MATLAB® Program in Unix Platform

Naeem Khoshnevis
Center for Earthquake Research and Information
University of Memphis, Memphis, TN

June 26, 2016

1 Introduction

Data processing is an important step of many scientific studies. Type and size of data as well as the type of processing can determine the processing method. In seismological studies, we mostly deal with numerical data with different customized processing. In many cases, users write a function in a programming language (e.g. MATLAB®, Python, C, Fortran) and process the data. Depending on the type of data and processing, we may need several steps of processing data in a project. Also we may have to change a parameter and repeat the processing several times. Writing down the processing steps is a good practice to avoid confusion, keep track of the processing, and be able to return back and look for the probable bugs in case of wrong results. However, since it is not automated, you may unintentionally skip one step of processing or forget to properly document it. Even with complete documentation, in the case of finding the problem in processing steps, you need to repeat the whole process and go through all the steps.

Developing a data analysis package could be an excellent alternative to applying customized processing functions on massive data sets. In this short note, I present how to develop a data processing package. This note is not about the methods of data processing, or getting and cleaning data, or even programming. It presents how to use shell script to make your process faster, more accurate, and trackable. If you have scripts in a different programming language and you are using those scripts repeatedly to process data, you may find this document helpful. The idea of having a data processing package is not novel. People mostly define such a structure in their projects. The goal of this note is to make this process easy to understand.

Shell scripts (e.g. Unix shell) are robust computer program for handling files and folders. In this short note, I explain how to use shell scripts to mange the input and output files, and report any problem during the processing if necessary. The core function could be written in any programming language. Extra steps are required to use MATLAB in unix, so the following example show how to do this. However, the method is applicable to other programming languages with lower complexity. In summary, shell script prepares data with a required format and name at the right time and right place to the core function and receive the results at the right time and right place to pass to other program or archive it.

Having a basic knowledge of programming in MATLAB and shell is necessary to understand this note. In the next two sections we will discuss the basic commands of shell and MATLAB.

2 Basic Shell Commands

Here is the list of important shell commands for the project. You may need other commands, this depends on the complication of your project.

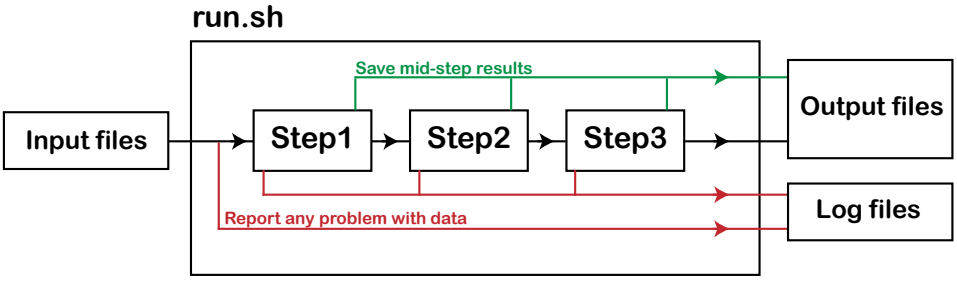

Figure 1: Schematic of a data processing package.

Command	Description
mkdir	make a directory
ls	list directory contents
cd	change the directory
pwd	return working directory name
cp	copy files
mv	move files
rm	remove directory entries
cat	concatenate and print files
head	display first lines of a file
tail	display the last part of a file
grep	file pattern searcher
echo	write arguments to the standard output
paste	Merge corresponding or subsequent lines of files
sed	stream editor
chmod	change file modes or access control lists

The basic shell commands are simple and easy to use. The beauty of the shell script is allowing the user to combine the commands together for complicated processes. Combining commands or passing data between commands happens by understanding the concept of "pipe, standard input," and "standard output." Generally, all commands print out some results into the screen. Standard input is the source of input data which can come from the keyboard, a file, or a standard output of another program. Sometimes, instead of obtaining the input from the keyboard or from a file, a program can use the output of another program as its input. This is accomplished through the use of a "pipe." The vertical line "|" is what is known as pipe.

