File System

2016-17, CSCI 3150 - Assignment 4

Release: 17 Nov 2016

Deadline: 1 Dec 2016 11:59AM

Contents

1 Introduction 2

1.1 Add . . . 2

1.2 Retrieve . . . 2

1.3 Diff . . . 2

1.4 Calculate . . . 3

2 Your assignment 12

2.1 Submission . . . 13

2.2 Grading . . . 14

2.2.1 Part A: Correctness (90 marks) . . . 14

2.2.2 Part B: Bonus (20 marks) . . . 14

2.3 To begin . . . 14

2.3.1 Usage of grader.sh . . . 14

2.4 Your job . . . 19

2.5 Notes . . . 20

2.6 Late Submission Policy . . . 20

3 Questions 21

4 Academic Honesty 21

1 Introduction

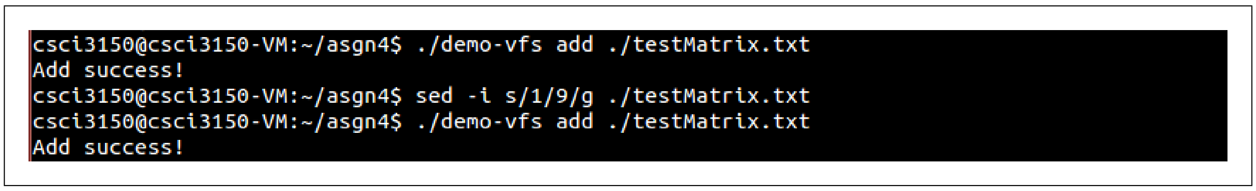
In this assignment, you are going to implement a simple version file system, namely, `vfs`, using C/C++. This `vfs` is specialized designed for matrixes. Each file stores an $n \times n$ matrix. The `vfs` keeps all versions of all files being added. It supports four commands.

1.1 Add

Usage: `vfs add <file path>`

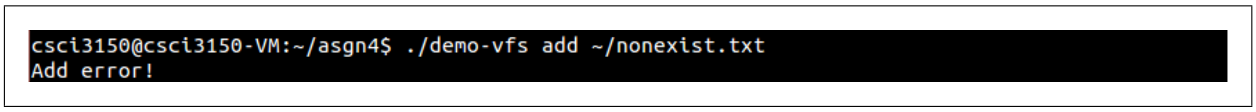
Use this command when you want to backup a local file to the `vfs`. Output `Add success!` if this operation is success or output `Add error!` otherwise. The first time the file is added to the `vfs`, that is version 1 of the file. The second time the file is added to the `vfs` using that command, that is version 2 of the file. Your `vfs` shall keep all versions of all files being added.

Examples:



```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs add ./testMatrix.txt
Add success!
csci3150@csci3150-VM:~/asgn4$ sed -i s/1/9/g ./testMatrix.txt
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs add ./testMatrix.txt
Add success!
```

Figure 1: Adding a file "testMatrix.txt" to the `vfs`. Afterwards, modify it a bit. Then, the new version is added to the `vfs` again.



```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs add ~/nonexist.txt
Add error!
```

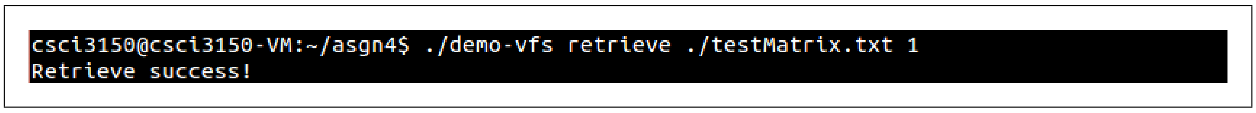
Figure 2: Adding a file that doesn't exist causes an error

1.2 Retrieve

Usage: `vfs retrieve <file path> <version number>`

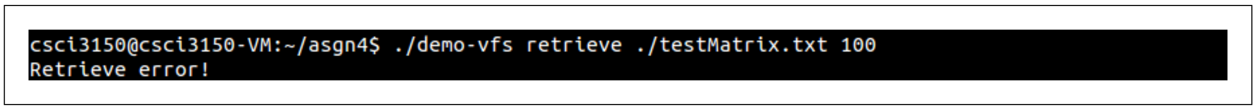
Use this command when you want to retrieve a previous version of your file. When retrieving, the current working copy of the file will be **overwritten** by the version that you retrieve from `vfs`. If the retrieval is successful, `vfs` shall output `Retrieve success!`, else output `Retrieve error!`.

