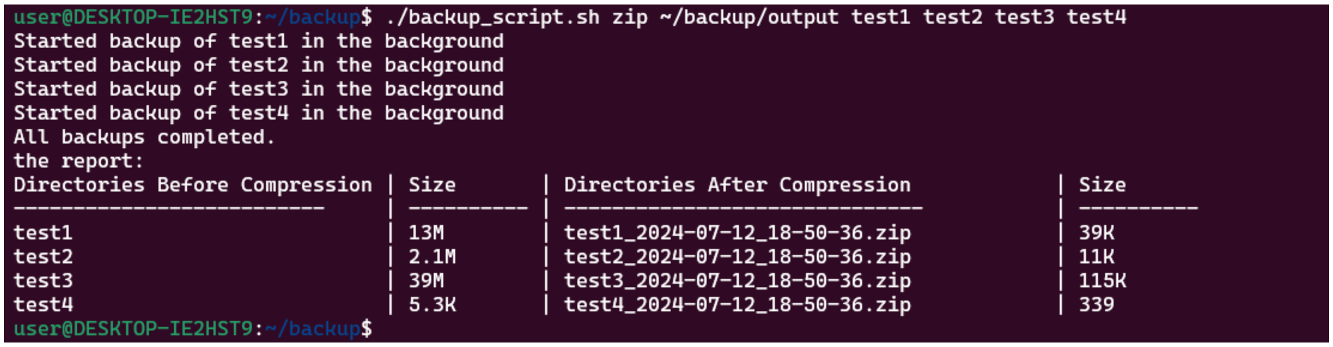


Name	Date modified	Type	Size
backup.log	7/12/2024 6:45 PM	Text Document	28 KB
test1_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	5 KB
test1_2024-07-12_18-45-44.tar.bz2	7/12/2024 6:45 PM	WinRAR	4 KB
test2_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	2 KB
test2_2024-07-12_18-45-44.tar.bz2	7/12/2024 6:45 PM	WinRAR	2 KB
test3_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	12 KB
test3_2024-07-12_18-45-44.tar.bz2	7/12/2024 6:45 PM	WinRAR	8 KB
test4_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	1 KB
test4_2024-07-12_18-45-45.tar.bz2	7/12/2024 6:45 PM	WinRAR	1 KB



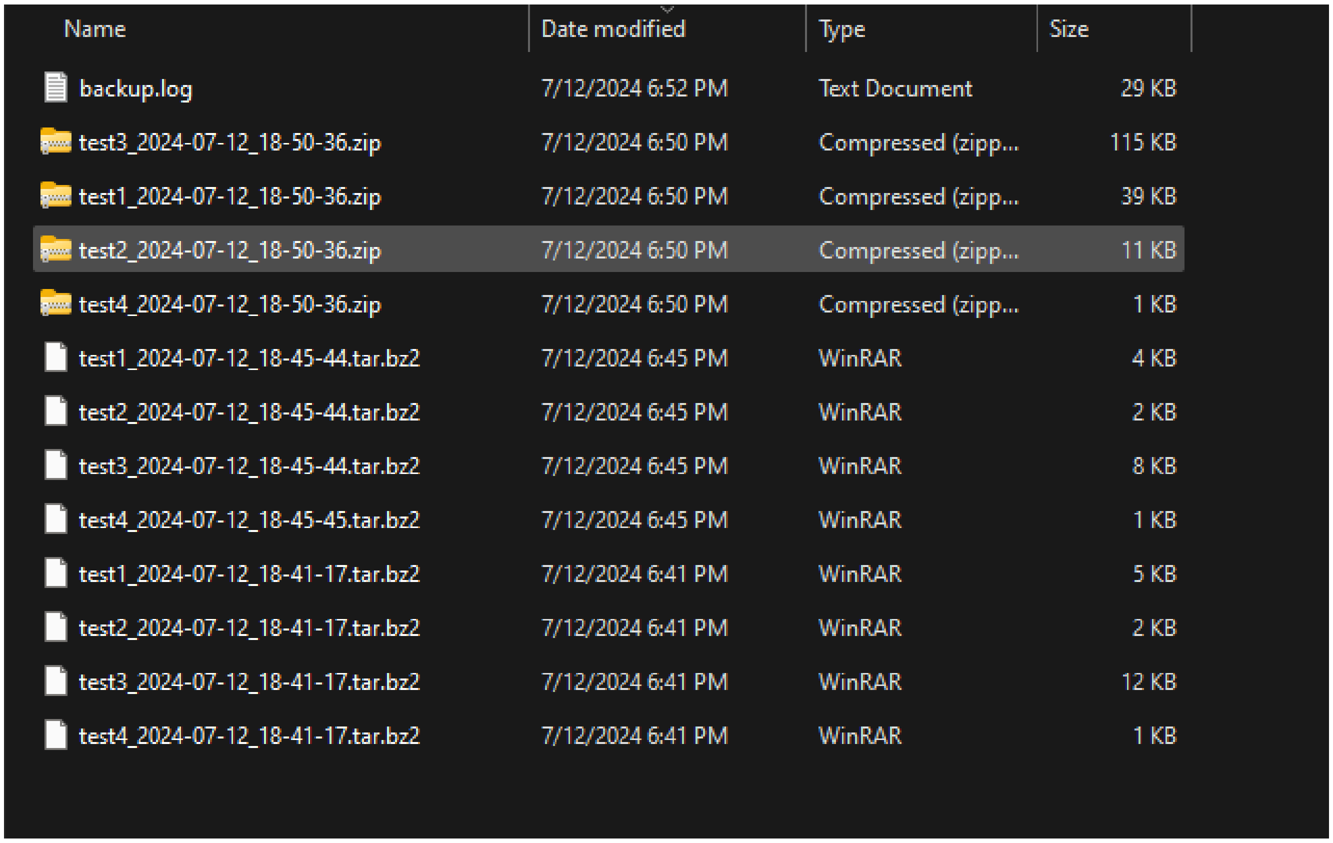
```
Backup started at 2024-07-12 18:45:44
[2024-07-12 18:45:44] Starting backup of test1
[2024-07-12 18:45:44] Starting backup of test2
[2024-07-12 18:45:44] Starting backup of test3
[2024-07-12 18:45:45] Starting backup of test4
[2024-07-12 18:45:45] All backups completed.
```

