Bash Organizer
CS104 Project

Arihant Vashista, 22b0958

June 8, 2023

Contents

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**

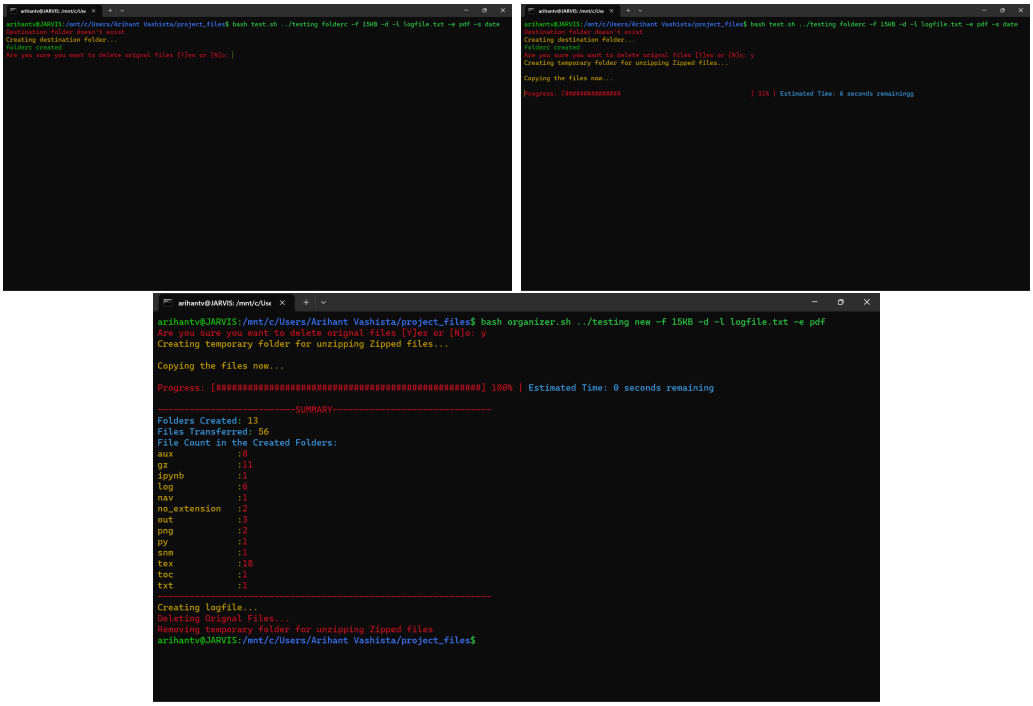
  This feature is used to includes only files of certain types for being copied.

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    extension=${name##*.}
71    name="${name%.*}" # Remove extension
72    name_check=$name
73
74    # Loop until a unique file name is found
75    while [ -f $dest"/"$ext"/"$name_check"."$extension ];
          do
76      name_check="${name}_$num" # Append increment number
77      let num=$num+1
78    done
79
80    echo $name_check"."$extension
81  }
82
83  # Function to print progress bar
84  print_progress() {
85    local width=50  # Width of the progress bar
86    local progress=$1
87    local completed=$((progress * width / 100))
88    local remaining=$((width - completed))
89    local bar=$(printf '%*s' "$completed" | tr ' ' '#')
          #creates a string of specified length and then
          replaces it with #
90    local space=$(printf '%*s' "$remaining" )
91
92    # Calculate elapsed time
93    local elapsed_time=$(($(date +%s) - start_time))
94
95    # Calculate estimated time of completion
96    if [ $progress -ne 0 ]; then
97      local total_time=$((elapsed_time * 100 / progress))
98      local remaining_time=$((total_time - elapsed_time))
99
100     echo -ne "${RED}Progress: [$bar$space] $progress%
            |${CYAN} Estimated Time: $remaining_time seconds
            remaining\r${NC}"
101
102   fi
103 }
104
105 convert_to_bytes() {
106   local input=$1
107   local size=$(echo "$input" | grep -oE '[0-9]+')
```