Examples:



```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs retrieve ./testMatrix.txt 1
Retrieve success!
```

Figure 3: Retrieve version 1 of file `testMatrix.txt` from the `vfs` successfully. The working copy now becomes version 1 of the file.



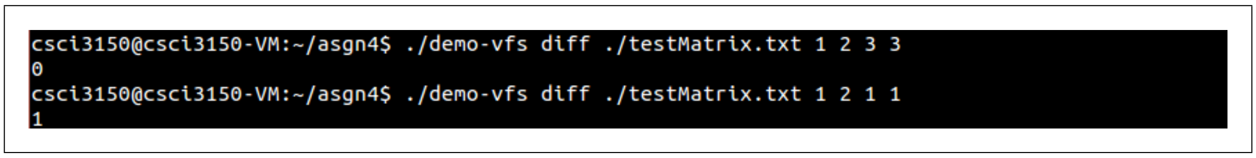
```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs retrieve ./testMatrix.txt 100
Retrieve error!
```

Figure 4: Retrieving a file/version that doesn't exist causes an error.

1.3 Diff

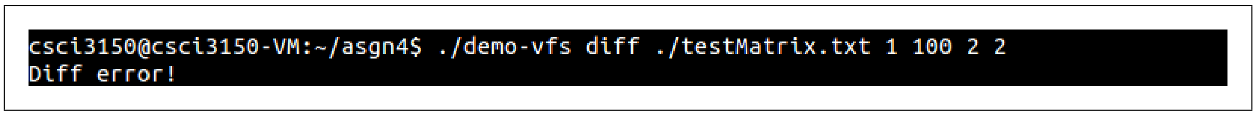
Usage: `vfs diff <file path> <version 1> <version 2> <row> <column>`

Use this command when you want to know whether a particular element in the matrix is the same across two versions of the same file. Outputs `0` when the elements of the two versions are the same, output `1` otherwise. On any error, outputs `Diff error!`.



```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs diff ./testMatrix.txt 1 2 3 3
0
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs diff ./testMatrix.txt 1 2 1 1
1
```

Figure 5: [First command] Comparing the elements at row 3 column 3 of the matrix stored in `testMatrix.txt`. Returning `0` means the two values of the same element are same among versions 1 and 2.



```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs diff ./testMatrix.txt 1 100 2 2
Diff error!
```

Figure 6: Return error when trying to `diff` against a version (version 100th) that doesn't exist.

1.4 Calculate

Usage: `vfs calculate <file path> <version 1> <version 2> <type> <row> <column>`

Use this command when you want to do some simple matrix calculations across the same row and column between two versions of a matrix. There are 3 types of calculation.

Suppose `./testMatrix.txt` is a 3×3 matrix as follows.

$$\underset{\text{testMatrix.txt version1}}{\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}} \qquad \underset{\text{testMatrix.txt version2}}{\begin{pmatrix} 9 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix}}$$

- `-r`

  It calculates the sum of differences between the corresponding elements in the row that contains the element at position `[<row>,<column>]` among the two versions.

  For example, `./vfs calculate ./testMatrix.txt 1 2 -r 3 2` calculates the sum of differences of row 3 in version 1 and version 2 of "testMatrix.txt". The result is $(7-7)+(8-8)+(9-1)=8$.

  Note: order matters, so the result of

  `./vfs calculate ./testMatrix.txt 1 2 -r 3`

  could be different from the result of

  `./vfs calculate ./testMatrix.txt 2 1 -r 3`.

  Note: In this command, the `<column>` value is ignored.

- `-c`

  Same as `-r` except it calculates the sum of differences between the corresponding elements in the column that contains the element at position `[<row>,<column>]` among the two versions.