Zip: ./backup_script.sh zip ~/backup/output test1 test2 test3 test4

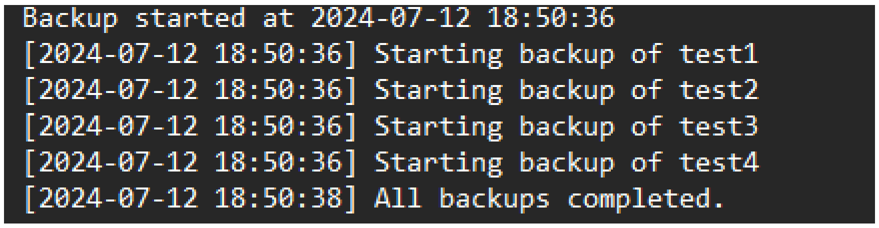


```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh zip ~/backup/output test1 test2 test3 test4
Started backup of test1 in the background
Started backup of test2 in the background
Started backup of test3 in the background
Started backup of test4 in the background
All backups completed.
the report:
Directories Before Compression | Size       | Directories After Compression          | Size
--------------------------     | ---------- | ------------------------------         | ----------
test1                          | 13M        | test1_2024-07-12_18-50-36.zip          | 39K
test2                          | 2.1M       | test2_2024-07-12_18-50-36.zip          | 11K
test3                          | 39M        | test3_2024-07-12_18-50-36.zip          | 115K
test4                          | 5.3K       | test4_2024-07-12_18-50-36.zip          | 339
user@DESKTOP-IE2HST9:~/backup$
```

Output and Log File:

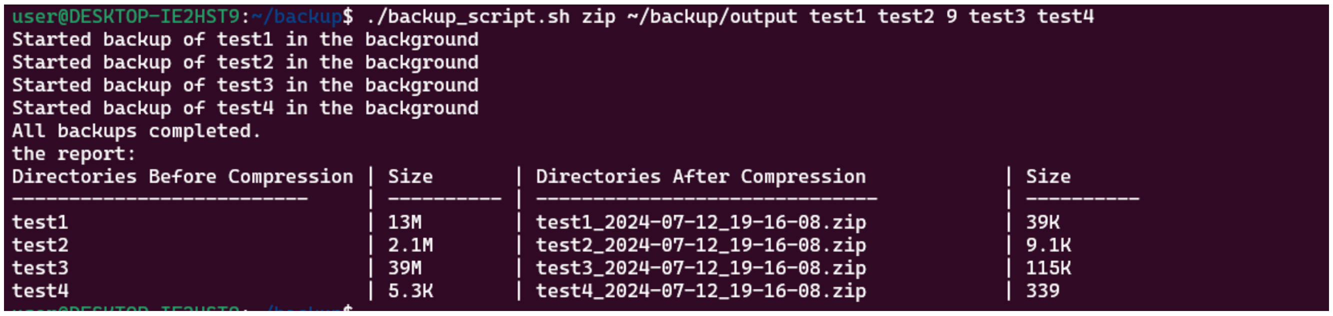


Name	Date modified	Type	Size
backup.log	7/12/2024 6:52 PM	Text Document	29 KB
test3_2024-07-12_18-50-36.zip	7/12/2024 6:50 PM	Compressed (zipp...	115 KB
test1_2024-07-12_18-50-36.zip	7/12/2024 6:50 PM	Compressed (zipp...	39 KB
test2_2024-07-12_18-50-36.zip	7/12/2024 6:50 PM	Compressed (zipp...	11 KB
test4_2024-07-12_18-50-36.zip	7/12/2024 6:50 PM	Compressed (zipp...	1 KB
test1_2024-07-12_18-45-44.tar.bz2	7/12/2024 6:45 PM	WinRAR	4 KB
test2_2024-07-12_18-45-44.tar.bz2	7/12/2024 6:45 PM	WinRAR	2 KB
test3_2024-07-12_18-45-44.tar.bz2	7/12/2024 6:45 PM	WinRAR	8 KB
test4_2024-07-12_18-45-45.tar.bz2	7/12/2024 6:45 PM	WinRAR	1 KB
test1_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	5 KB
test2_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	2 KB
test3_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	12 KB
test4_2024-07-12_18-41-17.tar.bz2	7/12/2024 6:41 PM	WinRAR	1 KB