```
108   local unit=$(echo "$input" | grep -oE '[A-Za-z]+')
109
110   case $unit in
111     B) size=$((size * 1));;
112     KB) size=$((size * 1024));;
113     MB) size=$((size * 1024 * 1024));;
114     GB) size=$((size * 1024 * 1024 * 1024));;
115     TB) size=$((size * 1024 * 1024 * 1024 * 1024));;
116     *) echo "Invalid unit: $unit"; exit 1;;
117   esac
118
119   echo "$size"
120 }
121
122 #creating empty temporary files needed for function
123 #There was some issue with touch so I didn't use it
124 echo > test | grep '[^ ]' > output
125 cat output > extensions.txt
126 cat output > moved_files.txt
127 cat output > added_folders.txt
128 rm test output
129 #trap handle_error ERR
130 #help function
131
132 if [ "$#" -lt 2 ]; then
133   if [[ $1 == "--help" ]]; then
134     show_help
135     exit 1
136   else
137     echo -e "${RED}Invalid Options"
138     show_help
139     exit 1
140   fi
141 fi
142
143 #We will first extract the src and dest directories
      from the command line
144 src=$1
145 dest=$2
146
147
148 #check if dest exists
149 if [ ! -d $dest ]; then
```

```
150   echo -e "${RED}Destination folder doesn't exist${NC}"
151   sleep 0.5
152   echo -en "${YELLOW}Creating destination folder${NC}\r"
153   sleep 0.75
154   echo -en "${YELLOW}Creating destination
        folder.${NC}\r"
155   sleep 0.75
156   echo -en "${YELLOW}Creating destination
        folder..${NC}\r"
157   sleep 0.75
158   echo -e "${YELLOW}Creating destination
        folder...${NC}\r"
159   sleep 0.75
160   mkdir -p $dest
161   echo -e "${GREEN}$dest created${NC}"
162   sleep 0.5
163 fi
164
165 #check if src exists
166 if [ ! -d $src ]; then
167   echo -e "${RED}Source doesn't exist, please enter
        valid source${NC}"
168   exit 1
169 fi
170 shift 2
171
172 sort_type="ext"
173 delete="false"
174 create_logfile="false"
175 exclude_list=()
176 enable_exclude="false"
177 include_list=()
178 enable_include="fasle"
179 disable_progress="false"
180 enable_max_size="false"
181
182 while getopts ":s:l:e:i:f:dp" opt; do
183   case $opt in
184     d)
185       echo -e -n "${RED}"
186       read -p "Are you sure you want to delete orignal
            files [Y]es or [N]o: " choice
187       if [ $choice = "Y" ]; then
```

```
188           delete="true"
189         elif [ $choice = "y" ]; then
190           delete="true"
191         fi
192         echo -e -n "${NC}"
193         ;;
194       p)
195         disable_progress="true"
196         ;;
197       f)
198         size_arg=$OPTARG
199         # Validate file size format using regex
200         if ! [[ $size_arg =~ ^[0-9]+(B|KB|MB|GB)$ ]]; then
201           echo -e "${RED}Invalid file size format. Please
              use the format <number>[B|KB|MB|GB].${NC}"
202           exit 1
203         fi
204         enable_max_size="true"
205         size_arg=$(convert_to_bytes $size_arg)
206         ;;
207
208       s)
209         if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
210           echo -e "${RED}-$opt requires an argument.${NC}"
211           show_help
212           exit 1
213         fi
214         sort_type=$OPTARG
215         ;;
216       l)
217         if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
218           echo -e "${RED}-$opt requires an argument.${NC}"
219           show_help
220           exit 1
221         fi
222         create_logfile="true"
223         log_file=$OPTARG
224         ;;
225       e)
226         if [[ "$enable_include" = "true" ]]; then
227           echo -e "${RED}Both include and exclude option
              can't be used together${NC}"
228           show_help
```

```
229         exit 1
230       fi
231       if [[ -z "$OPTARG" || "$OPTARG" == -* ]]; then
232         echo -e "${RED}-$opt requires arguments
              separated by commas.${NC}"
233         show_help
234         exit 1
235       fi
236       IFS=',' read -ra args <<< "$OPTARG"  # Split the
            comma-separated values into an array
237       exclude_list+=("${args[@]}")  # Append the array
            elements to the main array
238       enable_exclude="true"
239       ;;
240     i)
241       if [[ "$enable_exclude" = "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       include_list+=("${args[@]}")  # Append the array
            elements to the main array
253       enable_include="true"
254       ;;
255
256     :)
257       echo -e "${RED} -$OPTARG requires an argument:"
258       show_help
259       exit 1
260       ;;
261     \?)
262       echo -e "${RED}Invalid option: -$OPTARG"
263       show_help
264       exit 1
```