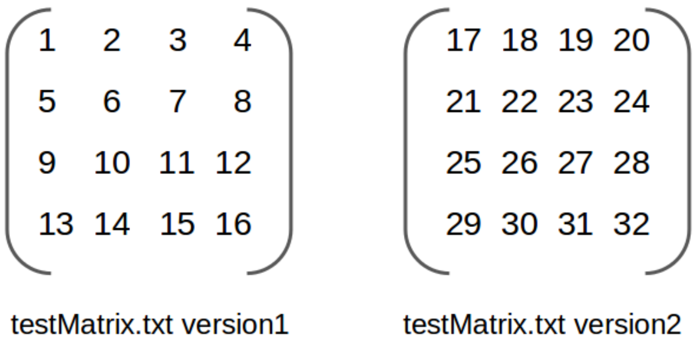
For example, `./vfs calculate ./testMatrix.txt 1 2 -c 3 2` calculates the sum of differences of column 2 in version 1 and version 2 of "testMatrix.txt". The result is $(2-2)+(5-5)+(8-8)=0$.

Note: In this command, the `<row>` value is ignored.

- `-a`

  It calculates the sum of differences between the corresponding elements in the specified sub-matrix among the two versions. The sub-matrix consists of all elements surrounding the element at position `[<row>,<column>]`.

  Suppose `./testMatrix.txt` is a 4×4 matrix as follows.



  Then, `./vfs calculate ./testMatrix.txt 1 2 -a 2 3` calculates the sum of differences of sub-matrix M1 (from version 1) and M2 (from version 2) centered at position `[2,3]`.

$$M1 = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \\ 10 & 11 & 12 \end{bmatrix}$$

  and

$$M2 = \begin{bmatrix} 18 & 19 & 20 \\ 22 & 23 & 24 \\ 26 & 27 & 28 \end{bmatrix}$$

So, the result is $(2 - 18) + (3 - 19) + \ldots. + (11 - 27) + (12 - 28) = -144$

As another example, `./vfs calculate ./testMatrix.txt 1 2 -a 4 1` calculates the sum of differences of sub-matrix M3 (from version 1) and M4 (from version 2) centered at position `[1,4]`.

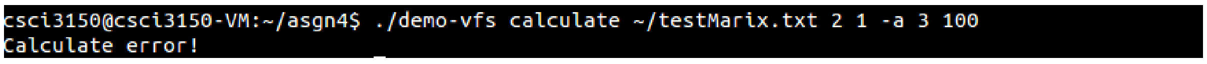
$$M3 = \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}$$

and

$$M2 = \begin{bmatrix} 19 & 20 \\ 23 & 24 \end{bmatrix}$$

So, the result is $(3 - 19) + (4 - 20) + (7 - 23) + (8 - 24) = -64$

On any error, this command shall output `Calculate error!`.



```
csci3150@csci3150-VM:~/asgn4$ ./demo-vfs calculate ~/testMarix.txt 2 1 -a 3 100
Calculate error!
```

2 Your assignment

You are given the following files:

Name	Description
`/asg4-func.c`	Code skeleton for you (**Work on it**).
`/asg4-vfs.c`	Framework of VFS. (**Don't touch**).
`/Makefile`	Makefile. If needed, you can write your own Makefile and submit it. If your makefile can not work correcttly, than you will get 0 marks.
`/vfs`	Executable but not functioning. (**Runnable on your 32-bit virtual machine**)
`/demo-vfs`	Executable, functioning, serves as the baseline to check the correctness as well as the performance. (**Runnable on your 32-bit virtual machine**)
`/testMatrix.txt`	A simple matrix for demonstration.
`/.vfsdata`	A **hidden** directory. Put your versioned data into this directory. You need to create it before running VFS (**Don't change to other directory or otherwise you will get labeled as cheating.**)
`/testcases`	All testcases use the input files under this directory
`/grader.sh`	We will run this script to grade your assignment (**Don't touch**).

2.1 Submission

You are required to submit one source code `asg4-func.c` to eLearning. If needed, you can also submit your `Makefile` to eLearning. You must make sure your `Makefile` works, otherwise, you get 0 marks.