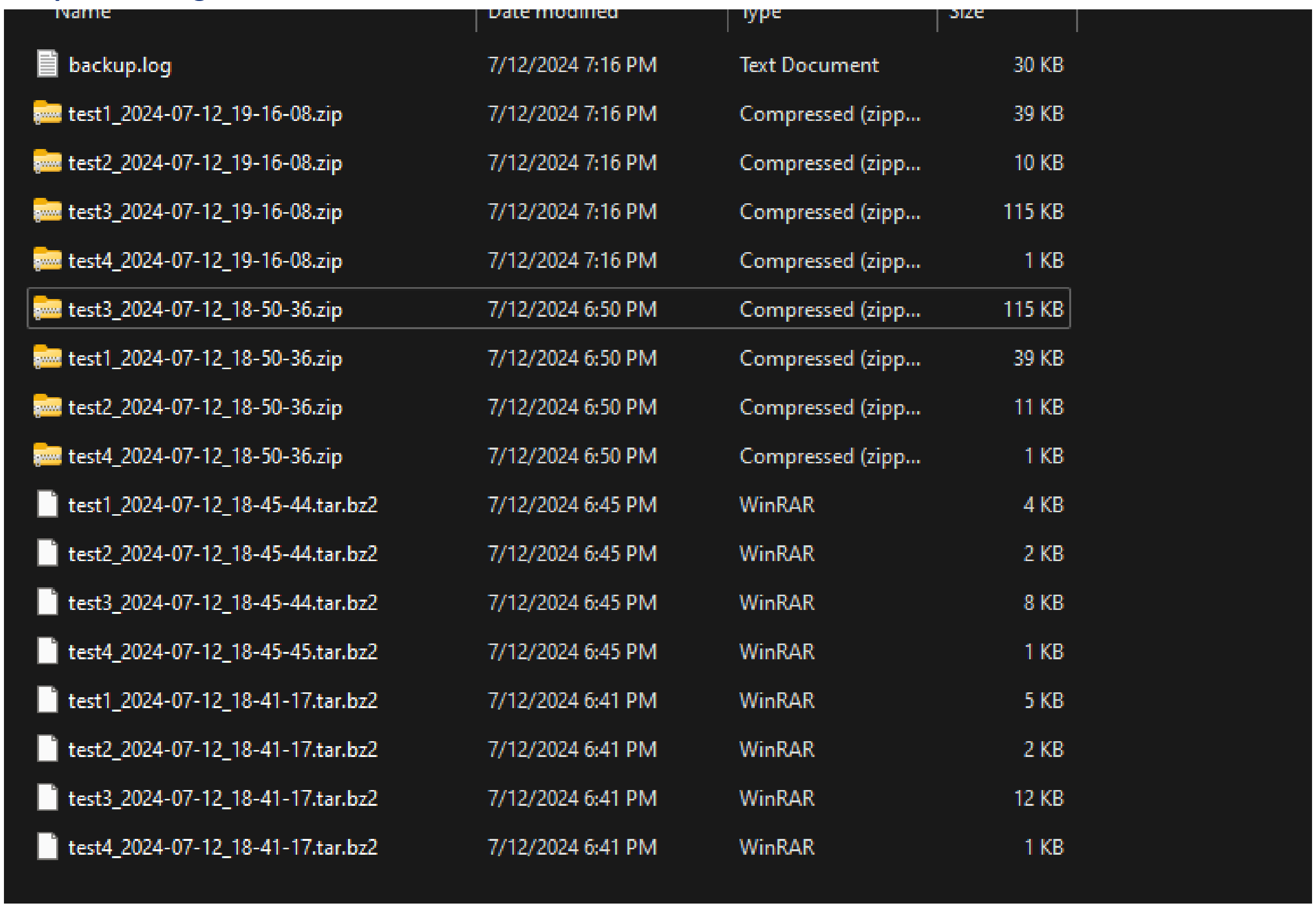
```
Backup started at 2024-07-12 18:50:36
[2024-07-12 18:50:36] Starting backup of test1
[2024-07-12 18:50:36] Starting backup of test2
[2024-07-12 18:50:36] Starting backup of test3
[2024-07-12 18:50:36] Starting backup of test4
[2024-07-12 18:50:38] All backups completed.
```

Zip with compression level: ./backup_script.sh zip ~/backup/output test1 test2 9 test3 test4

