# Scripting Lesson 3: HomeWork

| | |
|---|---|
| **Course** | DevOps Engineering |
| **Topic / Session number** | Scripting / Lesson 3 |
| **Objectives / Goals** | 1. Homework for Scripting Lesson 1<br>a. report.sh<br>b. sizecheck.sh<br>c. diskhog.sh<br>2. Homework for Scripting Lesson 2<br>a. split.sh<br>b. nxn_table.sh<br>c. ten_dirs.sh<br>d. validate_ip.sh |

---

**Objective 1: Homework for Scripting Lesson 1**

---

## report.sh

Write a script called `report.sh` to store server related information in a file. Include meaningful descriptions and data about:

- Disk usage
- Memory usage
- CPU utilization

With human readable sizes (MB/GB/etc.) instead of just bytes.

Create a folder called `reports`. The script should store the reports in this folder. The file name should be the `REPORT_` plus the current datetime in the following format `YEAR_MONTH_DAY_HOUR_MINUTE_SECOND`, e.g.: `REPORT_2025_03_09_14_59_44`

You will need to use the `date` command with a format specifier to get the current datetime in this format. The syntax is `date '+YOURFORMAT'`. You can find the different format specifiers by accessing the man page of date (`man date`).

Example output :

```
DISK USAGE:
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/root        7034376 2218932   4799060  32% /
tmpfs             490192       0    490192   0% /dev/shm
tmpfs             196080     892    195188   1% /run
tmpfs               5120       0      5120   0% /run/lock
/dev/xvda16       901520   77044    761348  10% /boot
/dev/xvda15       106832    6246    100586   6% /boot/efi
tmpfs              98036      12     98024   1% /run/user/1000

```

```
MEMORY USAGE:
               total        used        free      shared  buff/cache   available
Mem:           957Mi       545Mi        70Mi       904Ki       538Mi       411Mi
Swap:             0B          0B          0B

CPU UTILIZATION:
top - 15:09:28 up  3:17,  2 users,  load average: 0.00, 0.02, 0.00
Tasks: 112 total,   2 running, 110 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :    957.4 total,     70.1 free,    545.9 used,    538.4 buff/cache

```

**Hint:**

- Feel free to use a format string composer tool for creating the correct date format.
- To run the `top` command non-interactively, you could use `top -b -n 1`.

## sizecheck.sh

Write a script `sizecheck.sh` that outputs the size of a directory that was passed in as an argument:
Ex: `./sizecheck.sh /tmp` will show the size of `/tmp` directory (here `/tmp` is an argument for `sizecheck.sh` script).
Show the size of the current directory if no argument was passed to the script.

`./sizecheck.sh` should print the current directory size.

Try to modify `sizecheck.sh` so that it can accept more than 1 argument.
`./sizecheck.sh /tmp /bin /etc` should print the sizes of 3 directories that were passed in as arguments.

Example output (with multiple arguments):

```
$ sudo ./sizecheck.sh /usr /var /etc
1.5G    /usr
498M    /var
6.3M    /etc
```

**Hint:**

- If necessary, you may use the `-n` test operator to ensure that the first argument (`$1`) exists. Alternatively you can use the default parameter operator.
- Do you still remember the special variable that contains all the arguments in a list?

## diskhog.sh

Create a shell script called `diskhog.sh` that lists the **five largest items** (files or directories) directly in the current directory (excluding subdirectories and files inside them) in **decreasing order of size**. Include hidden files and directories too (those starting with a `.`).

The script takes one argument, the path to the directory where the largest files need to be found. If no argument provided, it can default to the current directory (`.`). You should output the sizes in a human readable format.

Example output:

```
$ sudo ~/scripts/diskhog.sh /usr
1.5G    /usr
890M    /usr/lib
266M    /usr/share
155M    /usr/src
112M    /usr/bin
```

If the script is run from an empty directory, make sure it prints nothing at all (and not an error message). Ideally, try to exclude the current working directory (`.`) from the list.

Check out the man pages for `du(1)`, `sort(1)`, and `head(1)` (or `tail(1)`).

