Exercises in Secure Mobile Systems

FH Hagenberg, WS 2022
FH-Prof. DI Dr. Erik Sonnleitner

Exercise 3: Shell Scripting

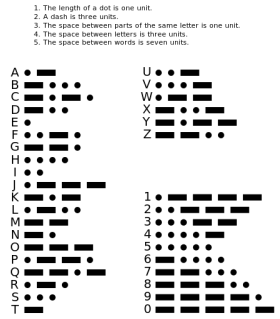
3.1) Automatic Image Downloader

Morse code (see **`https://en.wikipedia.org/wiki/Morse_code`**) has revolutionized tele-communication in the 19th century, since for the first time in human history, it was possible to transmit arbitrary information over extremely long distances – first over land-wires and subsequently via radio transmissions.

Suppose you are a morse-code enthusiast (dit-dah-dit) and you happily discover that the WikiMedia foundation has freely usable images of dozens of historic morse keys and paddles at **`https://commons.wikimedia.org/wiki/Category:Straight_keys`**.

Write a shell script which downloads the corresponding HTML page (using **`wget`** or **`curl`**), parses the HTML code in order to extract the image URLs, and then iteratively download and store all images in a separate folder (use high-resolution variants if possible).

Note: Parsing the source and extracting image URLs can easily be done in a single piped command-line.



3.2) System Integrity Assurance

Keeping the system not only intact, but free from malware, root-kits and other malicious modifications is important. We thus like to create a shell script **`integrity.sh`**, which scans a given directory (e.g. **`/home/eso`** or **`/boot/`**), recursively iterates all its file-system objects and creates a SHA-512 hash sum of each file.

Building an integrity map

The integrity map is created with the **`-b`** option (build). The file-names (including full path) and their respective hash value are stored in a data-file given as argument. All items in an existing map can be rebuilt with **`-r`**.

Checking an integrity map

A consistency check can be run via the **`-c`** (check) option, which checks all files in a given integrity map and prints warnings for all files which have been modified.

- Create map: **`./integrity.sh -b /boot /tmp/integrity-boot.map`**
- Check existing map: **`./integrity.sh -c /tmp/integrity-boot.map`**
- Rebuild existing map: **`./integrity.sh -r /tmp/integrity-boot.map`**
- Do proper error checking (e.g. map already exists on **`-b`**, map doesn't exist on **`-c`** and **`-r`**, no permissions to read/write file)

3.3) Dice Rolls

Many games rely on rolling dice, especially particular tabletop role-playing games like *Dungeons & Dragons*. Hereby, simple 6-sided dice are not enough – there are several different kinds of dice, including e.g. 8-sided or 20-sided ones.

Requirements

Implement a shell script which rolls dice according to the following guidelines:

- Arguments define how many rolls on which types of dice should be made. Roll count and the number of sides are divided by a hyphen.
- e.g. **`roll.sh 1-6`** prints the output of one roll of a standard 6-sided die
- e.g. **`roll.sh 6 3-8 2-20`** prints the result of one single 6-sided die roll, 3 8-sided die rolls and 2 20-sided die rolls
- Can hence have arbitrarily many arguments of rolls
- Output specifies result of each roll, and total sum of roll values per dice.

3.3) Dice Rolls

Simple, single 6-sided die roll (↓).

```
$ roll.sh 2-6
rolling 6-sided die 2x:
  roll #1 = 6
  roll #1 = 2
Rolled 6-sided die 2x, which adds up to 8.
```

Note: Can also use **printf** command instead of **echo**

Multiple dice rolled with different number of sides and output of the total sum of values (→).

Example dice rolls

```
$ roll.sh 1-6 2-20 1-8
rolling 6-sided die 1x:
  roll #1 = 5
Rolled 6-sided die 1x, which adds up to 5.

rolling 20-sided die 2x:
  roll #1 = 13
  roll #2 = 3
Rolled 20-sided die 2x, which adds up to 16.

rolling 8-sided die 1x:
  roll #1 = 1
Rolled 8-sided die 1x, which adds up to 1.

In total, all of those rolls add up to 22.
```

3.4) Creating Thumbnails

As a morse-code enthusiast, you decide to writ your own web-page about the joy of morse. Since you already downloaded the morse key images in the first assignment and due to the fact that they are licensed under the *Creative Commons* license, you can easily use them on your web-page.

Instead of transferring all images in full resolution, you need to create thumbnails for providing an image overview. Write a shell script **`thumbs.sh`** which takes image files as argument and automatically creates thumbnails of a given size. Use the tools **`identify`** and **`convert`** from the well-known package **`imagemagick`** to do so. Script requirements:

- Script usage: **`./thumbs.sh (-e|-p) [-d] dimension image [...]`**
- Dimension d is given as single-number string e.g. **`300px`**
- **`-e`** creates *exact* thumbnails, i.e. does not keep original image proportions and thus stretches image to $d \times d$
- **`-p`** creates *proportional* thumbnails, i.e. scales image proportionally, with the longer dimension (x or y) being scaled to given dimension d
- **`-s`** creates *square* thumbnail without stretching, i.e. scales to shorter dimension (x or y) to d and the crops rectangle $d \times d$
- **`-d`** additionally deletes EXIF meta-data from thumbnails for web publishing.

3.4) Creating Thumbnails

Original Image (600x400)

Proportional thumbnail 300x300 cutout **`-s 300`** (scaled to 450x300 & cropped 300x300 area)

Proportional thumbnail scaled to 300x200 with **`-p 300`**

Non-proportionally scaled thumbnail 300x300 with **`-e 300`** (stretched)

Bonus: Continuous Monitoring

Implement a script for continuously supervising, monitoring and autonomously restarting arbitrary system processes called **`monitor.sh`**. This script takes two mandatory arguments: the first one being either a program name or a PID, the second one being an integer value N. The script has to continuously check every N seconds if the given process is still running. If so, print a message to **`stdout`** (unless the option **`-quiet`** was given). If not, print a message to **`stderr`** and ask the user if he/she wants to restart the program.

Requirements:

- If program name is given, the script has to determine the PID itself
- If program name is given and there are multiple processes with the same name running, list all corresponding PIDs on terminal and force user to choose exactly one
- If the user chooses to restart the program, it should be started in background (detached from the shell) and the script continues to monitor it
- All arguments of the originally running program must be maintained when restarted
- If restarting fails for some reason, exit with error.
- Script should accept an optional argument **`-respawn`**, which automatically performs the restart if program has exited, and informs the user if that happened