```
265         ;;
266   esac
267 done
268
269 if [ ! -d "$src/unzipped_files" ]; then
270   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files\r"
271   sleep 0.75
272   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files.\r"
273   sleep 0.75
274   echo -ne "${YELLOW}Creating temporary folder for
        unzipping Zipped files..\r"
275   sleep 0.75
276   echo -e "${YELLOW}Creating temporary folder for
        unzipping Zipped files...\r"
277   sleep 0.5
278   mkdir -p "$src/unzipped_files"
279 fi
280
281 #unzipping the zipped folders
282 for file in $(find "$src" -type f -name "*.zip"); do
283   # Unzip the files
284   echo -e "${GREEN}Unzipping `basename $file` ${CYAN}"
285   unzip -q "$file" -d "$src/unzipped_files"
286 done
287
288 total_files=$(find "$src" -type f -not -path '*/\.*' |
      wc -l)
289 start_time=$(date +%s)
290 current_file=0
291 echo
292 echo -e "${YELLOW}Copying the files now..."
293 echo -e "${RED}"
294 sleep 0.5
295
296 #first I took file from the source then piped it to the
      sed command
297 #to get the files which have an extension
298 for file in $(find "$src" -type f -not -path '*/\.*')
299
300 do
301   #check whether the user wants a progress bar or not
```

```
302   if [ $disable_progress = "false" ]; then
303     # Update progress
304     ((current_file++))
305     progress=$((current_file * 100 / total_files))
306 
307     # Display progress bar
308     print_progress "$progress"
309   fi
310 
311   #exctracted basename from file
312   name=`basename $file`
313 
314   #get whether to sort about date or extension
315   ext=`get_ext $file $sort_type`
316   #now we check whether we need to copy the file or not
317   copy_files="true"
318   if [ $enable_exclude = "true" ]; then
319     for f in "${exclude_list[@]}"
320     do
321       if [ $f = ${name##*.} ]; then
322         copy_files="false"
323       fi
324     done
325   fi
326 
327   if [ $enable_include = "true" ]; then
328     copy_files="false"
329     for f in "${include_list[@]}"
330     do
331       if [ $f = ${name##*.} ]; then
332         copy_files="true"
333       fi
334     done
335   fi
336 
337   if [ $enable_max_size = "true" ];then
338     file_size=$(stat -c %s "$file")
339     if [ $file_size -gt $size_arg ]; then
340       copy_files="false"
341     fi
342   fi
343 
344 
```

```
345   if [ $copy_files = "true" ]; then
346
347     #make extension folders to copy files
348     echo $ext >> extensions.txt
349     if [ ! -d "$dest/$ext" ]; then
350       echo $ext>>added_folders.txt
351     fi
352     mkdir -p "$dest/$ext"
353
354     #creating the logfile
355     if [ $create_logfile = "true" ]; then
356       if [ -f $log_file  ]; then
357         rm $log_file
358       fi
359       echo "`get_available_filename $file $dest $ext`
             moved from $src to $dest/$ext" >> $log_file
360     fi
361
362     #copying the file
363     echo `get_available_filename $file $dest $ext` >>
          moved_files.txt
364     cp $file $dest"/"$ext"/"`get_available_filename
          $file $dest $ext`
365
366     #delete the moved files if option was given
367     if [ $delete = "true" ]; then
368       rm $file
369     fi
370
371   fi
372 done
373 echo -e "${NC}"
374 echo
375
376 cat extensions.txt|sort|uniq > extensions.txt
     #created a file containing all the extensions
377 cat added_folders.txt|sort|uniq > added_folders.txt
378 #time to print the Summary and other user friendly
     messages
379 sleep 0.5
380 echo -e
     "${RED}--------------------------SUMMARY-----------------------------
381 sleep 0.5
```

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

```
422 rm moved_files.txt added_folders.txt extensions.txt
423 echo -e "${RED}Removing temporary folder for unzipping
    Zipped files${NC}"
424 sleep 0.5
425 rm -r "$src/unzipped_files"
426
427 #trap - ERR
```