Warning: DON'T change the file names, otherwise you get 0 marks

2.2 Grading

2.2.1 Part A: Correctness (90 marks)

30 test cases. A test case is essentially a script that executes a sequence of content edits and `vfs` operations. Each test case will test if your `vfs` is functioning properly or not. Passing one test case will get 3 marks.

2.2.2 Part B: Bonus (20 marks)

A versioned file system shall care about both space and time efficiency. It shall use less space to store all copies if possible but it shall not sacrifice the operation performance either. Bonus will be awarded to the following students.

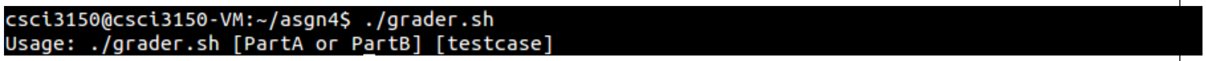
- 10 marks: [Test case P1] If the total space under `/.vfsdata` used by your `vfs` is smaller than the TA's demo.
- 10 marks: [Test case P2] If the total running time of your `vfs` command in test case P2 is faster than the TA's demo (Cache will be cleaned before the measurement).

(The TA's demo is not of top quality indeed)

2.3 To begin

2.3.1 Usage of grader.sh

Run `grader.sh`, you shall see something like this:

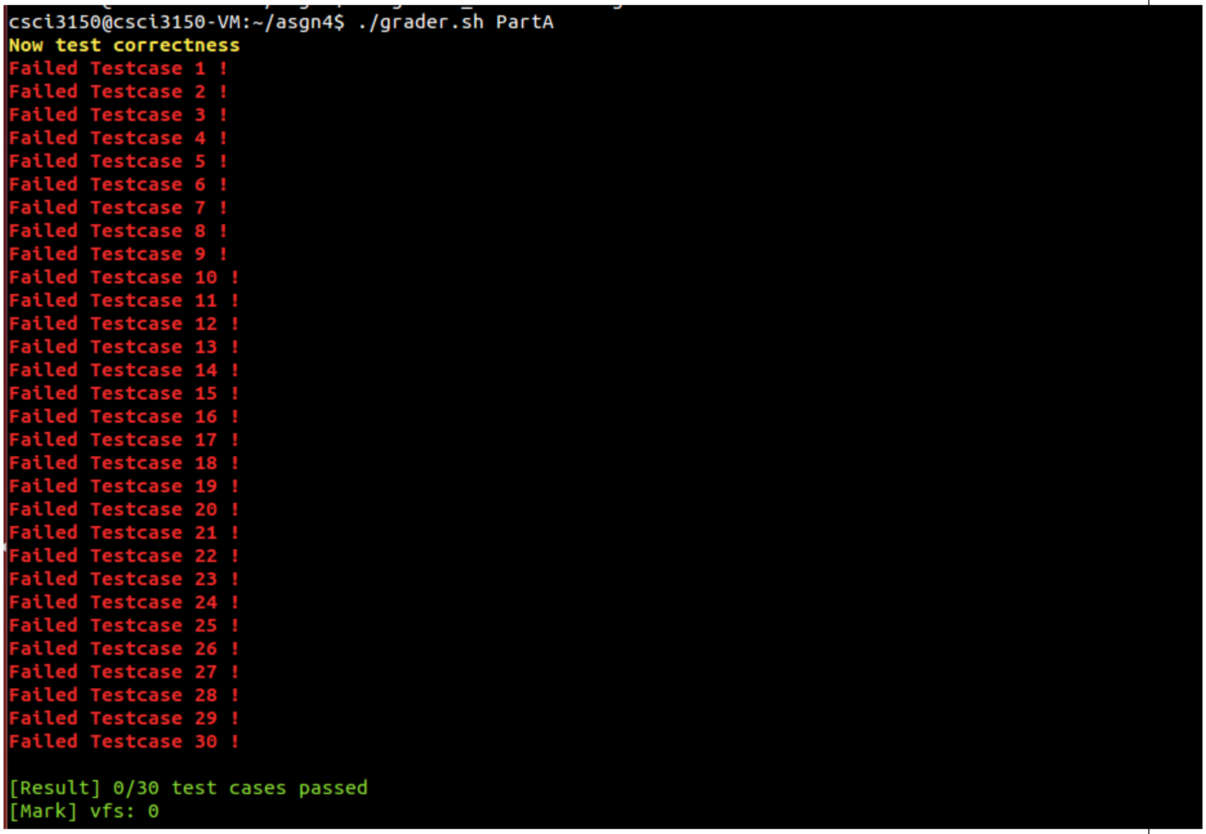


```
csci3150@csci3150-VM:~/asgn4$ ./grader.sh
Usage: ./grader.sh [PartA or PartB] [testcase]
```

