Bash Organizer
CS104 Project

Arihant Vashista, 22b0958

June 14, 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.I have used the following sources to make my project [1][2]

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 files the name of the log 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.If you want to include files with no extension use the* ***`no_extension`*** *tag.*
`-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`.[3] 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.Here if you want to include files with no extension use the `no_extension` tag.

- **Include**

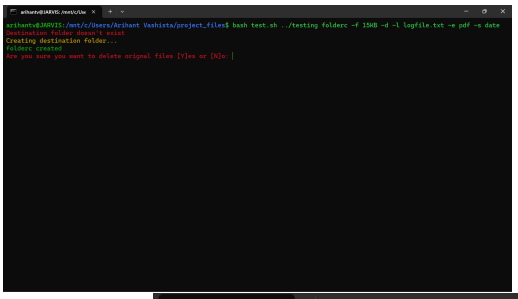
  This feature is used to includes only files of certain types for being copied. Here if you want to include files with no extension use the `no_extension` tag.

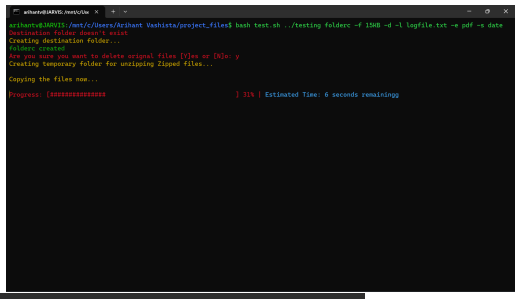
7 Error Handling and User friendly interface

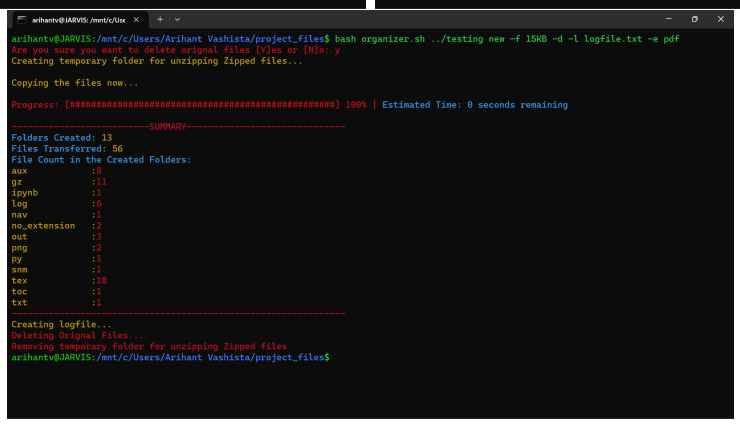
I have included several other features which makes the interface user friendly , these features include:

- 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:-







References

[1] URL: https://www.w3schools.io/terminal/bash-tutorials/.

[2] URL: https://www.geeksforgeeks.org/introduction-linux-shell-shell-scripting/.

[3] URL: https://www.computerhope.com/unix/bash/getopts.htm.

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. Use the no_extension tag
```

```
    for handling files without extension"
  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
}
```

```
61
62 #This function is used to get the available filename on
      the basis of the files present in the
63 #destination folder
64 get_available_filename(){
65   local file=$1
66   local dest=$2
67   local ext=$3
68   let num=1
69   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 let check_time=$((2**63 - 1))
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   # Calculate estimated time of completion
106   if [ $progress -ne 0 ]; then
107     local total_time=$((elapsed_time * 100 / progress))
108     local remaining_time=$((total_time - elapsed_time))
109     #echo "$remaining_time $check_time"
110     echo -n
111     if [ $remaining_time -le $check_time ]; then
112         echo -ne "${RED}Progress: [$bar$space]
              $progress% |${CYAN} Estimated Time:
              $remaining_time seconds remaining\r${NC}"
113               check_time=$remaining_time
114     fi
115   fi
116 }
117
118 convert_to_bytes() {
119   local input=$1
120   local size=$(echo "$input" | grep -oE '[0-9]+')
121   local unit=$(echo "$input" | grep -oE '[A-Za-z]+')
122
123   case $unit in
124     B) size=$((size * 1));;
125     KB) size=$((size * 1024));;
126     MB) size=$((size * 1024 * 1024));;
127     GB) size=$((size * 1024 * 1024 * 1024));;
128     TB) size=$((size * 1024 * 1024 * 1024 * 1024));;
129     *) echo "Invalid unit: $unit"; exit 1;;
130   esac
131
132   echo "$size"
133 }
134
135 #creating empty temporary files needed for function
136 #There was some issue with touch so I didn't use it
137 echo > test | grep '[^ ]' > output
138 cat output > extensions.txt
```

```
cat output > moved_files.txt
cat output > added_folders.txt
cat output > all_files.txt
rm test output
#trap handle_error ERR
#help function

if [ "$#" -lt 2 ]; then
  if [[ $1 == "--help" ]]; then
    show_help
    exit 1
  else
    echo -e "${RED}Invalid Options"
    show_help
    exit 1
  fi
fi

