Bash Organizer
CS104 Project

Arihant Vashista, 22b0958

June 10, 2023

Contents

1 Introduction **2**

2 Project Overview **2**

3 Usage **2**

4 Functions **3**
- 4.1 `show_help` . . . 3
- 4.2 `get_ext` . . . 3
- 4.3 `get_date` . . . 3
- 4.4 `get_available_filename` . . . 4
- 4.5 `print_progress` . . . 4
- 4.6 `convert_to_bytes` . . . 4

5 Code Structure **4**

6 Customization **5**

7 Error Handling and User friendly interface **6**

8 Source Code **7**

1 Introduction

Welcome to the documentation for the Bash Organizer project. This documentation provides a comprehensive guide on how to use and understand the functionalities of the File Organizer script.

2 Project Overview

The File Organizer is a bash script designed to help you organize your files by moving them into categorized folders based on their file extensions. It simplifies the task of managing and organizing a large number of files by automating the process.

3 Usage

The following table provides the usage and the options used in the project:

```
bash organizer.sh [srcdir] [destdir] [options]
```

Here is the description of the various options used in the code:

Option	Description
`--help`	Display the help message
`-s [type]`	Sort files based on `ext` for extension or `date` for creation date.Default is `ext`
`-d`	Delete the files from the destination after copying them
`-l [log_file]`	Creates a logfile creating a record of the moved filesThe name of the lof file is required
`-p`	Disable the progress bar
`-e [ext1,ext2,...]`	Exclude file types or directories from being organized.Arguments should be comma separated."
`-i [ext1,ext2,...]`	Include file types or directories from being organized.Arguments should be comma separated. *Note: Both exclude and include commands cannot be used together*
`-f [max_size]`	Specifies an upper limit for the size if the files to be transferred. The file size should be of the format `<integer>[B/KB/MB/KB]`

Some examples of usage are:

- ```
  bash organizer.sh src dest -e pdf,png -d
  ```

  This organizes the file according to their extension excluding the pdf and png files . Then later deletes the orignal files.

- ```
  bash organizer.sh src dest -i txt, tex -l
     logfile.txt -s date
  ```

  This organizes only the files having txt and tex extensions and puts them into folder according to their creation dates. Then later creates a log file which has all the moved files.

- ```
  bash organizer.sh src dest -f 15KB -s ext -d
  ```

  This command takes all the files less than 15KB of size and then organizes them on the basis if their extension and then later deletes the files which were moved.

- ```
  bash organizer.sh --help
  ```

  This simply displays the help message.

4 Functions

Here I will describe a few functions I used in the bash script for variuos purposes

4.1 `show_help`

This fucntion jsut displays the usage and help message

4.2 `get_ext`

This function is used to extract the folder the file has to be transferred on the basis of the file name and options chosen by the user

4.3 `get_date`

The bash command for getting the date of the file creation is an expanded format, so this function is used ti extract the date of the file creation and convert it into desirable format.

4.4 get_available_filename

This function is used to get the available filename for a file on the basis if the the files orignally present in the folder. for example of we need to copy a file `abc.txt` to the `txt` folder and there already exists a file `abc.txt` in that folder this function will help in renaming the file to `abc_1.txt` and so on.

4.5 print_progress

This function is used to print a progress bar while the file copying takes place.

4.6 convert_to_bytes

Simply converts the file size given by user to bytes so that I can process it in my program

5 Code Structure

The code begins with definition of a few variavles to make the project colourful.

```
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
CYAN='\033[0;36m'
NC='\033[0m' # No Color
```

After that I gave the function definition for all the functions.Next I created a few temporary emtpy files to store filenames , extension names etc. After that I parsed the command line arguments to get the src and dest directory , and added a few user friendly messages in case the directories dont exist.

The multiple lines are present is used to give the effect of the dots increasing one by one. I achieved that by using the `\r` tag, which overwrites the previous line. I have used this technique in many other places.