> 💡 **Hint:**
>
> - You may use the `-n` test operator to ensure that the first argument (`$1`) exists. Alternatively you can use the default parameter operator.
> - Since we have a chain of commands on the same data use may make use of the pipe (`|`) operator.
> - Sort based on the values of the first column (the size values).
> - Sorting in descending order can be achieved by taking the negative value of the column (e.g. given **1, 2, 3** if we sort by their negative values (**-3, -2, -1**) it will be sorted as **3, 2, 1**).
> - Make sure you **DO NOT** sort based on textual value. Either sort based on "human readable numbers" or use a fixed block size (e.g. 1MB) and sort based on numeric value.
> - You could use a tool like `AWK` for filtering out the current directory from the list (just make sure you filter it out before you restrict the results to 5 items).
> - Due to different implementation strategies, you may get an output with a different format than the provided example. This is not an issue as long as the script does its job.

Create an alias under the name `dh` for your newly created script in your `~/.bashrc` file. Don't forget to use the `source ~/.bashrc` command to refresh your environment after you finished editing the `.bashrc` file.

Test the newly defined `dh` command with different paths and see if it works as expected. Look out for edge cases such as empty directories.

---

### ⭐ Objective 2: Homework for Scripting Lesson 2

---

## split.sh

Create a bash script that takes a string argument and prints the words in the sentence line by line.

```
$ ./split.sh "Hi People. How Are You?"
Hi
People.
How
Are
You?
```

## nxn_table.sh

Write a script that takes a number as an argument `N`, and generates an N×N table populated with the first N*N numbers.

Example:

`./nxn_table.sh 3` - prints a 3×3 table with 9 numbers starting from 1.

```
$ ./nxn_table.sh 3
1 2 3
4 5 6
7 8 9
```

> 💡 **Hint:**

- Use `echo -n` to print text without a newline
- Don't forget to add a newline after every `N` number

## ten_dirs.sh

Write a script `ten_dirs.sh` that does following:

1. Creates a directory called `ten`.
2. Makes ten subdirectories `ten/dir01`, `ten/dir02`,... through `ten/dir10`.
   (Note the formatting `dir01`, `dir02`, etc., **NOT** `dir1`, `dir2`, etc.)
3. In each subdirectory, make four files:
   `file1.txt` `file2.txt` `file3.txt` `file4.txt`
   Such that:
   - `file1.txt` has one line consisting of the digit 1
   - `file2.txt` has two lines, each consisting of the digit 2
   - `file3.txt` has three lines, each containing the digit 3
   - `file4.txt` has four lines, each containing the digit 4

Expected file tree:

```
ten
├── dir01
│   ├── file1.txt
│   ├── file2.txt
│   ├── file3.txt
│   └── file4.txt
├── dir02
│   ├── file1.txt
│   ├── file2.txt
│   ├── file3.txt
│   └── file4.txt
```

```
└── dir10
    ├── file1.txt
    ├── file2.txt
    ├── file3.txt
    └── file4.txt
```

💡 **Hint:**

- You could use a command like `seq` for generating the numbers. Try to figure out how to make the padding universal (add zero padding if the sequence includes numbers that have two or more digits).
- You are doing a repetitive action → you need a loop. You are doing a repetitive action within another repetitive action (creating directories and files inside those directories) → you need to nest loops.
- Since you will be running the script a lot, you might want to remove the `ten` directory before recreating it in the beginning of the script, during the course of development, so that you don't have to manually do that every time you run the script. Just don't forget to remove the part that removes the `ten` folder once the script is working.

## validate_ip.sh

Create a bash script that verifies whether the given IP a correct IPv4 address.

Requirements for validity:

- There should be exactly 4 octets separated by a dot
- Octet should be only numbers between 0 and 255 (inclusive)
- Octets should not be empty

```
$ ./validate_ip.sh 10.0.0.1
The provided IP address '10.0.0.1' is valid
$ ./validate_ip.sh 172.0.0.264
Invalid IP address: '172.0.0.264'
$ ./validate_ip.sh a.sdddd
Invalid IP address: 'a.sdddd'
```

**Recommended steps:**

- Divide the string into octets
- Check if the number of octets is correct
- Validate the octets
  - Ensure that they have numeric value
  - Check if it falls between 0 and 255
- If at any point a condition evaluates as false, print that the IP is invalid and exit the script.
- Once all the octets have been successfully validated print that the IP is valid.

**Hint:**

- There are many ways a string can be split in bash, a common way to achieve this is using Internal Field Separator (IFS)
- You can use the `-z` test operator to check whether a string is empty or the `-n` operator to ensure that it isn't empty.

Practice correct typing using all fingers:

- Typing Lessons | Short Paragraphs - Typing.com
- Monkeytype | A minimalistic, customizable typing test