End-To-End walk through and explanation of rpiClassify.py

Produced by: Alan Feria & Dr. Alexander White '22

alan214@ucsb.edu

Table of Contents

About This File

This file was created for the benefit of all teachers and students wanting to use the work

located on (link to SCI github)

SECTION 1

About this Document

This document serves as a procedural guide for various processes, detailing explicitly how tasks have been accomplished and how to navigate mass_rename.sh. This software and document is written for students and mentors of the UCSB-Smithsonian Scholars Program.

If at any moment you feel that there is any ambiguity, lack of explanation, or uncertainty, please refer to section 7 of this document as it provides an in-depth explanation of most functions, modules, and packages that are utilized throughout the Shell executable.

1.1 A Quick Breakdown of Unix

Unix Shell and Bash is a scripting languages used to interact with an operating system and do terminal-based tasks, but it can basically do anything a simple programming language can do; it's like C but for the terminal. A shell script utilized Unix commands combined with the ability to declare relatively simple variables (primarily strings). In our case, we will use Unix Shell scripting to automate the execution of repetitive tasks so that we (humans) do not need to perform them individually. Bash scripting also supports variables, conditional statements, and loops like programming languages which we will utilize to our advantage.

SECTION 2

A Brief Overview

This section will give a brief walkthrough of what mass_rename.sh does and the purpose of this Shell executable. For a more in-depth explanation, please reference section 3 and onward.

2.1 Locating Parent Directory

In order to use this shell executable, we must be in the correct directory. In our case, the correct directory would be the SD card 'DCIM' folder; I will be referring to this directory as the 'parent directory' throughout this instructional document. To switch over to this directory, we can use the cd command.

To **change our directory to the parent directory**, we can use the following steps.

- Right-click on the parent directory folder
- Select 'copy path'.
- Open terminal
- Within the terminal window, type 'cd', right-click, and select 'paste'.

Once we have changed our working directory, we should see a static message on your terminal window (seen below). This signifies that you are currently working from within the correct parent directory.

---

pi@raspberrypi: DCIM $

2.2 Renaming Files With mass_rename.sh

Now that we are within the correct parent directory, the Shell executable with do a couple of things in the background, and depending on the number of files present on the SD card, the time will vary.

1. **Reading in parent directory**: The first thing we must do is ask the user for the parent directory. This is done to avoid errors when running the code. This can be done by executing the following steps.

    - Right-click on the parent directory folder
    - Select 'copy path'.
    - Open terminal
    - Within the terminal window, right-click and select 'paste'.

2. **Validating Specified Parent Directory**: Up next, we will use a simple while loop, which will return false if the specified parent directory doesn't exist and return true if the specified parent directory exists. This while loop will not break until the parent location 'read_location' is correctly specified and validated.

3. **Saving Directory Path**: Once the path has been validated, we will save it to the envVars.csv file to use later.

4. **Specifying mass_rename.sh File Path**: The next thing that will be to take in the file path for mass_rename.sh. To this, we will use a one-liner that obtains that will take the current working directory of the running file and append the file name (mass_rename.sh). This will result in the following line being saved as a variable called 'mass_rename_direct'.

5. **Unique Name for Files**: Since we want to unique rename each file, we have to to the file name in a way that will uniquely identify each image. To do so, we will rename the files with specific variables that will be **INPUT BY THE USER**. If we want four variables, we type in '4', and if we want three variables attached to the file name, we type in '3', and so on. See below for an example representation of this explanation.

Before User Input

```
How many variables do you want to include in your file name: 4
Enter variable names #1:
Enter variable names #2:
Enter variable names #3:
Enter variable names #4:

Enter year value:
Enter camera Location value:
Enter camera ID value:
Enter trip number value:
```

After User Input

```
How many variables do you want to include in your file name: 4
Enter variable names #1: year
Enter variable names #2: camera Location
Enter variable names #3: camera ID
Enter variable names #4: trip number

Enter year value: 2022
Enter camera Location value: UCSB02
Enter camera ID value: 2B
Enter trip number value: 01
```

6. **Exporting Variables**: These variables will then be saved and exported as a string separated by commas, as seen below.

```
2022_UCSB02_2B_01
```

   This string of variables will now be used to name every file; this is done to uniquely rename all files created from this specific sd card. As a result, the files will share the same schema and will be easily identifiable.

7. **Debugging Mode**: Next, the mass_rename.sh will ask us if we want to run in debugging mode or do a straight run without out debugging. To be clear, 'debugging mode' means that it will let you preview the file names and variables input before actually running; if you choose not to run in debugging mode, the code will run, and you will see the output once all is finished regardless of if there is an error or not.