The command `shift 2` in the last line is used to shift the parser two units right so I can start reading options.

Next I took parsed the various options using the `while getopts ":s:l:e:i:dp" opt; do`. I added various kind of error handling to be displayed if the options are not given in correct format. The various error handling I performed are to check whether the options are given where they are required, for example `-e` won't make any sense without options. If the user has chosen the delete option , the user is prompted whether they are sure they want to delete the

orignal files. The program also raises an error if the user choses both the `-e` and `-i` options together.

After that I added the feature to unzip the zip files and put them in a temporary folder `unzipped_files` which I later deleted.

Then comes the main loop where I serched for all the files which are not hidden and performed the operations on the files on the basis of the options given by the user. I also added the code for the progress bar in this main loop. Bascially what this main loop does is that for everyfile it first checks whether it has to be copied on the basis of the options provided, then extracts the folder using `get_ext` function, copies the file, deletes it if required , and sends it to the log file if required.

Next I printed the summary , showing the number of folders created when the program was run, the number of files transferred, the number of files in each folder which was involved in program.

6 Customization

I have added various customization features in the project, here are some of them:

- **Progress Bar**

  In the summary while the files are being copied a progress bar is displayed showing the progress of the files being transferred

- **Max File Size**

  This features allows you to set a maximum size of the files. Only the files less than this size will get organized

- **Unzippinig**

  The program unzips all the zip files automatically and stores them ina temporary folder to be organized later.

- **Logfiles**

  There is an option to create a logfile which stores which files were moved , their source and their destination.

- **Exclude**

  This feature is used to exclude files of certain types from being copied

- **Include**

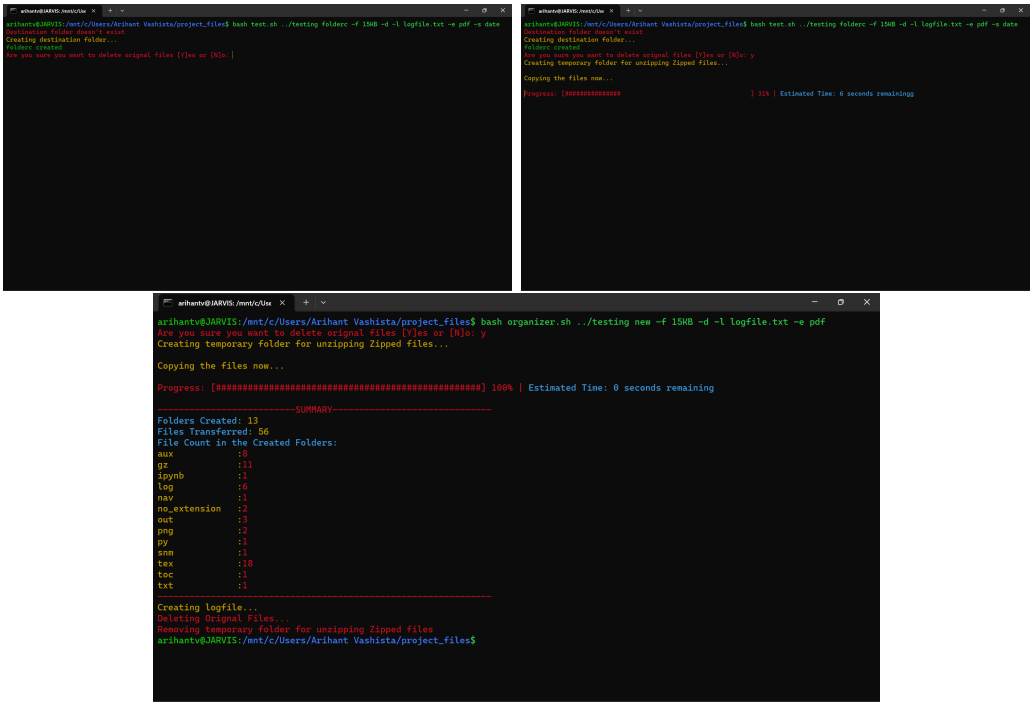
  This feature is used to includes only files of certain types for being copied.

