Lab 03: Basic Scripting

Name: João Pedro Marçal Storino
Course: Everything from the Command Line
Professor: Luis Gustavo Nardin
Date: October 17, 2024

1 Warming Up

For the first part of the Laboratory of Basic Scripting, in a way to get used to **Bash Language** we are asked to execute some scripts to practice.

1.1 Script 01

Create a script to write the number of parameters it gets on the command line as well as a list of these parameters if any, one per line:

```
#!/bin/bash

echo "Number of parameters: $#"

if [ $# -gt 0 ]; then
  echo "List of parameters:"
  for param in "$@"; do
    echo "$param"
  done
else
  echo "No parameters provided."
fi
```

And the result:

```
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_01.sh 1 2 3 4 5
Number of parameters: 5
List of parameters:
1
2
3
4
5
```

1.2 Script 02

Create a script to write the sequence of the first 10 integers starting at 1. Its return status is 0.

```
#!/bin/bash
# Script to print the first 10 integers
for i in {1..10}; do
  echo "$i"
done
exit 0
```

And the result:

```
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_02.sh
1
2
3
4
5
6
7
8
9
10
```

1.3 Script 03

Update your script to make it write a sequence of integers between min and max given on the command line (min is the first argument, max the second one).

```bash
#!/bin/bash
# Prints integers between a given range (min and max)

if [ $# -ne 2 ]; then
    echo "Please provide exactly two arguments (min and max)."
    exit 1
fi

min=$1
max=$2

if [ "$min" -ge "$max" ]; then
    echo "Error: min must be less than max."
    exit 2
fi

for i in $(seq "$min" "$max"); do
    echo "$i"
done
```

And the result:

```
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_03.sh
Please provide exactly two arguments (min and max).
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_03.sh 5 3
Error: min must be less than max.
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_03.sh 3 10
3
4
5
6
7
8
9
10
```

1.4 Script 04

Change your script to make it more robust:

a. If only one parameter is given, the script outputs the sequence of integers from 1 to this value and the error code is 0 (meaning this is a expected behaviour).

b. If no parameter is provided, the script simply outputs a sequence of integers from 1 to 10, but the error code is 1 instead of 0.

c. If min is not lesser than max, an error message is issued, no sequence is displayed (obviously) and the error core is 2.

d. Add to your script a last check on its parameters to ensure they all are actual integers.

```bash
#!/bin/bash
# Prints a sequence of integers based on the number of arguments
if ! [[ "$min" =~ ^-?[0-9]+$ ]] || ! [[ "$max" =~ ^-?[0-9]+$ ]] || ! [[
    "$step" =~ ^-?[0-9]+$ ]]; then
    echo "Error: All arguments must be integers."
    exit 3
fi
if [ $# -eq 0 ]; then
    echo "No arguments provided. Outputting default sequence (1 to 10)."
    seq 1 10
    exit 1
elif [ $# -eq 1 ]; then
    max=$1
    seq 1 "$max"
    exit 0
elif [ $# -eq 2 ]; then
    min=$1
    max=$2
    if [ "$min" -ge "$max" ]; then
        echo "Error: min must be less than max."
        exit 2
    fi
    seq "$min" "$max"
    exit 0
else
    echo "Too many arguments."
    exit 1
fi
```

And the result:

```
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_04.sh
No arguments provided. Outputting default sequence (1 to 10).
1
2
3
4
5
6
7
8
9
10
```

```
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_04.sh 5
1
2
3
4
5
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_04.sh 5 3
Error: min must be less than max.
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_04.sh 5.1
Error: All arguments must be integers.
```