Here is an example of using the pipe (|) to pass data among three different commands. The example is related to seismic data. Much of seismic data comes with a header. Generally, the header is a summary of information about the data (e.g. length, station location, components, ...). Following is the part of data from Southern California Earthquake Data Center at Caltech. The data format is COSMOS V1. The time increment of the data is indicated by "dt=" and it is located at the end of the line.

```
...
3 Comment lines(s) follow, each starting with a '|':
|Station: CI ADO HNE 34.5505 -117.4339 908.00 4.0978e+03
| Record: start= 2008/07/29,18:41:59.721 len= 338.630 ( 33864 pts) dt= 0.0100
| Event: 14383980 2008/07/29,18:42:15.710 33.9530 -117.7613 14.70 5.39
33864 velocity pts. approx 338 secs, units=cm/sec (05),Format=(6E12.4)
5.9990e-03 3.8027e-03 -1.0595e-02 -5.8995e-04 6.7311e-03 -1.5661e-03
4.5348e-03 4.2908e-03 -6.6908e-03 -3.2743e-03 8.4394e-03 -2.0542e-03
-9.1312e-03 5.5109e-03 6.4871e-03 -3.4591e-04 -1.5661e-03 5.2669e-03
...
```

The following script can extract the dt value from the file. First we read the file and pipe it to the `grep` command to find those lines with "dt". We pipe the result into the `awk` command and print out the last column (NF = number of fields).

```
cat SeismicData.V1 | grep "dt=" | awk '{print $NF}'
```

3 MATLAB®

MATLAB® is short for MATrix LABoratory, and is a common high-level tool for numerical analysis in science and engineering. Writing a program in MATLAB is much farther from the scope of this note. However, there are some basic command that is necessary in developing the package (e.g. `load, pwd, Try Catch, strrep, sprintf, save, saveas`). Obviously, the core function could have a different level of complexity based on the project. Assuming you already have your script to process the data, you will want to know how to use the current functions and scripts frequently or in combination with other scripts.
In unix, running programs which are written in C++, Python, Fortran, ... does not need any further settings. However, running MATLAB scripts is different from other programs. The procedure is explained well in MATLAB documentation (see MATLAB(Unix)).In the data processing package, we are interested in the following options in running the MATLAB application:

- **-r "command"** starts MATLAB and executes the specified MATLAB command
- **-nodisplay** do not display any X commands, and ignore the DISPLAY environment variable
- **-nosplash** starts MATLAB but does not display the splash screen during startup

In Unix environment `matlab` script runs the MATLAB functions and scripts. `matlab` is a Bourne shell script that starts the MATLAB executable on UNIX® platforms. Unix shell

needs to have access to the `matlab` script to run the application. On Apple Macintosh platforms, the `matlab` script is located inside the MATLAB application package. For example in my computer `matlab` is located inside the following path:

```
/Applications/MATLAB_R2013b.app/bin
```

You need to add the path, to the current path to be able to run the `matlab` script. Run the following command in terminal to export the path of `matlab` script path to the current path:

```
export PATH=/Applications/MATLAB_R2013b.app/bin:$PATH
```

Every processing package needs at least one shell script (.sh file) to call MATLAB codes. The main tasks of this script are:

- **Preparing the data as an input for MATLAB:** read the data from input folder, change the format if it is necessary (e.g. remove the header), and rename the file (MATLAB will load files with specific name.)
- **Running the MATLAB code:** now that we know the input data is ready, the script will run the code.
- **Processing the MATLAB code results:** MATLAB will write the results in the right place with a specific name. Shell script gets the file and renames it and will do other processing or will simply save it.

4 Data Processing Package - Structure

Each data processing package has four folders (`Input_files, Output_files, Log_files,` and `Functions_scripts`) and one file (`run.sh`). We can change the whole package name and transfer it to another location in the memory, or other computers. Once we choose a name to the internal folders, we can't change the name of the folders. This is because of using these folder names for handling the path. Each file and folder has a specific functionality.