#We will first extract the src and dest directories
   from the command line
src=$1
dest=$2


#check if dest exists
if [ ! -d $dest ]; then
  echo -e "${RED}Destination folder doesn't exist${NC}"
  sleep 0.5
  echo -en "${YELLOW}Creating destination folder${NC}\r"
  sleep 0.75
  echo -en "${YELLOW}Creating destination
     folder.${NC}\r"
  sleep 0.75
  echo -en "${YELLOW}Creating destination
     folder..${NC}\r"
  sleep 0.75
  echo -e "${YELLOW}Creating destination
     folder...${NC}\r"
  sleep 0.75
  mkdir -p $dest
  echo -e "${GREEN}$dest created${NC}"
  sleep 0.5
fi
```

```
178
179 #check if src exists
180 if [ ! -d $src ]; then
181   echo -e "${RED}Source doesn't exist, please enter
        valid source${NC}"
182   exit 1
183 fi
184 shift 2
185
186 sort_type="ext"
187 delete="false"
188 create_logfile="false"
189 exclude_list=()
190 enable_exclude="false"
191 include_list=()
192 enable_include="fasle"
193 disable_progress="false"
194 enable_max_size="false"
195
196 while getopts ":s:l:e:i:f:dp" opt; do
197   case $opt in
198     d)
199       echo -e -n "${RED}"
200       read -p "Are you sure you want to delete orignal
           files [Y]es or [N]o: " choice
201       if [ $choice = "Y" ]; then
202         delete="true"
203       elif [ $choice = "y" ]; then
204         delete="true"
205       fi
206       echo -e -n "${NC}"
207       ;;
208     p)
209       disable_progress="true"
210       ;;
211     f)
212       size_arg=$OPTARG
213       # Validate file size format using regex
214       if ! [[ $size_arg =~ ^[0-9]+(B|KB|MB|GB)$ ]]; then
215         echo -e "${RED}Invalid file size format. Please
             use the format <number>[B|KB|MB|GB].${NC}"
216         exit 1
217       fi
```

```
218       enable_max_size="true"
219       size_arg=$(convert_to_bytes $size_arg)
220       ;;
221
222     s)
223       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
224         echo -e "${RED}-$opt requires an argument.${NC}"
225         show_help
226         exit 1
227       fi
228       sort_type=$OPTARG
229       ;;
230     l)
231       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
232         echo -e "${RED}-$opt requires an argument.${NC}"
233         show_help
234         exit 1
235       fi
236       create_logfile="true"
237       log_file=$OPTARG
238       if [ -f $log_file  ]; then
239         rm $log_file
240       fi
241       ;;
242     e)
243       if [[ "$enable_include" = "true" ]]; then
244         echo -e "${RED}Both include and exclude option
              can't be used together${NC}"
245         show_help
246         exit 1
247       fi
248       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
249         echo -e "${RED}-$opt requires arguments
              separated by commas.${NC}"
250         show_help
251         exit 1
252       fi
253       IFS=',' read -ra args <<< "$OPTARG"  # Split the
            comma-separated values into an array
254       exclude_list+=("${args[@]}")  # Append the array
            elements to the main array
255       enable_exclude="true"
256       ;;
```

```
257     i)
258       if [[ "$enable_exclude" = "true" ]]; then
259         echo -e "${RED}Both include and exclude option
              can't be used together${NC}"
260         show_help
261         exit 1
262       fi
263       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
264         echo -e "${RED}-$opt requires arguments
              separated by commas.${NC}"
265         show_help
266         exit 1
267       fi
268       IFS=',' read -ra args <<< "$OPTARG"  # Split the
            comma-separated values into an array
269       include_list+=("${args[@]}")  # Append the array
            elements to the main array
270       enable_include="true"
271       ;;
272
273     :)
274       echo -e "${RED} -$OPTARG requires an argument:"
275       show_help
276       exit 1
277       ;;
278     \?)
279       echo -e "${RED}Invalid option: -$OPTARG"
280       show_help
281       exit 1
282       ;;
283   esac
284 done
285
286 if [ ! -d "$src/unzipped_files" ]; then
287   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files\r"
288   sleep 0.75
289   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files.\r"
290   sleep 0.75
291   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files..\r"
292   sleep 0.75
```

```
293   echo -e "${YELLOW}Creating temporary folder for
          unzipping Zipped files...\r"
294   sleep 0.5
295   mkdir -p "$src/unzipped_files"
296 fi
297
298 #unzipping the zipped folders
299 for file in $(find "$src" -type f -name "*.zip"); do
300   # Unzip the files
301   echo -e "${GREEN}Unzipping `basename $file` ${CYAN}"
302   unzip -q -o "$file" -d "$src/unzipped_files"
303 done
304
305 find $src -type f | sed -n '/\/[^./]\+\.[^./]\+$/p' >>
       all_files.txt
306 find $src -type f | sed -n '/\/[^./]\+$/p' >>
       all_files.txt
307
308 total_files=$(wc -l < "all_files.txt")
309 start_time=$(date +%s)
310 current_file=0
311 echo
312 echo -e "${YELLOW}Copying the files now..."
313 echo -e "${RED}"
314 sleep 0.5
315
316 #first I took file from the source then piped it to the
       sed command
317 #to get the files which have an extension
318 for file in `cat "all_files.txt"`
319
320 do
321   #check whether the user wants a progress bar or not
322   if [ $disable_progress = "false" ]; then
323     # Update progress
324     ((current_file++))
325     progress=$((current_file * 100 / total_files))
326
327     # Display progress bar
328     print_progress "$progress"
329   fi
330
331   #exctracted basename from file
```