7 Error Handling and User friendly interface

I have included several other features which makes the interface user friendly , these features include:

- Coloured texts, errors in red, information in yellow etc
- Prompting the user wether they are sure they want to delete the files
- Whenever some error has occured or the input given by user is no of proper format , an error message is raised and help function is showed
- Buffering, at several places I have added the buffering effect while creating directories or moving files to give a real feel.

Here are some screenshots:-



8 Source Code

I am also attaching the complete source code of the program in the documentation:

```
#!bin/bash

# Color variables
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
CYAN='\033[0;36m'
NC='\033[0m' # No Color

# Function to display the script's usage
show_help() {
  echo -e "${YELLOW}Usage: bash organizer.sh [srcdir] [destdir] [options]"
  echo "Options:"
  echo "  --help                Display this help message"
  echo "  -s [type]             Sort files based on 'ext' for extension or 'date' for creation date${NC}"
  echo "  -d                    Delete the files from the destination"
  echo "  -l [log_file]         The name of the lof file is required"
  echo "  -p                    Disable the progress bar"
  echo "  -e [ext1,ext2,...]    Exclude file types or directories from being organized.Arguments should be comma separated."
  echo "  -i [ext1,ext2,...]    Include file types or directories from being organized.Arguments should be comma separated."
  echo "                        Both include and exclude cannot be used together"
  echo "  -f [max_size]         Sets the upperlimit for the size if the files to copy, the max size should be given in this format: <integer>[B|KB|MB|GB]"
  echo -e "$NC"
  exit 1
}

handle_error(){
  echo -e "${RED} Some kind of error was incurred,
```

```
     please ensure you have the correct usage${NC}"
  show_help

}

get_ext(){
  local file=$1
  local name=`basename $file`
  if [ $2 = "date" ]; then
    echo $(get_date $1)

  elif [[ ! "$name" == *.* ]]; then
    echo "no_extension"
  else
    echo ${name##*.}
  fi
}

#gets the date in the required format
get_date(){
  creation_time=$(stat -c %x "$1")

  # Extract day, month, and year from the creation time
  IFS=' ' read -ra date_parts <<< "$creation_time"
  IFS='-' read -ra date <<< "${date_parts[0]}"
  day=${date[0]}
  month=${date[1]}
  year=${date[2]}

  # Rearrange the date parts to ddmmyyyy format
  formatted_date="${year}${month}${day}"
  echo $formatted_date
}

#This function is used to get the available filename on
   the basis of the files present in the
#destination folder
get_available_filename(){
  local file=$1
  local dest=$2
  local ext=$3
  let num=1
  local name=`basename $file`
```

```
70
71    if [[ ! "$name" == *.* ]]; then
72      name_check=$name
73      while [ -f $dest"/"$ext"/"$name_check ]; do
74        name_check="${name}_$num" # Append increment
             number
75        let num=$num+1
76      done
77      echo $name_check
78    else
79      extension=${name##*.}
80      name="${name%.*}" # Remove extension
81      name_check=$name
82
83      # Loop until a unique file name is found
84      while [ -f $dest"/"$ext"/"$name_check"."$extension
           ]; do
85        name_check="${name}_$num" # Append increment
             number
86        let num=$num+1
87      done
88
89      echo $name_check"."$extension
90    fi
91
92  }
93
94  # Function to print progress bar
95  print_progress() {
96    local width=50  # Width of the progress bar
97    local progress=$1
98    local completed=$((progress * width / 100))