`Input_files` folder includes all files (or folders) that we want to process. These files can be raw data, list of filenames, or results from other projects. `Output_files` folder contains all files that we produce, it could be text files, figures, pdf files, and temporary files. `Log_files` folder keeps files that help us to track the program's operation. It could be the history of runs (like, login info and time of running), report the success and failure for each set of data, and summary of the processing. `Functions_scripts` has all functions and scripts of MATLAB and Shell (or any other programing languages). `run.sh` is a shell script file that basically calls the other scripts and perform the whole processing sequence.

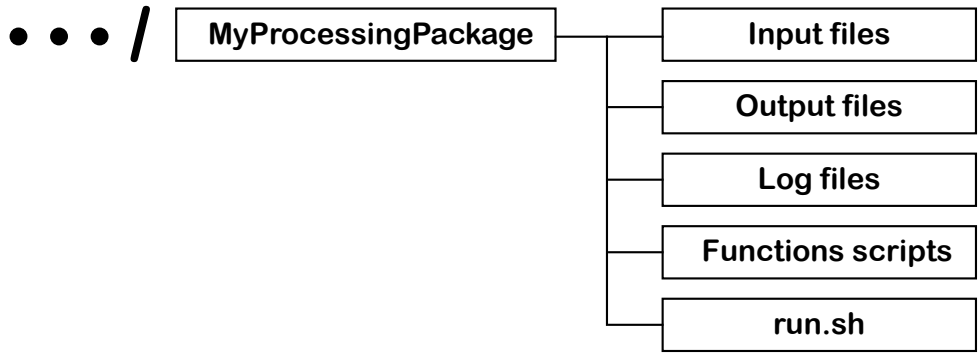

Figure 2: File and folders of the data processing package.

This file is the right place to define necessary parameters to expand the functionality of the processing package. For example we can ask the program to keep data of all steps, report only a final pdf file, or ignore some steps. `Log_files` and `Output_files` are the only folders in which we can add, remove, or modify data. Even the core programs read data from `Output_files` folder. We will discuss more about this in the following sections.

5 Handling the path

The most important factor in developing a processing package is handling the path. In each step of programming, we need to know, where we are, where we want to be (in terms of computer path), where the input data is and where we need to print the output data. We accept the following axioms:

- We can run any program, script, and function in the memory, if we know the location of the program.
- We can load any data into the program, if we know the location of the data.
- We can write any data in any part of memory, if we know the destination path.

Since we run the **run.sh**, the program has access to the current path (using `pwd`). Then we can navigate to other folders (since we know the folder names) through combining the folder name with the current path.
In the next section, we develop a data processing package in six easy steps. The main subject of the following example is understanding how to handle the input and output data and learning how to work with path. The files of the example package are attached to this document. However, it is strongly recommended to generate the files step by step to learn developing and testing a data processing package for other projects.

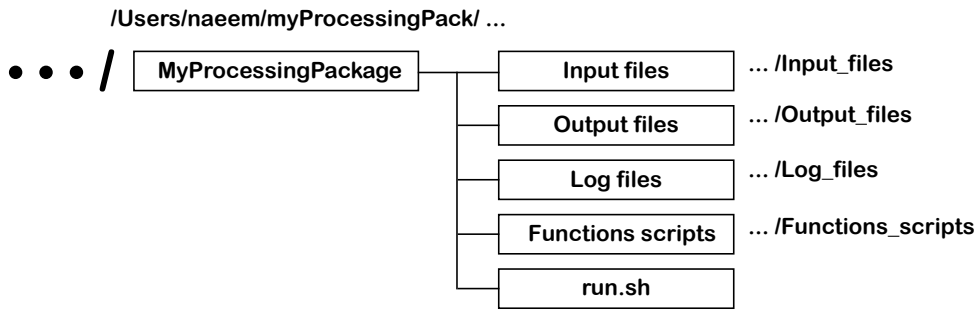

Figure 3: Relative path of folders.