```
332   name=`basename $file`
333
334   #get whether to sort about date or extension
335   ext=`get_ext $file $sort_type`
336   #now we check whether we need to copy the file or not
337   copy_files="true"
338   if [ $enable_exclude = "true" ]; then
339     for f in "${exclude_list[@]}"
340     do
341       if [ $f = "no_extension" ]; then
342         if [[ ! "$name" == *.* ]]; then
343           copy_files="false"
344         fi
345       elif [ $f = ${name##*.} ]; then
346         copy_files="false"
347       fi
348     done
349   fi
350
351   if [ $enable_include = "true" ]; then
352     copy_files="false"
353     for f in "${include_list[@]}"
354     do
355       if [ $f = "no_extension" ]; then
356         if [[ ! "$name" == *.* ]]; then
357           copy_files="true"
358         fi
359       elif [ $f = ${name##*.} ]; then
360         copy_files="true"
361       fi
362     done
363   fi
364
365   if [ $enable_max_size = "true" ];then
366     file_size=$(stat -c %s "$file")
367     if [ $file_size -gt $size_arg ]; then
368       copy_files="false"
369     fi
370   fi
371
372
373   if [ $copy_files = "true" ]; then
374
```

```
375     #make extension folders to copy files
376     echo $ext >> extensions.txt
377     if [ ! -d "$dest/$ext" ]; then
378       echo $ext>>added_folders.txt
379     fi
380     mkdir -p "$dest/$ext"
381
382     #creating the logfile
383     if [ $create_logfile = "true" ]; then
384       echo "`get_available_filename $file $dest $ext`
            moved from $src to $dest/$ext" >> $log_file
385     fi
386
387     #copying the file
388     echo `get_available_filename $file $dest $ext` >>
          moved_files.txt
389     cp $file $dest"/"$ext"/"`get_available_filename
          $file $dest $ext`
390
391     #delete the moved files if option was given
392     if [ $delete = "true" ]; then
393       rm $file
394     fi
395
396   fi
397 done
398 echo -e "${NC}"
399 echo
400
401 cat extensions.txt|sort|uniq > extensions1.txt
      #created a file containing all the extensions
402 cat added_folders.txt|sort|uniq > added_folders1.txt
403 #time to print the Summary and other user friendly
      messages
404 sleep 0.5
405 echo -e
      "${RED}-------------------------SUMMARY-----------------------------
406 sleep 0.5
407 echo -e "${CYAN}Folders Created${NC}: ${YELLOW}"`wc -l
      added_folders1.txt|awk '{print $1}'`"${NC}"
408 sleep 0.5
409 echo -e "${CYAN}Files Transferred${NC}: ${YELLOW}"`wc
      -l moved_files.txt|awk '{print $1}'`"${NC}"
```

```
410 sleep 0.5
411 echo -e "${CYAN}File Count in the Created Folders:${NC}"
412 for folder in $(cat extensions1.txt)
413 do
414   count=$(ls "$dest/$folder" | wc -l)
415   echo -e -n "${YELLOW}"
416   printf "%-15s:" "$folder"
417   echo -e -n "${RED}"
418   printf "%-5d\n" "$count"
419   sleep 0.2
420 done
421 sleep 0.5
422 echo -e
      "${RED}-----------------------------------------------------------
423 sleep 0.5
424
425 if [ $create_logfile = "true" ]; then
426   echo -ne "${YELLOW}Creating logfile\r"
427   sleep 0.75
428   echo -ne "${YELLOW}Creating logfile.\r"
429   sleep 0.75
430   echo -ne "${YELLOW}Creating logfile..\r"
431   sleep 0.75
432   echo -ne "${YELLOW}Creating logfile..."
433   sleep 0.5
434 fi
435 echo
436 if [ $delete = "true" ]; then
437   echo -ne "${RED}Deleting Orignal Files\r"
438   sleep 0.75
439   echo -ne "Deleting Orignal Files.\r"
440   sleep 0.75
441   echo -ne "Deleting Orignal Files..\r"
442   sleep 0.75
443   echo -ne "Deleting Orignal Files...${NC}"
444   sleep 0.5
445 fi
446 echo
447 rm moved_files.txt added_folders.txt extensions.txt
      all_files.txt extensions1.txt added_folders1.txt
448 echo -e "${RED}Removing temporary folder for unzipping
      Zipped files${NC}"
449 sleep 0.5
```

```
450 rm -r "$src/unzipped_files"
451
452 #trap - ERR
```