99    local remaining=$((width - completed))
100   local bar=$(printf '%*s' "$completed" | tr ' ' '#')
        #creates a string of specified length and then
        replaces it with #
101   local space=$(printf '%*s' "$remaining" )
102
103   # Calculate elapsed time
104   local elapsed_time=$(($(date +%s) - start_time))
105
106   # Calculate estimated time of completion
107   if [ $progress -ne 0 ]; then
```

```
108     local total_time=$((elapsed_time * 100 / progress))
109     local remaining_time=$((total_time - elapsed_time))
110 
111     echo -ne "${RED}Progress: [$bar$space] $progress%
            |${CYAN} Estimated Time: $remaining_time seconds
            remaining\r${NC}"
112 
113   fi
114 }
115 
116 convert_to_bytes() {
117   local input=$1
118   local size=$(echo "$input" | grep -oE '[0-9]+')
119   local unit=$(echo "$input" | grep -oE '[A-Za-z]+')
120 
121   case $unit in
122     B) size=$((size * 1));;
123     KB) size=$((size * 1024));;
124     MB) size=$((size * 1024 * 1024));;
125     GB) size=$((size * 1024 * 1024 * 1024));;
126     TB) size=$((size * 1024 * 1024 * 1024 * 1024));;
127     *) echo "Invalid unit: $unit"; exit 1;;
128   esac
129 
130   echo "$size"
131 }
132 
133 #creating empty temporary files needed for function
134 #There was some issue with touch so I didn't use it
135 echo > test | grep '[^ ]' > output
136 cat output > extensions.txt
137 cat output > moved_files.txt
138 cat output > added_folders.txt
139 cat output > all_files.txt
140 rm test output
141 #trap handle_error ERR
142 #help function
143 
144 if [ "$#" -lt 2 ]; then
145   if [[ $1 == "--help" ]]; then
146     show_help
147     exit 1
148   else
```

```
149     echo -e "${RED}Invalid Options"
150     show_help
151     exit 1
152   fi
153 fi
154
155 #We will first extract the src and dest directories
      from the command line
156 src=$1
157 dest=$2
158
159
160 #check if dest exists
161 if [ ! -d $dest ]; then
162   echo -e "${RED}Destination folder doesn't exist${NC}"
163   sleep 0.5
164   echo -en "${YELLOW}Creating destination folder${NC}\r"
165   sleep 0.75
166   echo -en "${YELLOW}Creating destination
        folder.${NC}\r"
167   sleep 0.75
168   echo -en "${YELLOW}Creating destination
        folder..${NC}\r"
169   sleep 0.75
170   echo -e "${YELLOW}Creating destination
        folder...${NC}\r"
171   sleep 0.75
172   mkdir -p $dest
173   echo -e "${GREEN}$dest created${NC}"
174   sleep 0.5
175 fi
176
177 #check if src exists
178 if [ ! -d $src ]; then
179   echo -e "${RED}Source doesn't exist, please enter
        valid source${NC}"
180   exit 1
181 fi
182 shift 2
183
184 sort_type="ext"
185 delete="false"
186 create_logfile="false"
```

```
187 exclude_list=()
188 enable_exclude="false"
189 include_list=()
190 enable_include="fasle"
191 disable_progress="false"
192 enable_max_size="false"
193
194 while getopts ":s:l:e:i:f:dp" opt; do
195   case $opt in
196     d)
197       echo -e -n "${RED}"
198       read -p "Are you sure you want to delete orignal
             files [Y]es or [N]o: " choice
199       if [ $choice = "Y" ]; then
200         delete="true"
201       elif [ $choice = "y" ]; then
202         delete="true"
203       fi
204       echo -e -n "${NC}"
205       ;;
206     p)
207       disable_progress="true"
208       ;;
209     f)
210       size_arg=$OPTARG
211       # Validate file size format using regex
212       if ! [[ $size_arg =~ ^[0-9]+(B|KB|MB|GB)$ ]]; then
213         echo -e "${RED}Invalid file size format. Please
              use the format <number>[B|KB|MB|GB].${NC}"
214         exit 1
215       fi
216       enable_max_size="true"
217       size_arg=$(convert_to_bytes $size_arg)