6 Setup a data processing package

In many cases, using MATLAB in Unix improves the efficiency and accuracy of the project (considering the fact that it eliminates human intervention). However, for a massive processing, it may be a good idea to write the core function in other programs (e.g. C++, Fortran). The following steps are necessary for all data processing packages regardless of the core function. Let's setup a data processing package to get students grades and compute the average grade for each student. We are using simple example t to get the whole structure without adding complexity in each step. The input file has some number of grades of different courses. We want to write a program to get the student's file and write student's file name, number of courses, and average grade in the output file. We also want to make sure that the data of each student, processed successfully or if there was a problem in processing, what it was.

Step 1: Creating the folders and the file

The data processing package will have 4 folders and 1 file. Create a folder and name it as `student_grades`. Inside the folder create `Input_files, Output_files, Log_files, Functions_scripts`, and `run.sh`.

```
> mkdir student_grades
> cd student_grades
> mkdir Input_files Output_files Log_files Functions_scripts
> touch run.sh
> ls
Functions_scripts Log_files run.sh
Input_files Output_files
```

Step 2: Writing the MATLAB function (or script) to process one data.

We need to make sure that the core function works properly (It will be hard to debug

the program if the core function doesn't do what it is supposed to do). At step two, the core function should be able to read the data from the same directory, process the data, and write the results at the same directory. Let's say we have one of the input files (`David.txt`) in `Functions_scripts` folder (at this step we test the core function, at the final version there should not be any data file in this folder). Here is the input file content:

```
> head David.txt
4.00
4.00
2.84
4.00
3.32
4.00
3.94
2.15
```

Now we want to write the core function. In the processing package, each MATLAB program has three major sections: **Loading data**, **Processing data**, and **Saving results**. The MATLAB code (`matlab_st_grade.m`) is according to below:

```
%% Load the input file

student_grade = load('David.txt');

%% Process data

size_data = size(student_grade,1);
sum_data = sum(student_grade);
av_data = sum_data ./ size_data;
grade_mat = [size_data av_data];

%% Save the results

file_id = fopen('output.txt','w');
fprintf(file_id,'%d\t%2.3f',grade_mat(1,1),grade_mat(1,2));
fclose(file_id);
```

If the input file is in the same directory as the script, upon running the script, we will see the `output.txt` file at the same folder. Take a close look at the results, and make sure the program works well (It will save you a considerable amount of time if you fix the bugs at this step). This script is almost complete, however, there are three major differences between this version and the final version. In the final version:

- The input and output file will not be in this directory.
- The name of input and output file will be same for all data.

In my projects, I never use MATLAB to produce the final results. In that case you need to bring in the file name, which will add unnecessary complication to the project. You can ask shell script to rename, merge, or ... the final result.

Step 3: Generating the shell script to run the MATLAB code.

In this step we are writing a shell script to run the MATLAB code. Note that shell script does not control where MATLAB is supposed to read the data or write the data, it just runs the code. In `Function_script` folder we create a file and name it `average_grade.sh`. Since we are in the `Function_script` folder and both MATLAB and Shell script are inside this folder, at this point we don't need to handle the path. The `average_grade.sh` is according below:

```
#!/bin/sh
matlab -nodisplay -nosplash -r "matlab_st_grade,quit"
```

The first line sets the shell environment. The second line calls the MATLAB application (make sure you exported the MATLAB path as previously mentioned in MATLAB section), then tell the application to run the `matlab_st_grade.m` script (does not need .m) and quit the program. So far everything is set up. Here is the summary of what we have done so far.