- **Run PartA Test Cases**

  PartA test cases check the correctness. To run all PartA test cases, you should use the following command.

```
./grader.sh PartA
```

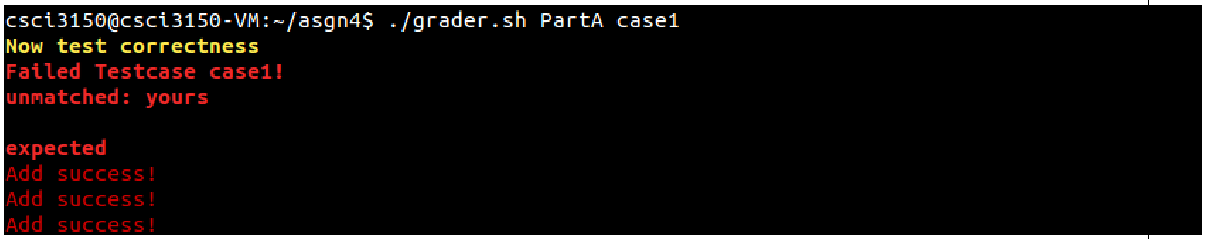


```
csci3150@csci3150-VM:~/asgn4$ ./grader.sh PartA
Now test correctness
Failed Testcase 1 !
Failed Testcase 2 !
Failed Testcase 3 !
Failed Testcase 4 !
Failed Testcase 5 !
Failed Testcase 6 !
Failed Testcase 7 !
Failed Testcase 8 !
Failed Testcase 9 !
Failed Testcase 10 !
Failed Testcase 11 !
Failed Testcase 12 !
Failed Testcase 13 !
Failed Testcase 14 !
Failed Testcase 15 !
Failed Testcase 16 !
Failed Testcase 17 !
Failed Testcase 18 !
Failed Testcase 19 !
Failed Testcase 20 !
Failed Testcase 21 !
Failed Testcase 22 !
Failed Testcase 23 !
Failed Testcase 24 !
Failed Testcase 25 !
Failed Testcase 26 !
Failed Testcase 27 !
Failed Testcase 28 !
Failed Testcase 29 !
Failed Testcase 30 !

[Result] 0/30 test cases passed
[Mark] vfs: 0
```

If you want to run a specified PartA test case, for example `case1`, use command like this:

```
./grader.sh PartA case1
```