218       ;;
219
220     s)
221       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
222         echo -e "${RED}-$opt requires an argument.${NC}"
223         show_help
224         exit 1
225       fi
226       sort_type=$OPTARG
227       ;;
```

```
228     l)
229       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
230         echo -e "${RED}-$opt requires an argument.${NC}"
231         show_help
232         exit 1
233       fi
234       create_logfile="true"
235       log_file=$OPTARG
236       if [ -f $log_file  ]; then
237         rm $log_file
238       fi
239       ;;
240     e)
241       if [[ "$enable_include" = "true" ]]; then
242         echo -e "${RED}Both include and exclude option
              can't be used together${NC}"
243         show_help
244         exit 1
245       fi
246       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
247         echo -e "${RED}-$opt requires arguments
              separated by commas.${NC}"
248         show_help
249         exit 1
250       fi
251       IFS=',' read -ra args <<< "$OPTARG"  # Split the
           comma-separated values into an array
252       exclude_list+=("${args[@]}")  # Append the array
           elements to the main array
253       enable_exclude="true"
254       ;;
255     i)
256       if [[ "$enable_exclude" = "true" ]]; then
257         echo -e "${RED}Both include and exclude option
              can't be used together${NC}"
258         show_help
259         exit 1
260       fi
261       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
262         echo -e "${RED}-$opt requires arguments
              separated by commas.${NC}"
263         show_help
264         exit 1
```

```
265       fi
266       IFS=',' read -ra args <<< "$OPTARG"  # Split the
             comma-separated values into an array
267       include_list+=("${args[@]}")  # Append the array
             elements to the main array
268       enable_include="true"
269       ;;
270
271     :)
272       echo -e "${RED} -$OPTARG requires an argument:"
273       show_help
274       exit 1
275       ;;
276     \?)
277       echo -e "${RED}Invalid option: -$OPTARG"
278       show_help
279       exit 1
280       ;;
281   esac
282 done
283
284 if [ ! -d "$src/unzipped_files" ]; then
285   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files\r"
286   sleep 0.75
287   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files.\r"
288   sleep 0.75
289   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files..\r"
290   sleep 0.75
291   echo -e "${YELLOW}Creating temporary folder for
        unzipping Zipped files...\r"
292   sleep 0.5
293   mkdir -p "$src/unzipped_files"
294 fi
295
296 #unzipping the zipped folders
297 for file in $(find "$src" -type f -name "*.zip"); do
298   # Unzip the files
299   echo -e "${GREEN}Unzipping `basename $file` ${CYAN}"
300   unzip -q -o "$file" -d "$src/unzipped_files"
301 done
```

```
302
303 find $src -type f | sed -n '/\/[^./]\+\.[^./]\+$/p' >>
       all_files.txt
304 find $src -type f | sed -n '/\/[^./]\+$/p' >>
       all_files.txt
305
306 total_files=$(wc -l < "all_files.txt")
307 start_time=$(date +%s)
308 current_file=0
309 echo
310 echo -e "${YELLOW}Copying the files now..."
311 echo -e "${RED}"
312 sleep 0.5
313
314 #first I took file from the source then piped it to the
       sed command
315 #to get the files which have an extension
316 for file in `cat "all_files.txt"`
317
318 do
319   #check whether the user wants a progress bar or not
320   if [ $disable_progress = "false" ]; then
321     # Update progress
322     ((current_file++))
323     progress=$((current_file * 100 / total_files))