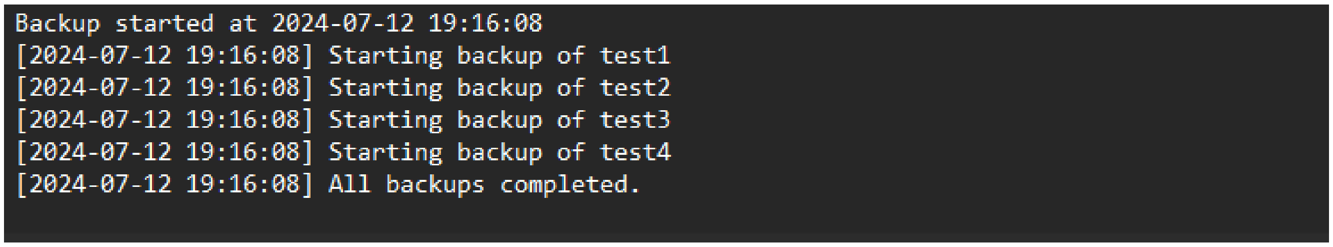


```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh zip ~/backup/output test1 test2 9 test3 test4
Started backup of test1 in the background
Started backup of test2 in the background
Started backup of test3 in the background
Started backup of test4 in the background
All backups completed.
the report:
Directories Before Compression | Size       | Directories After Compression           | Size
--------------------------     | ---------- | ------------------------------          | ----------
test1                          | 13M        | test1_2024-07-12_19-16-08.zip           | 39K
test2                          | 2.1M       | test2_2024-07-12_19-16-08.zip           | 9.1K
test3                          | 39M        | test3_2024-07-12_19-16-08.zip           | 115K
test4                          | 5.3K       | test4_2024-07-12_19-16-08.zip           | 339
```

Output and Log File:

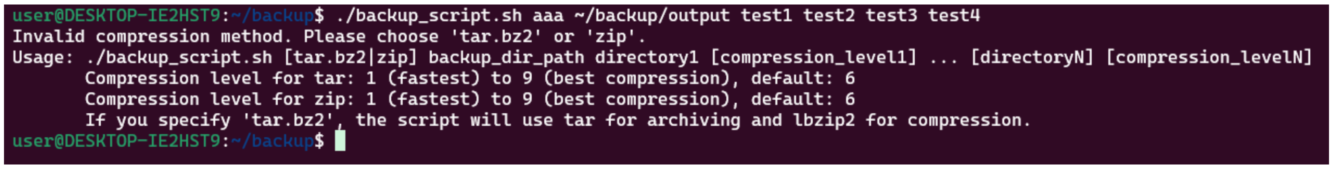


```
Name                              Date modified        Type                 Size
backup.log                        7/12/2024 7:16 PM    Text Document        30 KB
test1_2024-07-12_19-16-08.zip     7/12/2024 7:16 PM    Compressed (zipp...  39 KB
test2_2024-07-12_19-16-08.zip     7/12/2024 7:16 PM    Compressed (zipp...  10 KB
test3_2024-07-12_19-16-08.zip     7/12/2024 7:16 PM    Compressed (zipp...  115 KB
test4_2024-07-12_19-16-08.zip     7/12/2024 7:16 PM    Compressed (zipp...  1 KB
test3_2024-07-12_18-50-36.zip     7/12/2024 6:50 PM    Compressed (zipp...  115 KB
test1_2024-07-12_18-50-36.zip     7/12/2024 6:50 PM    Compressed (zipp...  39 KB
test2_2024-07-12_18-50-36.zip     7/12/2024 6:50 PM    Compressed (zipp...  11 KB
test4_2024-07-12_18-50-36.zip     7/12/2024 6:50 PM    Compressed (zipp...  1 KB
test1_2024-07-12_18-45-44.tar.bz2 7/12/2024 6:45 PM    WinRAR               4 KB
test2_2024-07-12_18-45-44.tar.bz2 7/12/2024 6:45 PM    WinRAR               2 KB
test3_2024-07-12_18-45-44.tar.bz2 7/12/2024 6:45 PM    WinRAR               8 KB
test4_2024-07-12_18-45-45.tar.bz2 7/12/2024 6:45 PM    WinRAR               1 KB
test1_2024-07-12_18-41-17.tar.bz2 7/12/2024 6:41 PM    WinRAR               5 KB
test2_2024-07-12_18-41-17.tar.bz2 7/12/2024 6:41 PM    WinRAR               2 KB
test3_2024-07-12_18-41-17.tar.bz2 7/12/2024 6:41 PM    WinRAR               12 KB
test4_2024-07-12_18-41-17.tar.bz2 7/12/2024 6:41 PM    WinRAR               1 KB
```

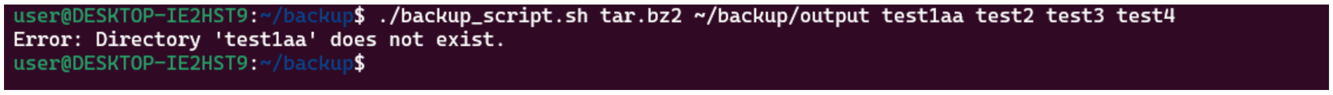


```
Backup started at 2024-07-12 19:16:08
[2024-07-12 19:16:08] Starting backup of test1
[2024-07-12 19:16:08] Starting backup of test2
[2024-07-12 19:16:08] Starting backup of test3
[2024-07-12 19:16:08] Starting backup of test4
[2024-07-12 19:16:08] All backups completed.
```

Validations on inputs:



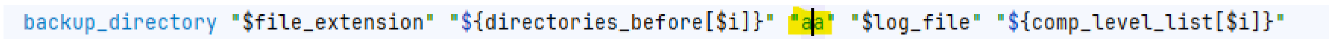
```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh aaa ~/backup/output test1 test2 test3 test4
Invalid compression method. Please choose 'tar.bz2' or 'zip'.
Usage: ./backup_script.sh [tar.bz2|zip] backup_dir_path directory1 [compression_level1] ... [directoryN] [compression_levelN]
       Compression level for tar: 1 (fastest) to 9 (best compression), default: 6
       Compression level for zip: 1 (fastest) to 9 (best compression), default: 6
       If you specify 'tar.bz2', the script will use tar for archiving and lbzip2 for compression.
user@DESKTOP-IE2HST9:~/backup$
```



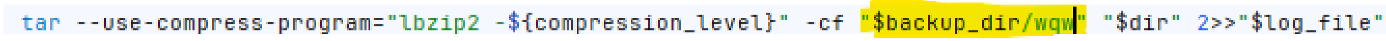
```
user@DESKTOP-IE2HST9:~/backup$ ./backup_script.sh tar.bz2 ~/backup/output test1aa test2 test3 test4
Error: Directory 'test1aa' does not exist.
user@DESKTOP-IE2HST9:~/backup$
```

Simulate error by using a non-existent path: We validate the inputs but for simulating, we modified the script by changing the path to a non-existent directory(back-up directory path).

The 2>>"$log_file" part redirects any error messages generated by the command to the specified log file. This ensures that errors are captured for later review.

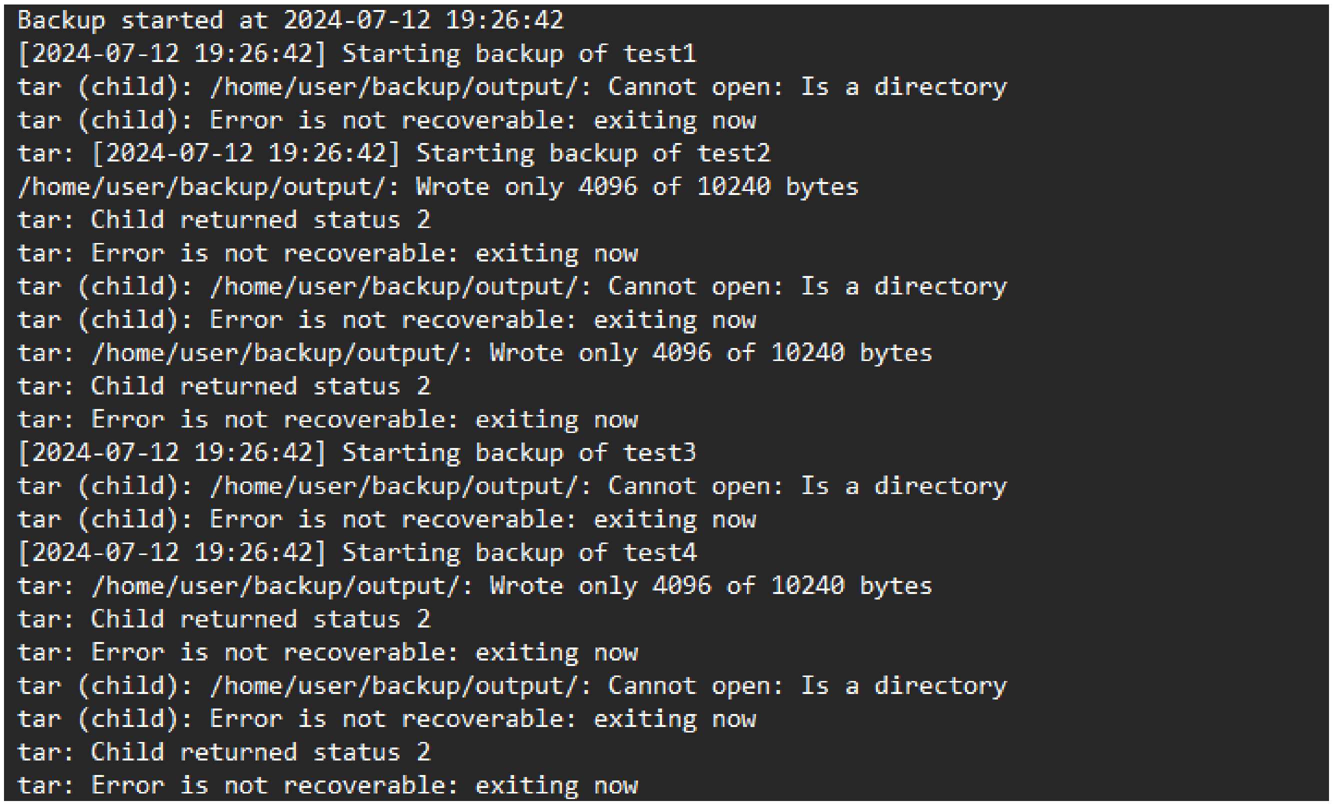


```
backup_directory "$file_extension" "${directories_before[$i]}" "aa" "$log_file" "${comp_level_list[$i]}"
```



```
tar --use-compress-program="lbzip2 -${compression_level}" -cf "$backup_dir/wqw" "$dir" 2>>"$log_file"
```