   If you choose to run in debugging mode, you will see 15 random entries printed out, as shown below.

File Name Preview

```
2022_UCSB02_2B_01_img_00002.jpg
2022_UCSB02_2B_01_img_00249.jpg
2022_UCSB02_2B_01_img_00132.jpg
2022_UCSB02_2B_01_img_01032.jpg
2022_UCSB02_2B_01_img_00014.jpg
2022_UCSB02_2B_01_img_02028.jpg
2022_UCSB02_2B_01_img_00809.jpg
2022_UCSB02_2B_01_img_01204.jpg
2022_UCSB02_2B_01_img_32049.jpg
2022_UCSB02_2B_01_img_32890.jpg
2022_UCSB02_2B_01_img_00032.jpg
2022_UCSB02_2B_01_img_00232.jpg
2022_UCSB02_2B_01_img_00102.jpg
2022_UCSB02_2B_01_img_00934.jpg
2022_UCSB02_2B_01_img_00139.jpg
```

Once you see what is in the file name, the code will again ask if this is correct. If it isn't correct, then you will restart from step 3 (shown above) else wise, you will continue to the next part.

2.3 Saving Our Work

1. **Extracting Metadata with ExifTool**: To comply with Camera Trap DP standards, we must extract all the metadata from the images. To accomplish this, we will use the ExifTool on all our .jpg files located on the sd card. Once that has been done, you will see a message saying:

```
Now we are creating CSV and saving it to the sd card
```

2. **Saving Important Key/Value Pairs**: Along the way, we created many variables and specified many paths that have been validated. It is important to save these path variables so we can use them down the line without typing them out again or repeating any unnecessary steps. It is important to note that we will be using these variables within rpiClassify.py. We will save the variables to envVars.csv, a CSV file that was automatically created at the beginning of this executable.

SECTION 3

A More Technical Explanation

This section serves as a procedural guide for various processes, detailing explicitly how tasks have been accomplished and how to navigate mass_rename.sh so future interns and users of this software can understand and navigate it with ease.

3.1 Locating mass_rename.sh File Path

While executing this script, you will be allowed to restart all over if you make a mistake when assigning your variables or read location. However, to restart, we must again call the script. To do this, we must locate the file path to mass_rename.sh. To do this, I created a one-liner that extracts the absolute path to the mass_rename.sh file.

```
mass_rename_direct=$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &>
    /dev/null && pwd )"/mass_rename.sh"
```

3.2 String Of Variables

In order to comply with Camera Trap Data Package standards, we must rename the files using a schema that gives each file a unique name. To do this we append n number of variables to the original file name and rename. To do this, we use the following while loop to create a unique string of variables which is then attached to the original file name, ultimately resulting in a unique file name.

1. This while loop will start off will begin by creating two empty arrays, one called vars and one called values. 'vars' will hold the keys, and 'values' will hold the values of each key.

```
vars=()
values=()
```

2. Now the user will be prompted to input the number of variables that they want to add to the file. if the user wants to add five variables the user should input '5' and '4' if they want to add only four variables.

```
read -p "How many variables do you want to include in your file
    name: " cvar
```

3. Having specified the number of variables, we must now enter the key and values. To do this, we will first input the keys by looping over 'cvar'. 'cvar' is equal to the number of variables that the user decided to add to the name in the previous step.

```
loop=1
while [ $loop -le $cvar ]
do
    read -p "Enter variable names #${loop}: " vars[$loop]
    loop=$((loop+1))
done
```

4. Now that we have input the keys, all that is left is the input of the values; we will repeat the same steps mentioned above, except this we will input the values to each key rather than the keys themselves.

```
loop=1
while [ $loop -le $cvar ]
do
    read -p "Enter ${vars[loop]} value: " values[$loop]
    loop=$((loop+1))
done
```

5. The last step to creating this string of variables is to join them all with an underscore ('_') separating all of them. This is done with the following line of code.

```
separator="_"
foo=$values

regex="$( printf "${separator}%s" "${values[@]}" )"
regex="${regex:${#separator}}"
```

6. lastly, as a safety measure we will ask the user to confirm that the string of variables is correct and that the variables aren't in the wrong order or miss spelled. To accomplish

this, we will use a while loop that will only break if the user confirms that the string of variables is correct.

```
read -r -p "Does this look correct, $regex " response
```

