Lab 2

Disk Management and Storage Management

CSCI 344 Shell Programming

Due Friday February 3, 11:59pm on Canvas
30 points

OBJECTIVE

The objective of this lab is get more practice with scripting using bash on the Unix environment. In addition to developing scripts, this lab and future labs will tackle common Unix administration tasks. Several Unix features and utilities will be covered in class/lab that can be applied to meet the requirements of this lab. These include:

1. Disk usage management
2. Associative array storage structure
3. File processing storage structure
4. Error checking
5. Sorting data

A **storage structures** will need to be implemented such that the data is only gathered ONCE and thereafter, different queries can be applied on the stored data. The storage structures covered in this lab pertain to the usage of an array and simple text files. The UNIX account ***uid*** should be used to index the array.

Our end goal is to determine who the users are which are using the most disk space on a Unix system. An **example** on how this lab is as follows. Generate and ***store*** a list of usernames and their respective disk usage. Then, ***remove*** the usernames having less than 8Mb from your storage structure and display the results in ***sorted*** order thus, **generating** a simple high usage report.

INCREMENTAL STEPS

To help write the code, follow the incremental step approach as shown below. This will allow you to make progress on your design.

Step 1: Generate a large file

Since this lab assignment analyzes the disk space usage of the student accounts on our AWS instance, we want to have a variety of data. Hence, as discussed in class, every student should generate a large file under his/her $HOME directory. The size of the file should have a size of X MB such that X is the number of letters of the student's first and last name. For example, student with name Abe Lincoln would need to create a file with size of 10MB (3 letters in Abe and 7 letters in Lincoln). Include a **screen shot** that shows your implementation. Hint: use the UNIX **dd** command or Google the Internet for other similar commands.

Step 2: Generate a list of all UNIX usernames on the AWS instance

Since we need to determine the disk usage for every username, we'll need a list of all username that we can prosses as a queue within a while loop. So the first thing is to generate the list. Try using the *ls* command and redirecting standard output to a file.

NOTE: On our AWS instance, by using the ls command as stated above, it will include some garbage data. For example, in my case my list included an entry with the contents of "lost+found". As you proceed with this lab, this "lost+found" entry will **need to be handled programmatically**. It is not right to simply vi the file and delete this line. I addressed this by testing the username value with the id command and checking the return *$?* status value. A non-zero return value represents an error. For example, the following command returns an error status.

```
id -u lost+found
if [ $? -ne 0 ]
then
   echo "error"
fi
```

When is the lost+found directory created and what is its purpose?

Step 3: Determine disk usage per username

Here is the command we went over during lab time to determine the disk usage.

```
$ du -s /home/jim 2>/dev/null > fout
```

Are you able to explain every parameter in the above command?

NOTE: The du command saves the results to file fout. You may take this approach or pipe the output to the next parsing step. I had to parse the 1-line entry by piping the output through the following *awk* command and filtering out the first parameter value.

```
awk '{printf $1}'
```

The awk utility is frequently used to parse data in the Unix world. The above is just a small example and more is discussed during class time.

Now that we have discussed some of the basics, lets discuss the array data structure method.

Step 4: Storing data in array

Let's discuss first the array data storage structure and the terminology we will use. You are all familiar with an example code to store an array value such as the following:

```
myarray[5] = 2000
```

Let's refer to the value of 5 as the **key** (aka index) and 2000 as the **value**.

Next, during lab time we discussed that every UNIX user account has a unique **id** value. To store the array, your goal is to use the user id as the **key** and the amount of disk storage usage as the **value** for your array. For example, with the username jim on our AWS instance, its user id is 1001 and suppose username jim disk space usage is 2000 bytes. Hence, the store command would be : myarry[1001] = 2000.

Step 5: Save the data in an array

To add data into an array, the array structure is defined as:

declare -A myarray

Then, given a key/value pair, the array is loaded as:
myarray[$key]=$value

The contents of the array can be checked using the following loop:

```
for i in "${!myarray[@]}"
do
  echo ${myarray[$i]}" "$i
done
```

Now that we discussed the array data structure storage, recall that you will also need to implement a file storage data structure.

Step 6: Save the data in a file

Above we went over the steps to save data into an array. Recall that this lab also requires that you implement your data storage by using simple text files. Above we mentioned how the array values can be examined with a loop. A while loop can also be used to loop thru the contents of a file.

Once you have the data stored, whether it is in an array or in files, you will need to implement the UNIX sort command.

Step 7: Sort the data

Depending on your implementation, the usage of the sort command can vary. More will be discussed during lab time.

What is the difference between sorting by numerical and ASCII value. What is the default using the UNIX sort command and how can you specify which method is desired?

Step 8: Search the data

For either storage structure used (i.e. array or file), perform the following search.

a) Search your data storage to identify the users having a usage of less than 8MB and remove them from your storage structure. Do not simply re-gather the data. The data should already have been gathered and stored.
b) Display **only** the top 3 users and the respective disk usage amount of those utilizing the most disk space.

Lab submission

For this lab and all future labs, it is important to clearly present the work done. If you are only able to complete the lab partially, clearly state it. Explain where you time was spent and any hurdles that were or were not met. Given the steps mentioned above, your deliverables are the following:

1. Include all the code within your write-up.
2. Paste as many screen shots as possible that describe your work for each step into one continuous document. Your screen shot should indicate **proof** that the code worked as specified.
3. Spend much time on adding your personal comments within your lab submission. Refer to syllabus and example as shown in class. Make sure to include embedded comments within your lab write-up. Anyone should be able to read your lab write-up and understand what it is trying to present. DO NOT SIMPLY TAKE SCREEN SHOTS. Include highlights and arrows to describe specific points.
4. Add a conclusion section to your write-up. Elaborate on at any aspect of your lab. Some examples are: what was challenging, how were you able to overcome any hurdle(s), and/or what did you learn.
5. *** Please refer to the required format specified in the syllabus (i.e. turn in **one** PDF/Word/OpenOffice document). No individual image files. And no zipped, no compressed and no rtf-formatted files.

Review the following document posted on Canvas under the Misc content area.

SampleLabWriteup : This is a good example on the lab write-up format. Notice the different embedded highlights and comments.

GRADING

To achieve a maximum score, students will need to clearly prove that they completed the goal. A clear description of the steps taken and screen shots are essential. Partial credit is given if students can clearly state what was done and what is not working. If the instructor is required to decipher incompleteness, much less partial credit will be given.

Points lost for incompleteness, sloppiness, lateness, or failure to follow instructions.

Late policy: refer to syllabus