1.5 Script 05

Update your script to accept a third optional parameter, the step between to integers. For example, if the step is 3, the sequence from 1 to 10 would be 1 4 7 10.

```
#!/bin/bash
# Prints a sequence with optional step value.

min=1
max=10
step=1

# Integer verification
is_integer() {
    [[ "$1" =~ ^-?[0-9]+$ ]]
}

# No argument provided
if [ $# -eq 0 ]; then
    echo "No arguments provided. Default sequence  (1 - 10)"
    seq "$min" "$max"
    exit 1

# One argument provided
elif [ $# -eq 1 ]; then
    max=$1
    if is_integer "$max"; then
        seq "$min" "$max"
        exit 0
    else
        echo "Error: Argument must be an integer."
        exit 3
    fi

# Two arguments provided
elif [ $# -eq 2 ]; then
    min=$1
    max=$2
    if is_integer "$min" && is_integer "$max"; then
        if [ "$min" -ge "$max" ]; then
            echo "Error: min must be less than max."
            exit 2
        fi
        seq "$min" "$max"
        exit 0
    else
```

```
        echo "Error: Both arguments must be integers."
        exit 3
    fi

# Three arguments provided (Step)
elif [ $# -eq 3 ]; then
    min=$1
    max=$2
    step=$3
    if is_integer "$min" && is_integer "$max" && is_integer "$step"; then
        if  [ "$min" -ge "$max" ]; then
            echo "Erro: min must be less than max."
            exit 2
        fi
        seq "$min" "$step" "$max"
        exit 0
    else
        echo "Error: All arguments must be integers."
        exit 3
    fi
# More than three arguments
else
    echo "Too many arguments. Expected at most 3."
    exit 1
fi
```

And the result:

```
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_05.sh
No arguments provided. Outputting default sequence (1 to 10).
1
2
3
4
5
6
7
8
9
10
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_05.sh 2 5
2
3
4
5
(base) Laptop-de-Joao:Lab 03 marcalstorino$ ./warmup_05.sh 2 10 2
2
4
6
8
10
```

2 Task Automation

In this section, we have to execute the Automation of a process that we've seen in the last Laboratory. For this example, we've used the **Dracula.txt**, same as in the past Laboratory.

For so:

1. Based on what you ve learned from last lab (text editing and processing), write a script to fetch a text given its number in the Gutenberg ebook database, remove its header and footer and save it to a file named after its title. The name must not contain any space though, so replace them with the underscore character.
2. Update your script to output the number of sentences in the text and the number of words once the pruning from the previous question is done.
3. Make a last addition to your script to create a histogram of the words in it (a representation of the number of occurrences of each word in the text).

The script, with all of the exigences:

```bash
#!/bin/bash
# Function to replace spaces with underscores and remove non-alphanumeric characters
sanitize_filename() {
    echo "$1" | tr ' ' '_' | tr -cd '[:alnum:]_'
}
# Check if a book number is provided
if [ $# -ne 1 ]; then
    echo "Usage: $0 <Gutenberg ebook number>"
    exit 1
fi
# Fetch the ebook content from Gutenberg, following redirects with -L
BOOK_NUM=$1
URL="https://www.gutenberg.org/ebooks/${BOOK_NUM}.txt.utf-8"
TEXT=$(curl -sL "$URL")  # Use -L to follow redirects
# Check if the download was successful
if [ $? -ne 0 ] || [ -z "$TEXT" ]; then
    echo "Failed to fetch the ebook. Please check the book number."
    exit 1
fi

# Display the first few lines of the text for debugging purposes
echo "First few lines of the fetched text:"
echo "$TEXT" | head -n 20  # Display the first 20 lines to check the structure

# Extract the title using multiple patterns (for flexibility)
TITLE=$(echo "$TEXT" | grep -m 1 -E '^Title: ' | sed 's/^Title: //')

# If no title is found, use a default filename
if [ -z "$TITLE" ]; then
    echo "Failed to find the title in the ebook."
    TITLE="gutenberg_book_${BOOK_NUM}"
fi

# Sanitize the title to use as the filename
FILENAME=$(sanitize_filename "$TITLE")
FILENAME="${FILENAME}.txt"

# Customizable text boundary markers
START_MARKER="START OF THIS PROJECT GUTENBERG EBOOK"
END_MARKER="END OF THIS PROJECT GUTENBERG EBOOK"
```

```bash

# Find the lines that indicate the start and end of the main content
START_LINE=$(echo "$TEXT" | grep -n "$START_MARKER" | cut -d: -f1)
END_LINE=$(echo "$TEXT" | grep -n "$END_MARKER" | cut -d: -f1)

if [ -z "$START_LINE" ] || [ -z "$END_LINE" ]; then
    echo "Failed to find the text boundaries. Saving the full text."
    echo "$TEXT" > "$FILENAME"
else
    # Prune the text and save it to a file
    echo "$TEXT" | sed -n "${START_LINE},${END_LINE}p" > "$FILENAME"
fi

echo "Text saved to $FILENAME."

# Count the number of sentences and words in the pruned text
NUM_SENTENCES=$(grep -o '\.' "$FILENAME" | wc -l)
NUM_WORDS=$(wc -w < "$FILENAME")
echo "Number of sentences: $NUM_SENTENCES"
echo "Number of words: $NUM_WORDS"

# Create a histogram of the word occurrences
echo "Generating word histogram..."
grep -oE '\w+' "$FILENAME" | tr '[:upper:]' '[:lower:]' | sort | uniq -c |
    sort -nr > "${FILENAME%.txt}_histogram.txt"
echo "Word histogram saved to ${FILENAME%.txt}_histogram.txt."
```