3.3 Specifying the Parent Directory

In order to use parameter expansion we must specify the parent directory from which we will start our search from later on. Since we previously set the working directory we can simply use PDW to get the correct wording directory.

```
read_location=${PWD}
```

3.4 Renaming Images

This is now the final step; we will utilize all the previous work to rename all the images uniquely. The user will be asked to run in either two mode. The first mode is debugging mode, and the other is without debugging. To be clear, 'debugging mode' means that it will let you preview the file names and variables input before actually running; if you choose not to run in debugging mode, the code will run, and you will see the output once all is finished regardless of if there is an error in your variables or not.

```
read -r -p "Would you like to run this function in debug mode? [y/N] "
    response
```

If you **select debugging mode**, you should expect to see the following; you will see 15 random entries printed out with the string of variables attatched to it. Which will be produced by the following line of code.

```
i=1
case "$response" in
    [yY][eE][sS]|[yY])
            a=0
            while [ $a -lt 15 ]
            do
                dir=("${read_location}"/*/*.jpg)
                img_num="${dir[a]##*_}"
                printf '%q\n' "${destination}${regex}_${img_num}"
                a=`expr $a + 1`
            done
```

1. We will start by setting the counter variable 'i' to one. 'i' is used to numerically renumber

all the images in each subdirectory from one to i, where i is the total number of images present on the SD card.

2. Now we will enter the while look since we selected yes to debugging mode, this while look contains another counter variable that is initiated at zero and has a max of fifteen. For each iteration the while loop will display the renamed file name as shown below

File Name Preview

```
2022_UCSB02_2B_01_img_00002.jpg #first random exmaple displayed
2022_UCSB02_2B_01_img_00249.jpg
2022_UCSB02_2B_01_img_00132.jpg
.
.
.
2022_UCSB02_2B_01_img_00139.jpg #fifteenth random example displayed
```

Again you will be asked input either yes or no. Here you will input 'yes' if what was displayed above looks correct and 'no' if does not look correct.

```
read -r -p "If this looks correct, enter [y/N] to run with previous
    entries " yesorno
```

If you enter 'yes' you will find yourself moving moving onward and if you enter 'no' you will be sent back to the very beginning and will start all over and be asked to 'Try again an be sure to check your variables twice'.

3. Since you have entered 'yes' you will rename and renumber all the images on the SD card from one to i, where i is the total number of images present on the SD card, by using the following line of code.

```
case "$yesorno"
in
    [yY][eE][sS]|[yY])
        for dir in "${read_location}"/*; do
                [ -d "${dir}" ] || continue
                for img in "${dir}"/*.jpg; do
                        [ -e "${img}" ] || break
                        new="$(printf ""${regex}"_img_%05d.jpg" "${i}")"
                        mv "${img}" "$(dirname "${img}")/${new}"
                        ((i++))
                done
        done
        ;;
```

This chunk of code does the following:

(a) Uses parameter expansion to find all the subdirectories/files (i.e. read_location/RECNYX100)

(b) for each subdirecotry/file that is found we make sure that the path as a directory and not a file by using the '-d' flag on 'dir'.

(c) If the path is validated as a directory we can continue and repeat steps a-b execpt this time we will be searching for files rather than directories by replacing the '-d' arguement with '-e' to ensure that the path exist.

(d) Now that we have completed all the preliminary steps we can create the new file name 'new'. 'new' is a a combination of the string of varibales ('regex') and the new image number 'img_%05d'. Here the '05d' part will ranging from 00001 to i where i is equal to the total number of images present on the SD card.

(e) Now that we have created a new file name ('new') we can use the mv command to rename the file. The mv command is going grab file 'img' and replace the old name with the new name ('new'). please reference the illustration below for a better understanding.

> Old name: img_1.jpg
>
> New name: 2022_UCSB02_2B_01_img_00001.jpg

4. These steps will be repeated for all the images present on the SD card.

3.5 Extracting Metadata with ExifTool

Again we will be using Camera Trap DP to guide our work. The next step that needs to be taken is to extract the meta data for all of the images present on throughout the SD card. To do this we will use the wildcard matching to from within the current working directory to find all file that end in .jpg and subsequently create a CVS file with all the entries. This CVS file will be name following the same schema as the images except it will begin with 'metaData' and end with the string of variables

```
exiftool -csv */*.jpg > "metaData_${regex}.csv"
```

If everything was done correctly we you will see the following message and the file prodcued should follow a schema similar to the one shown below.