Log file after simulating:



```
Backup started at 2024-07-12 19:26:42
[2024-07-12 19:26:42] Starting backup of test1
tar (child): /home/user/backup/output/: Cannot open: Is a directory
tar (child): Error is not recoverable: exiting now
tar: [2024-07-12 19:26:42] Starting backup of test2
/home/user/backup/output/: Wrote only 4096 of 10240 bytes
tar: Child returned status 2
tar: Error is not recoverable: exiting now
tar (child): /home/user/backup/output/: Cannot open: Is a directory
tar (child): Error is not recoverable: exiting now
tar: /home/user/backup/output/: Wrote only 4096 of 10240 bytes
tar: Child returned status 2
tar: Error is not recoverable: exiting now
[2024-07-12 19:26:42] Starting backup of test3
tar (child): /home/user/backup/output/: Cannot open: Is a directory
tar (child): Error is not recoverable: exiting now
[2024-07-12 19:26:42] Starting backup of test4
tar: /home/user/backup/output/: Wrote only 4096 of 10240 bytes
tar: Child returned status 2
tar: Error is not recoverable: exiting now
tar (child): /home/user/backup/output/: Cannot open: Is a directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
```

System Health Check Script

Purpose: The health check script monitors and reports the system's health status, focusing on critical aspects such as disk space, memory usage, running services, and recent system updates.

Functionality:

1. **Color Printing Functions:**

```
print_red() {
    printf "\033[1;31m%s\033[0m\n" "$1"
}


print_green() {
    printf "\033[1;32m%s\033[0m\n" "$1"
}
```

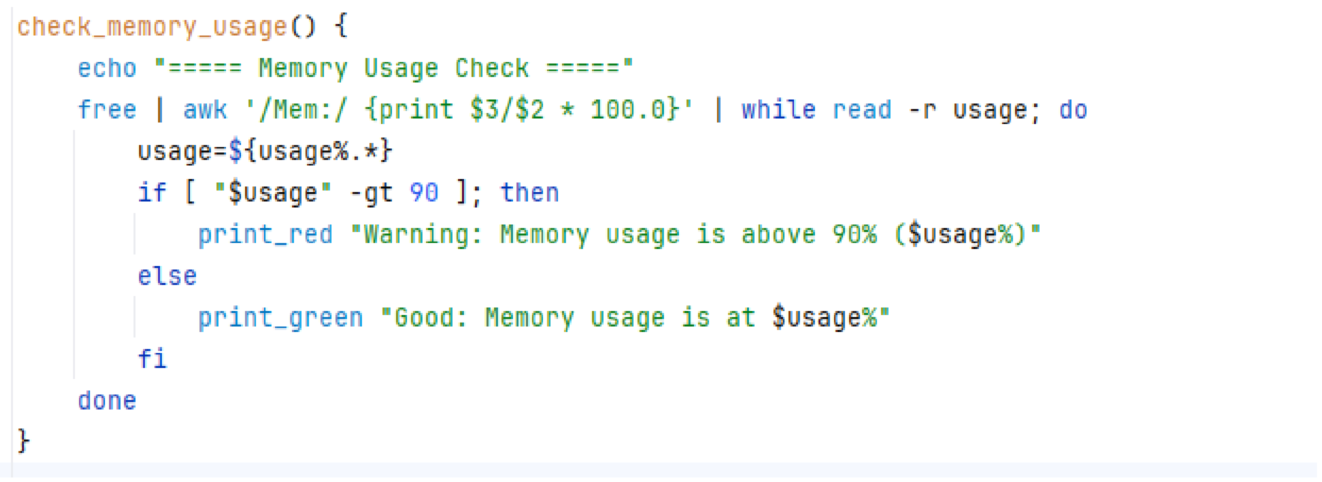
   Explanation: These functions print messages in red or green, respectively. This color-coding improves readability, with red indicating warnings or errors and green indicating normal statuses.

2. **Disk Space Check:**

```
check_disk_space() {
    echo "===== Disk Space Check ====="
    df -h | awk 'NR>1 {print $1, $5}' | while read -r filesystem usage; do
        usage=$(echo "$usage" | sed 's/%//g')
        if [ "$usage" -gt 90 ]; then
            print_red "Warning: Disk space usage on $filesystem is above 90% ($usage%)"
        else
            print_green "Good: Disk space usage on $filesystem is at $usage%"
        fi
    done
}
```

   Explanation: This function checks disk usage for all filesystems. It uses df -h to get human-readable disk usage statistics and awk to filter relevant columns. If any filesystem usage exceeds 90%, it prints a warning in red; otherwise, it prints a normal status in green.