And the output:

```
    (base) Laptop-de-Joao:Task Automation marcalstorino$ ./task_auto.sh 345
First few lines of the fetched text:
The Project Gutenberg eBook of Dracula

This ebook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this ebook or online
at www.gutenberg.org. If you are not located in the United States,
you will have to check the laws of the country where you are located
before using this eBook.

Title: Dracula

Author: Bram Stoker

Release date: October 1, 1995 [eBook #345]
                Most recently updated: November 12, 2023

Language: English

Credits: Chuck Greif and the Online Distributed Proofreading Team
Failed to find the text boundaries. Saving the full text.
Text saved to Dracula.txt.
Number of sentences:     8534
Number of words:   164351
Generating word histogram...
Word histogram saved to Dracula_histogram.txt.
```

The **Dracula_histogram.txt** is attached to the .zip archive of this lesson.

3 Analysing Scripts

For this part of the lab, we have to Analyse the script of "Game of Life". In this script, we have to make the following steps:

1. Identify all functions in the scripts, giving their purpose and their arguments. Give the role of each argument and indicate if it is mandatory or not (support your claim).

   (a) `Display()`:

   i. **Propose**: Displays the current state of the cell grid. Counts the number of live cells and prints the grid in a readable format, replacing dead cells with spaces.

   ii. **Arguments**: Receives a string representing the current state of the grid (live and dead cells).

   iii. **Mandatory?**: Yes, the argument is necessary to correctly display the grid.

   (b) `IsValid()`:

   i. **Purpose**: Checks if a cell's coordinate is valid (if it is within the grid boundaries).

   ii. **Arguments**:

   a. The first argument is the index of the cell.

   b. The second argument is the row where the cell is located.

   iii. **Mandatory?**: Yes, both arguments are required to verify the cell's validity.

   (c) `IsAlive()`:

   i. **Propose**: Checks if the cell is alive based on the number of live neighbors and the current state of the cell.

   ii. **Arguments**:

   a. The first argument is the grid's current state.

   b. The second argument is the index of the cell.

   c. The third argument is the current state of the cell (alive or dead).

   iii. **Mandatory?**: Yes, all arguments are needed to determine the cell's state.

   (d) `GetCount()`:

   i. **Propose**: Counts the number of live cells in the neighborhood of a specific cell.

   ii. **Arguments**:

   a. The first argument is the grid's state.

   b. The second argument is the index of the cell.

   iii. **Mandatory?**: Yes, both arguments are necessary to correctly count the live neighbors.

   (e) `next_gen()`:

   i. **Propose**: Updates the state of the grid to the next generation, determining the status of each cell based on the game's rules.

   ii. **Arguments**: Receives a string representing the current state of the grid.

   iii. **Mandatory?**: Yes, the argument is necessary to calculate the next generation.

2. What is the let command for? Where did you find the answer?

   The answer for the purpose of the let command can be found in the official Bash documentation, which describes its use for mathematical operations. The let command is used to perform integer arithmetic operations in Bash. In the context of the script, it is used for index calculations and cell counting. For example, the line let cells = $ROWS * COLS$ calculates the total number of cells in the grid by multiplying the number of rows by the number of columns.

3. Answer the question in line 285.

   In line 285, the commented line let "generation += 1" was originally used to increment the generation counter variable. The reason it is commented out is likely because the increment is being handled automatically elsewhere in the code, making this line redundant.

4. Why are some variables used inside quotes (eg. lines 156, 169, 172, ...) and some without quotes (eg. lines 169, 179, 221, ...)? Try to find a global rule, don't try to explain each line.