- We have a MATLAB code which does some processing. However, it reads input and output from the same directory. ⟶ We need to modify it to read data from and write the results into `Output_files`.
- The MATLAB code reads a file by exact file name. ⟶ It should read the file by the name that we assign to the file.
- We have a shell script that runs the MATLAB. ⟶ We need to modify the script to also provide the input file for MATLAB and sets the output file for other scripts or save it.

Step 4: Modifying the MATLAB code to r/w from/into `Output_files` folder

Since we are in the `Functions_scripts` folder, the `pwd` prints out the working directory ending in `Functions_scripts`. We know, in the processing package we have also three other folders (`Input_files`,`Output_files`, and `Log_files`). If we substitute the `Functions_scripts` in the path with each one of input and output folder names, we will have access to those folders. Also we want to read a file with the name `temp_grade_file.txt` (Shell script will generate this file, but for now, to test the program, we generate it manually). The modified `matlab_st_grade.m` file is below:

```
  %% Load data
% Load the input file
path1=pwd;
path1 = strrep(path1,'Functions_scripts','Output_files');
file1='/temp_grade_file.txt';
cfile1 = [path1 file1];

student_grade = load(cfile1);

%% Process data

size_data = size(student_grade,1);
sum_data = sum(student_grade);
av_data = sum_data ./ size_data;
grade_mat = [size_data av_data];

%% Save the results

%write the results in to the file.
path2 = strrep(path1,'/Functions_scripts','/Output_files');
file2 = '/temp_grade_mat.txt';
cfile2 = [path2 file2];

file_id_2 = fopen(cfile2,'w');
fprintf(file_id_2,'%d\t%2.3f',grade_mat(1,1),grade_mat(1,2));
fclose(file_id_2);
```

In order to test the code, create a file in `Output_files` folder and name it `temp_grade_file.txt`. Now open the MATLAB program and run the command (You can run it through shell script, too). It should create the `temp_grade_mat.txt` file inside the `Output_files` folder. Up to this point, the MATLAB script is capable of reading `temp_grade_file.txt` file and producing `temp_grade_mat.txt`. Now we need to provide these files to MATLAB program through shell script. We need to modify the `average_grade.sh` to read data from `Input_files` folder, create a temporary file (`temp_grade_file.txt`) in `Output_files` folder, run the program, and get the output file (`temp_grade_mat.txt`) and put the re-

sult in the final output file. The modified `average_grade.sh` is according to below:

```
  #!/bin/sh
my_file=David.txt

cd ../Input_files

current_path=‘pwd‘
script_path=‘echo $current_path| sed -e ’s/Input_files/ Functions_scripts/g’‘

cat $my_file > ../Output_files/temp_grade_file.txt

matlab -nodisplay -nosplash -r \
"try,run $script_path/matlab_st_grade,catch,end,quit"

number_of_grades=‘cat ../Output_files/temp_grade_mat.txt | awk ’{print $1}’‘
average=‘cat ../Output_files/temp_grade_mat.txt | awk ’{print $2}’‘
echo $my_file ’Number of grades:’  $number_of_grades \
              ’Average score:’   $average >> ../Output_files/Final_grade.txt
```

Make sure you have the `David.txt` file in the `Input_files` folder. Go to the `Functions_scripts` folder and run the `average_grade.sh`. First we define the `my_file` name variable and switch the path to `Input_files` folder. Being in the `Input_files` folder, we retrieve the current directory path and generate the `Functions_scripts` folder's path. Using `cat` command we generate a `temp_grade_file.txt` inside the `Output_files` folder. We run the MATLAB code (Note that we are using run command, because the script path is not in the current MATLAB path) and extract the number of grades and average values from the result. Using `echo` command, we write the results into the `Final_grade.txt` file. Here we used "`>>`" symbol to append the value.

Now we have a programming package with the following characteristics:

- We have a MATLAB code which reads data from `Output_files` folder and generates results there.
- We have a shell script that can generate an input file for the MATLAB script, run the MATLAB script, and process the results. It seems good. However, that shell script is good for one file, where as we want to create a processing package to process as much file as we want. At the next step, we will modify the `average_grade.sh` script to take care of numerous files.