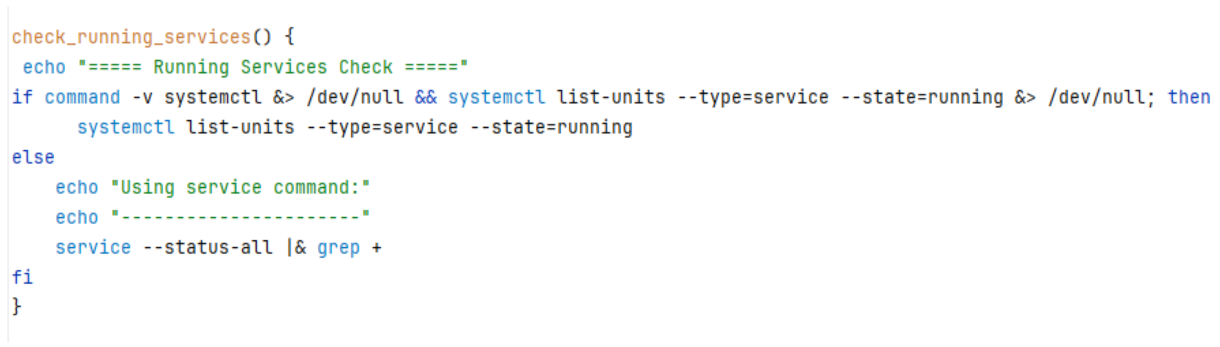
3. **Memory Usage Check:**



```
check_memory_usage() {
    echo "===== Memory Usage Check ====="
    free | awk '/Mem:/ {print $3/$2 * 100.0}' | while read -r usage; do
        usage=${usage%.*}
        if [ "$usage" -gt 90 ]; then
            print_red "Warning: Memory usage is above 90% ($usage%)"
        else
            print_green "Good: Memory usage is at $usage%"
        fi
    done
}
```

Explanation: This function checks memory usage using the free command. It calculates the percentage of used memory and prints a warning if usage exceeds 90%. Otherwise, it prints a normal status.

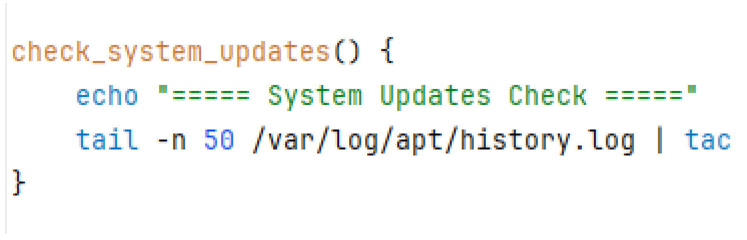
4. **Running Services Check:**



```
check_running_services() {
 echo "===== Running Services Check ====="
if command -v systemctl &> /dev/null && systemctl list-units --type=service --state=running &> /dev/null; then
      systemctl list-units --type=service --state=running
else
    echo "Using service command:"
    echo "---------------------"
    service --status-all |& grep +
fi
}
```

Explanation: This function lists currently running services. It prefers systemctl for systems using systemd. If systemctl is unavailable, it falls back to the service command and filters running services.

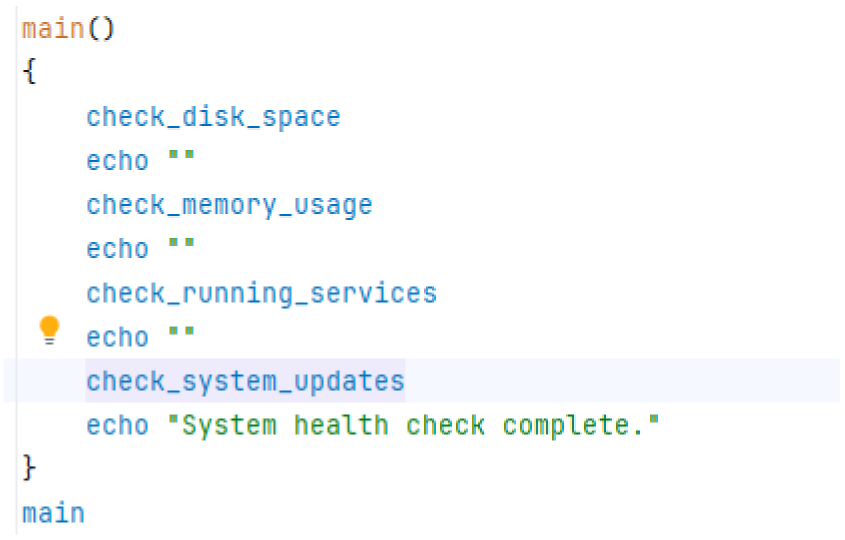
5. **System Updates Check:**



```
check_system_updates() {
    echo "===== System Updates Check ====="
    tail -n 50 /var/log/apt/history.log | tac
}
```

Explanation: This function displays the last 50 lines of the APT history log in reverse order (most recent first) using tail and tac. This provides a quick overview of recent system updates.