324
325     # Display progress bar
326     print_progress "$progress"
327   fi
328
329   #exctracted basename from file
330   name=`basename $file`
331
332   #get whether to sort about date or extension
333   ext=`get_ext $file $sort_type`
334   #now we check whether we need to copy the file or not
335   copy_files="true"
336   if [ $enable_exclude = "true" ]; then
337     for f in "${exclude_list[@]}"
338     do
339       if [ $f = ${name##*.} ]; then
340         copy_files="false"
341       fi
```

```
342     done
343   fi
344
345   if [ $enable_include = "true" ]; then
346     copy_files="false"
347     for f in "${include_list[@]}"
348     do
349       if [ $f = ${name##*.} ]; then
350         copy_files="true"
351       fi
352     done
353   fi
354
355   if [ $enable_max_size = "true" ];then
356     file_size=$(stat -c %s "$file")
357     if [ $file_size -gt $size_arg ]; then
358       copy_files="false"
359     fi
360   fi
361
362
363   if [ $copy_files = "true" ]; then
364
365     #make extension folders to copy files
366     echo $ext >> extensions.txt
367     if [ ! -d "$dest/$ext" ]; then
368       echo $ext>>added_folders.txt
369     fi
370     mkdir -p "$dest/$ext"
371
372     #creating the logfile
373     if [ $create_logfile = "true" ]; then
374       echo "`get_available_filename $file $dest $ext`
            moved from $src to $dest/$ext" >> $log_file
375     fi
376
377     #copying the file
378     echo `get_available_filename $file $dest $ext` >>
          moved_files.txt
379     cp $file $dest"/"$ext"/"`get_available_filename
          $file $dest $ext`
380
381     #delete the moved files if option was given
```

```
382     if [ $delete = "true" ]; then
383       rm $file
384     fi
385
386   fi
387 done
388 echo -e "${NC}"
389 echo
390
391 cat extensions.txt|sort|uniq > extensions1.txt
      #created a file containing all the extensions
392 cat added_folders.txt|sort|uniq > added_folders1.txt
393 #time to print the Summary and other user friendly
      messages
394 sleep 0.5
395 echo -e
      "${RED}--------------------------SUMMARY-----------------------------
396 sleep 0.5
397 echo -e "${CYAN}Folders Created${NC}: ${YELLOW}"`wc -l
      added_folders1.txt|awk '{print $1}'`"${NC}"
398 sleep 0.5
399 echo -e "${CYAN}Files Transferred${NC}: ${YELLOW}"`wc
      -l moved_files.txt|awk '{print $1}'`"${NC}"
400 sleep 0.5
401 echo -e "${CYAN}File Count in the Created Folders:${NC}"
402 for folder in $(cat extensions1.txt)
403 do
404   count=$(ls "$dest/$folder" | wc -l)
405   echo -e -n "${YELLOW}"
406   printf "%-15s:" "$folder"
407   echo -e -n "${RED}"
408   printf "%-5d\n" "$count"
409   sleep 0.2
410 done
411 sleep 0.5
412 echo -e
      "${RED}-----------------------------------------------------------
413 sleep 0.5
414
415 if [ $create_logfile = "true" ]; then
416   echo -ne "${YELLOW}Creating logfile\r"
417   sleep 0.75
418   echo -ne "${YELLOW}Creating logfile.\r"
```

```
419   sleep 0.75
420   echo -ne "${YELLOW}Creating logfile..\r"
421   sleep 0.75
422   echo -ne "${YELLOW}Creating logfile..."
423   sleep 0.5
424 fi
425 echo
426 if [ $delete = "true" ]; then
427   echo -ne "${RED}Deleting Orignal Files\r"
428   sleep 0.75
429   echo -ne "Deleting Orignal Files.\r"
430   sleep 0.75
431   echo -ne "Deleting Orignal Files..\r"
432   sleep 0.75
433   echo -ne "Deleting Orignal Files...${NC}"
434   sleep 0.5
435 fi
436 echo
437 rm moved_files.txt added_folders.txt extensions.txt
      all_files.txt extensions1.txt added_folders1.txt
438 echo -e "${RED}Removing temporary folder for unzipping
      Zipped files${NC}"
439 sleep 0.5
440 rm -r "$src/unzipped_files"
441
442 #trap - ERR
```