   With Quotes: Using quotes around variables ensures that the value is treated as a single unit, even if it contains spaces or special characters. This is important when dealing with strings or expressions that may contain multiple words, as in the line echo -n "$cell".

   Without Quotes: Variables without quotes are interpreted as separate words or arguments, which can be useful when iterating over individual elements in an array or processing numeric values. For example, let "i += 1" is an arithmetic operation and does not require quotes.

5. Create a "gen0" file to seed this script. You should guess the format and syntax of this file from the reading procedure of this script.

   To create the gen0 file, which serves as the initial cell grid for the game, you can infer the syntax by reading the initial function in the script. The file should contain a representation of the grid, where live cells are represented by dots (.) and dead cells by underscores (_), with no line breaks. The file should have a grid layout corresponding to the number of rows and columns defined in the script (in this case, 10x10). For example:

```
        _ _ _ _ . _ _ _ _ _
        _ _ _ . . . _ _ _ _
        _ _ _ _ . _ _ _ _ _
        _ _ _ _ . _ _ _ _ _
        _ _ _ _ . _ _ _ _ _
        _ _ _ _ . _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _
        _ _ _ _ _ _ _ _ _ _
```

4 Make the Script Better

In the last part of the Laboratory, we have to make the script better with some alterations in the major script. For so:

1. The grid in this script has a "boundary problem". Change the script to have the grid wrap around, so that the left and right sides will "touch", as will the top and bottom.

```
GetCount () {
  local cell_number=$2
  local array
  local top
  local center
  local bottom
  local r
  local row
  local i
  local t_top
  local t_cen
  local t_bot
  local count=0
  local ROW_NHBD=3

  array=( `echo "$1"` )

  # Compute top, center, bottom with wrap-around logic
  let "top = ($cell_number - $COLS - 1 + $cells) % $cells"
  let "center = ($cell_number - 1 + $cells) % $cells"
  let "bottom = ($cell_number + $COLS - 1 + $cells) % $cells"
  let "r = $cell_number / $COLS"

  for ((i=0; i<$ROW_NHBD; i++))
  do
    let "t_top = ($top + $i) % $cells"
    let "t_cen = ($center + $i) % $cells"
    let "t_bot = ($bottom + $i) % $cells"

    IsValid $t_cen $r
    if [ $? -eq "$TRUE" ]
    then
      if [ ${array[$t_cen]} = "$ALIVE1" ]
      then
        let "count += 1"
      fi
    fi

    IsValid $t_top $(($r-1))
    if [ $? -eq "$TRUE" ]
    then
      if [ ${array[$t_top]} = "$ALIVE1" ]
      then
        let "count += 1"
      fi
    fi

    IsValid $t_bot $(($r+1))
    if [ $? -eq "$TRUE" ]
    then
      if [ ${array[$t_bot]} = "$ALIVE1" ]
      then
        let "count += 1"
      fi
    fi
  done

  if [ ${array[$cell_number]} = "$ALIVE1" ]
  then
    let "count -= 1"
  fi

  return $count
}
```

2. Modify this script so that it can determine the grid size from the ”gen0” file, and set any variables necessary for the script to run.

```
# Load the grid from the gen0 file
initial=( `cat "$startfile" | sed -e '/#/d' | tr -d '\n' | sed -e 's/\./\. /g' -e 's/_/_ /g'` )

# Dynamically determine ROWS and COLS based on the gen0 file
ROWS=$(cat "$startfile" | grep -v '#' | head -n 1 | wc -m)  # Count columns
ROWS=$((ROWS - 1))  # Remove trailing newline from count
COLS=$(cat "$startfile" | grep -v '#' | wc -l)              # Count rows

let "cells = $ROWS * $COLS"  # Calculate the number of cells
```

3. Create a new ”gen0” file to test your new script.

```
    _ . _ _ _ . . _ _ _
    . . . _ _ . _ _ . _
    _ _ _ . _ . _ . _ .
    . _ . _ . _ . . _ _
    _ . _ _ _ . . _ . .
```

4. Identify the sections in the script where redundant code may be optimized.

```
wrap_index () {
    # Function to wrap indices around the grid size
    local index=$1
    let "wrapped_index = ($index + $cells) % $cells"
    echo $wrapped_index
    }

    # Example use:
    wrapped_top=$(wrap_index $top)
    wrapped_bottom=$(wrap_index $bottom)
```