> "Now we are creating csv and saving it to the sd card"
>
> Example: metaData_2022_UCSB02_2B_01_img_00001.jpg

3.6 envVars.csv

Throughout this executable, we created a large number of variables such as the string of variables used to rename the images and so on. Since we already input them once we will export this variables via a CSV file so that user of this program will not have to re-enter the variables once more. This is a good practice as it will avoid human error and any possible mistake when running rpiClassify.py

To create and save this CSV we will use the following lines of code.

```
#using same metadata to create csv name

echo "Variables=Values" >> envVars.csv

echo "read_location=${read_location}" >> envVars.csv

echo "mass_rename_direct=${mass_rename_direct}" >> envVars.csv

echo "exifFileName=metaData_${regex}.csv" >> envVars.csv

echo "fileNamesMeta=${regex}" >> envVars.csv
```

If these file paths/names were all saved correctly we should see the following message.

> "Metadata, and variables csv have been saved to $read_location"

Here $read_location is SD card parent directory which was specified above.

4

SECTION

Understanding Our Work

In summary, mass_rename.sh is an executable shell script that utilizes the camera traps sd card to **rename all images** present on the sd card and subsequently produce two CSV files, one named 'metaData....csv' and another named 'envVars.csv' which has a custom delimeter of '='.

4.1 envVars.csv

envVars.csv will contain the following columns: 'variables' & 'values'. Within the variables column we should expect to find the many of the variables created such as 'read_location', 'mass_rename_direct', and 'regex' which is the string of variables we created earlier. The'values' column will store the values for for each entry in the variables columns. Please reference the table below for a better understanding.

variable	Value
read_location	/Volumes/NO NAME/DCIM1
regex	2022_UCSB02_2B_01.csv
mass_rename_direct	/Desktop/SCI_skunks-main/Shell_scripts/mass_rename.sh

4.2 metaData...csv

Metadata...csv is a CSV file which contains meta about the images on the SD card. The length of CSV file should be equal to the total number of the images on the SD card. This file name will always begin with metaData and end with the string of variables created at the beginning of the this script.

5

SECTION

Data Scientist Tool Box - Unix Shell

5.1 cd Unix Shell Command

The change directory (cd) command is built into the system shell and changes the current working directory. This command can take in a relative path or an absolute path.

Syntax: cd <directory>

5.2 mv Unix Shell Command

Using the mv command with its default syntax allows you to rename a single file, move files or move and rename files.

Syntax: mv [options] [current file name] [new file name]

5.3 read Unix Shell Command

read command in the Linux system is used to read from a file descriptor. To us, it serves as a method to read in user input for various variables as 'read_location'.

Syntax: read [option]

5.4 echo Unix Shell Command

echo command in Linux is used to display a line of text/string passed as an argument. At a junior level, it is good practice to use echo to see what your code will be doing before actually running it.

Syntax: echo [option] [string]

5.5 printf Unix Shell Command

the printf command is used to display the given string, number, or any other format on the terminal window. It works the same way as printf works in programming languages like C.

Syntax: $printf [-v var] format [arguments]

5.6 expr Unix Shell Command

The 'expr' command in Unix evaluates a given expression and displays its corresponding output. It is used for:

- Basic operations like addition, subtraction, multiplication, division, and modulus on integers.
- Evaluating regular expressions, string operations like substring, length of strings, etc.

Syntax: expr expression

5.7 pwd Unix Shell Command

The pwd command is a command line utility for printing the current working directory. It will print the full system path of the current working directory to standard output.

5.8 Bash Parameter Expansion

A 'parameter' would be something like a string or variable set by the user or shell script itself. Parameter expansion is used to modify, add, or extract certain parts of a parameter.
For a more in-depth explanation and examples, about parameter expansion, please visit linux-hint.com

5.9 Parent and Child Directory

'Parent Directory' refers to the directory above another directory. Every directory, except the root directory, lies beneath another directory. The higher directory is called the parent directory, and the lower directory is called a subdirectory or child directory.

5.10 Absolute Path

An absolute path is defined as specifying the location of a file or directory from the root directory(/). In other words, we can say that an absolute path is a complete path from start of the actual file system from the / directory.

5.11 Relative Path

The relative path is defined as the path related to the present working directly(pwd). It starts at your current directory and never starts with a / .