Step 5: Modifying the `average_grade.sh` script to read all data in the `input_files` folder

In simple programs, the `run.sh` file, commonly runs another script in the `Functions_scripts` folder, however, it is good to have it, since it can decrease the complexity for user. If you set up the `run.sh` properly, the user will only need to execute the run.sh file, without getting too much involved in details about the program. For this project the `run.sh` file is according below:

```
#!/bin/sh
# Program Name:  Students Final Grade
# Written by:  Naeem Khoshnevis
# Last update:  Jun 5, 2016


# ------------------------------------------------------------------
# --------------------- Input files format -----------------------
# student.txt:
# 3.5
# 4
# 3.2
# .


# ------------------------------------------------------------------
# ---------------------- Change the directory ----------------------

cd Functions_scripts

# ------------------------------------------------------------------
# ---------------------- Run the program ---------------------------

./average_grade.sh ../Input_files
```

We set up the `run.sh` file. You can put as many details about the program (as comments) in this file, as you want. Now we need to modify the `average_grade.sh` script. The file should be able to run all the input files (based on a special pattern) from the `Input_files` folder and process them. In this project we assume all input files are `.txt` files. Here is the modified version of the `average_grade.sh` script.

```
#!/bin/sh
# This function call the MATLAB script to take the average of students grades.
# results will be at the output_files folder

# go in to the first variable after the ./xx.sh command.
# In this case is ../Inpute_files.

cd $1

for i in $(ls *.txt)

do

current_path=‘pwd‘
script_path=‘echo $current_path| sed -e ’s/Input_files/Functions_scripts/g’‘

cat $i > ../Output_files/temp_grade_file.txt

matlab -nodisplay -nosplash -r \
"try,run $script_path/matlab_st_grade,catch,end,quit"

number_of_grades=‘cat ../Output_files/temp_grade_mat.txt | awk ’{print $1}’‘
average=‘cat ../Output_files/temp_grade_mat.txt | awk ’{print $2}’‘
echo $i ’Number of grades:’  $number_of_grades \
             ’Average score:’   $average >> ../Output_files/Final_grade.txt

done

rm ../Output_files/temp*
```

The package is now complete. You may put as many students' grade files in the `Input_files` folder and get the results at the `Output_files` folder. The `Final_grade.txt` file for five students could be similar to the following:

```
> cat Final_grade.txt
Amy.txt      Number of grades:  31 Average score:  3.306
David.txt    Number of grades:  19 Average score:  3.387
John.txt     Number of grades:  28 Average score:  3.408
Matt.txt     Number of grades:  22 Average score:  3.334
Rob.txt      Number of grades:  22 Average score:  3.334
```

The package processes the data properly. However, there is a potential problem. If some data is not as what we assumed, we will never be informed about it. In the next step, we

modify the code to be able to report the potential problems in the data.

Step 6: Reporting warnings and producing run summary

Let's say in this example, which we are finding the average grade, and grades supposedly should be 4 or less, mistakenly grade of one course recoded as 4.5, or we have negative grades, or the format of the data is not what we expected. In these cases, we need to make sure that, the program report the files with problems. These warning reports goes to the `Log_files` folder. In general, we can consider following steps to make sure about the warnings:

- Define a flag for a problem. For example, if the data loads properly and does not have any problem, use **0**, if the data is not in format and MATLAB can not load it, use **1**, if it has a grade greater than 4 use **2**, in the case of negative grade use **3**, and if we have both out of range grades use **5** .
- Save the flag in a file (for example `temp_status_file.txt`)
- Load the flag in the shell script and prepare the report based on the flag type.