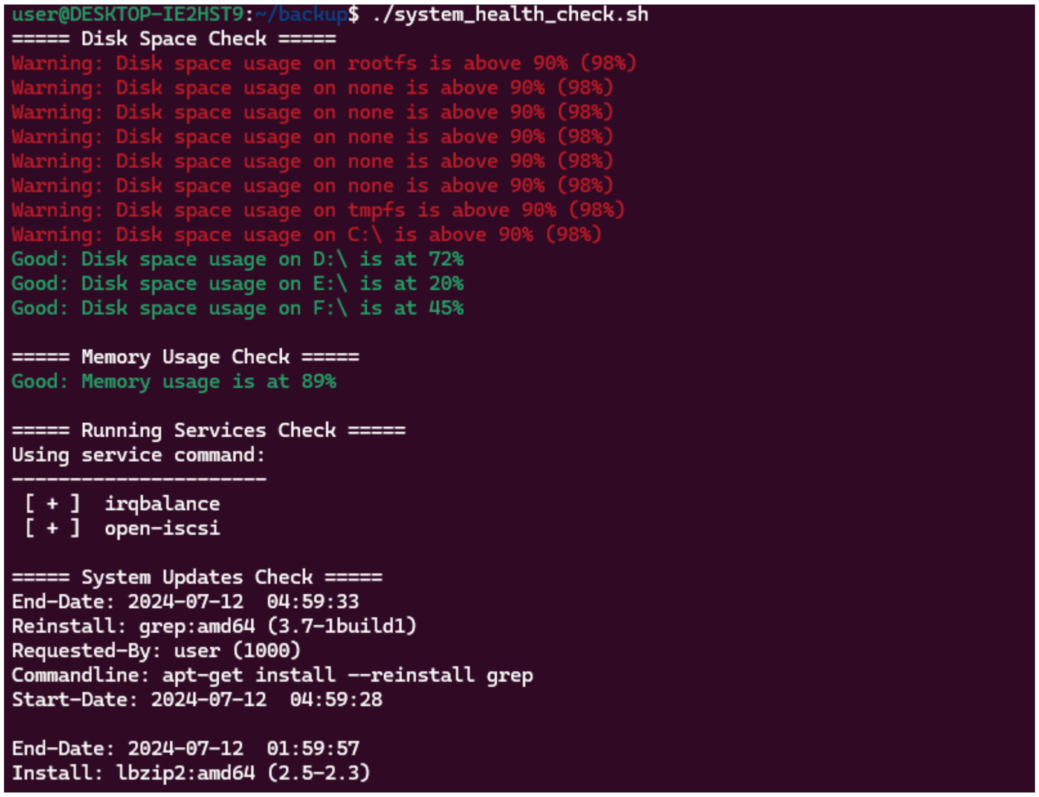
6. **Main Function:**



```
main()
{
    check_disk_space
    echo ""
    check_memory_usage
    echo ""
    check_running_services
    echo ""
    check_system_updates
    echo "System health check complete."
}
main
```

Explanation: The main function orchestrates the execution of all checks sequentially, ensuring comprehensive system health reporting. Each check's output is separated by an empty line for readability. Finally, it prints a message indicating the health check's completion.

Sample Execution and Output: ./system_health_check.sh



```
user@DESKTOP-IE2HST9:~/backup$ ./system_health_check.sh
===== Disk Space Check =====
Warning: Disk space usage on rootfs is above 90% (98%)
Warning: Disk space usage on none is above 90% (98%)
Warning: Disk space usage on none is above 90% (98%)
Warning: Disk space usage on none is above 90% (98%)
Warning: Disk space usage on none is above 90% (98%)
Warning: Disk space usage on none is above 90% (98%)
Warning: Disk space usage on tmpfs is above 90% (98%)
Warning: Disk space usage on C:\ is above 90% (98%)
Good: Disk space usage on D:\ is at 72%
Good: Disk space usage on E:\ is at 20%
Good: Disk space usage on F:\ is at 45%

===== Memory Usage Check =====
Good: Memory usage is at 89%

===== Running Services Check =====
Using service command:
----------------------
 [ + ]  irqbalance
 [ + ]  open-iscsi

===== System Updates Check =====
End-Date: 2024-07-12  04:59:33
Reinstall: grep:amd64 (3.7-1build1)
Requested-By: user (1000)
Commandline: apt-get install --reinstall grep
Start-Date: 2024-07-12  04:59:28

End-Date: 2024-07-12  01:59:57
Install: lbzip2:amd64 (2.5-2.3)
```

Future Features: Pooling Process

In the context of optimizing backup processes, the implementation of process pooling emerges as a strategic enhancement. Process pooling involves maintaining a predefined number of active processes that can be reused for executing tasks, rather than creating and terminating processes repeatedly. This approach aims to mitigate the overhead associated with process management, thereby potentially improving overall system performance and resource utilization.

Key Benefits:

- **Reduced Overhead:** By reusing existing processes, the system avoids the overhead of frequent process creation and termination.
- **Enhanced Efficiency:** Tasks can be executed more swiftly by utilizing available resources effectively.
- **Scalability:** Allows the system to handle a larger volume of concurrent tasks without compromising performance.

Implementation Considerations:

- **Initialization:** Establishing an array to manage the pool of processes, ensuring that the maximum number of concurrent processes is defined based on system capabilities.
- **Task Assignment:** Mechanism to assign tasks to available processes within the pool, ensuring that each task is executed efficiently.
- **Integration:** Integration into backup script to leverage the benefits of process pooling for specific tasks, such as backup operations.