5.12 Flags

There are several different types of objects than can live in a file system. Regular files and directories are just two of them; others include pipes, sockets, and various device files. Here's an example using pipes to illustrate the differences between the three options. -e just checks if the named argument exists, regardless of what it actually is. -f and -d require its argument to both exist and be of the appropriate type.

```
$ mkfifo pipe  # Create a pipe
$ mkdir my_dir # Create a directory
$ touch foo.txt # Create a regular file
$ [ -e pipe ]; echo $? # pipe exists
0
$ [ -f pipe ]; echo $? # pipe exists, but is not a regular file
1
$ [ -d pipe ]; echo $? # pipe exists, but is not a directory
1
$ [ -e my_dir ] ; echo $? # my_dir exists
0
$ [ -f my_dir ] ; echo $? # my_dir exists, but is not a regular file
1
$ [ -d my_dir ] ; echo $? # my_dir exists, and it is a directory
0
$ [ -e foo.txt ] ; echo $? # foo.txt exists
0
$ [ -f foo.txt ] ; echo $? # foo.txt exists, and it is a regular file
0
$ [ -d foo.txt ] ; echo $? # foo.txt exists, but it is not a directory
```

6

SECTION

mass_rename.sh

6.1 Source Code

```
mass_rename_direct=$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &>
    /dev/null && pwd )"/mass_rename.sh"

vars=()
values=()

read -p "How many variables do you want to include in your file name: "
    cvar
loop=1
while [ $loop -le $cvar ]
do
    read -p "Enter variable names #${loop}: " vars[$loop]
    loop=$((loop+1))
done
loop=1
while [ $loop -le $cvar ]
do
    read -p "Enter ${vars[loop]} value: " values[$loop]
    loop=$((loop+1))
done

#echo "${values[*]}"
```

```
#echo "${vars[*]}"

separator="_"
foo=$values
regex="$( printf "${separator}%s" "${values[@]}" )"
regex="${regex:${#separator}}"

read -r -p "Does this look correct, $regex " response

while :
do
    case "$response"
     in
        [yY][eE][sS]|[yY])
                break
                ;;
        *)
            read -p "How many variables do you want to include in your file
                name: " cvar
            loop=1
            while [ $loop -le $cvar ]
            do
                read -p "Enter variable names #${loop}: " vars[$loop]
                loop=$((loop+1))
            done
            loop=1
            while [ $loop -le $cvar ]
            do
                read -p "Enter ${vars[loop]} value: " values[$loop]
                loop=$((loop+1))
            done

            #echo "${values[*]}"
            #echo "${vars[*]}"

            separator="_"
            foo=$values
            regex="$( printf "${separator}%s" "${values[@]}" )"
            regex="${regex:${#separator}}"
        esac

done


mkdir $regex
```

```
read -r -p "Would you like to run this function in debug mode? [y/N] "
    response


i=1

case "$response" in
    [yY][eE][sS]|[yY])
            a=0
            while [ $a -lt 15 ]
            do
                dir=("${read_location}"/*/*.jpg)
                img_num="${dir[a]##*_}"
                printf '%q\n' "${destination}${regex}_${img_num}"
                a=`expr $a + 1`
            done
        read -r -p "If this looks correct, enter [y/N] to run with previous
            entries " yesorno
        case "$yesorno"
        in
            [yY][eE][sS]|[yY])
                for dir in "${read_location}"/*; do
                        [ -d "${dir}" ] || continue
                        for img in "${dir}"/*.jpg; do
                                [ -e "${img}" ] || break
                                new="$(printf ""${regex}"_img_%05d.jpg"
                                    "${i}")"
                                mv "${img}" "$(dirname "${img}")/${new}"
                                ((i++))
                        done
                done
                ;;
            *)
                echo "Try again, be sure to check your variables twice."
                #/home/pi/Downloads/SCI_skunks-main/Shell_scripts/mass_rename.sh
                    #change this to the where the local file will be stored
                    on RPI
                bash "${mass_rename_direct}"
                ;;
            esac
        ;;
    *)
        for dir in "${read_location}"/*; do
```

```
                [ -d "${dir}" ] || continue
                for img in "${dir}"/*.jpg; do
                        [ -e "${img}" ] || break
                        new="$(printf ""${regex}"_img_%05d.jpg" "${i}")"
                        mv "${img}" "$(dirname "${img}")/${new}"
                        ((i++))
                done
        done
        ;;
esac

exiftool -csv */*.jpg > "metaData_${regex}.csv"

echo "Now we are creating csv and saving it to the sd card"

#using the same metadata to create csv name

echo "Variables=Values" >> envVars.csv

echo "mass_rename_direct=${mass_rename_direct}" >> envVars.csv

echo "exifFileName=metaData_${regex}.csv" >> envVars.csv

echo "fileNamesMeta=${regex}" >> envVars.csv

echo "Metadata, and variables csv have been saved to $read_location"
```

End