We start with the MATLAB script. We need to define the conditions and save the results. Here is the modified MATLAB code (`matlab_st_grade.m`).

```
%% Load data
% Load the input file

path1=pwd;
path2 = strrep(path1,'Functions_scripts','Output_files');
file1='/temp_grade_file.txt';
cfile1 = [path2 file1];


try
      student_grade = load(cfile1);
      student_grade = student_grade(:);
      status_temp = 0;
catch
      status_temp = 1;
end

if status_temp == 0
```

```
        % control for numbers bigger than 4.
        if any(student_grade >4) == 1
            status_temp = 2;
        end

        % control for negative numbers.
        if any(student_grade < 0) == 1
            status_temp = status_temp + 3;
        end
end

% Write the results into a file.
status_file_name = sprintf('%s%s',path2,'/temp_status_file.txt');
file_id = fopen(status_file_name,'w');
fprintf(file_id,'%d',status_temp);
fclose(file_id);

if status_temp ~= 0
        break;
end

%% Process data

size_data = size(student_grade,1);
sum_data = sum(student_grade);
av_data = sum_data ./ size_data;
grade_mat = [size_data av_data];

%% Save the results

%write the results in to the file.
path2 = strrep(path1,'/Functions_scripts','/Output_files');
file2 ='/temp_grade_mat.txt';
cfile2 = [path2 file2];

file_id_2 = fopen(cfile2,'w');
fprintf(file_id_2,'%d\t%2.3f',grade_mat(1,1),grade_mat(1,2));
fclose(file_id_2);
```

Now we have another temporary file (`temp_status_file.txt`) which shows the status of the file. We need to load this file into the shell script and create a report based on the flag. Here is the modified version of the `average_grade.sh`

```
#!/bin/sh
# This function call the MATLAB script to take the average of students grades.
```

```
# results will be at the output_files folder

# go in to the first variable after the ./xx.sh command.
# In this case is ../Input_files.

cd $1

for i in $(ls *.txt)

do

current_path=‘pwd‘
script_path=‘echo $current_path| sed -e ’s/Input_files/Functions_scripts/g’‘

cat $i > ../Output_files/temp_grade_file.txt

matlab -nodisplay -nosplash -r \
"try,run $script_path/matlab_st_grade,catch,end,quit"

status_mode=‘cat ../Output_files/temp_status_file.txt |awk ’{print $1}’‘

case $status_mode in
0) status_report=‘echo Processed successfully.‘;;
1) status_report=‘echo Unable to load.‘;;
2) status_report=‘echo There is at least one grade greater than 4.‘;;
3) status_report=‘echo There is at least one negative grade.‘;;
5) status_report=‘echo There are out of range grades.‘;;
esac


if (( $status_mode > 0 )); then

echo ’File ’ $i $status_report >> ../Log_files/grade_run_status.txt

else

echo ’File ’ $i $status_report >> ../Log_files/grade_run_status.txt
number_of_grades=‘cat ../Output_files/temp_grade_mat.txt | awk ’{print $1}’‘
average=‘cat ../Output_files/temp_grade_mat.txt | awk ’{print $2}’‘
echo $i ’Number of grades:’  $number_of_grades \
             ’Average score:’   $average >> ../Output_files/Final_grade.txt

fi

done

rm ../Output_files/temp*
```

```
# ------------------------------------------------
# ------- run summary file ---------

date_t=‘date‘
user="$USER"

echo $user ’ran the program on’ $date_t >> ../Log_files/run_summary.txt
```

For example, the `grade_run_status.txt` from running the package for some input file could be as following:

```
 File Amy.txt Procecessed successfully.
File Andy.txt There are out of range grades.
File David.txt Procecessed successfully.
File John.txt There is at least one negative grade.
File Matt.txt Procecessed successfully.
File Philip.txt There is at least one grade greater than 4.
File Rob.txt Unable to load.
```

The summary of the run could be a good place to give a short report about the run for your future reference, for example, those parameters that you change in each run. For this example, I just print out the user and time of running the processing package. Here is an example of the `run_summary.txt` for the processing package:

```
naeem ran the program on Mon Jun 6 211:20:40 CDT 2016
```