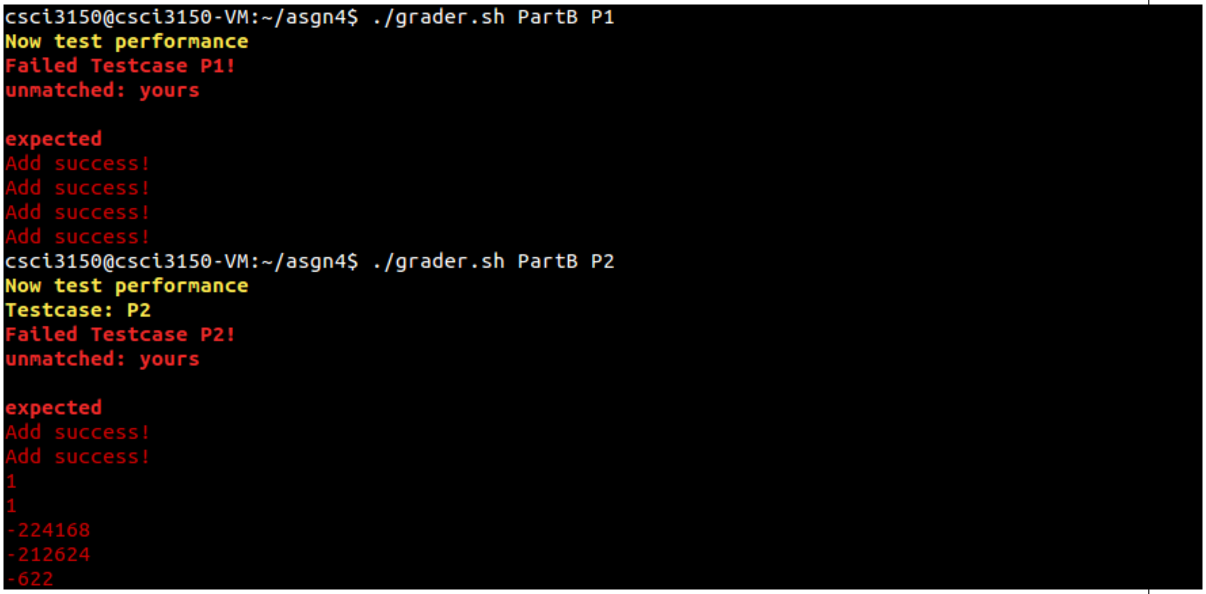


```
csci3150@csci3150-VM:~/asgn4$ ./grader.sh PartA case1
Now test correctness
Failed Testcase case1!
unmatched: yours

expected
Add success!
Add success!
Add success!
```

- **Run PartB Test Cases**

  To run PartB test cases, you must specify the name of test case. PartB has two test cases, P1 and P2, where P1 measures the space your VFS use and P2 measures the time of your `vfs` operations. For example, to run P1, we use command like this:

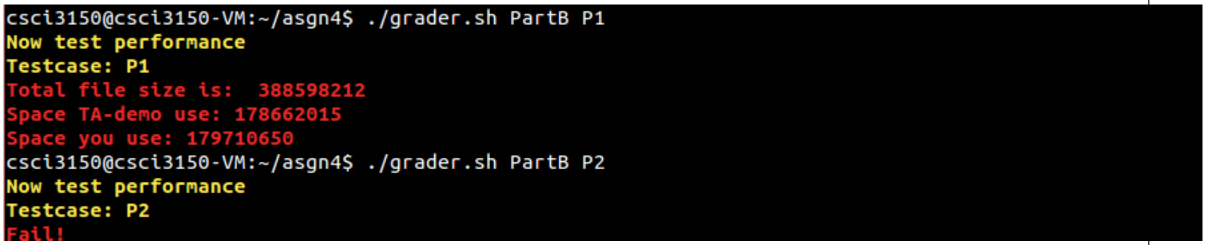


```
csci3150@csci3150-VM:~/asgn4$ ./grader.sh PartB P1
Now test performance
Failed Testcase P1!
unmatched: yours

expected
Add success!
Add success!
Add success!
Add success!
csci3150@csci3150-VM:~/asgn4$ ./grader.sh PartB P2
Now test performance
Testcase: P2
Failed Testcase P2!
unmatched: yours

expected
Add success!
Add success!
1
1
-224168
-212624
-622
```

  After you finish the assignment, when you grade it with PartB test cases, you will see something like this:



```
csci3150@csci3150-VM:~/asgn4$ ./grader.sh PartB P1
Now test performance
Testcase: P1
Total file size is:  388598212
Space TA-demo use: 178662015
Space you use: 179710650
csci3150@csci3150-VM:~/asgn4$ ./grader.sh PartB P2
Now test performance
Testcase: P2
Fail!
```

2.4 Your job

Your job is to start from the given `asg4-func.c` and pass as many test cases as possible. In addition, your program shall do a good balance between space and time to get the bonus marks.

2.5 Notes

1. One matrix per file. The test case won't change the dimension (i.e., the number of rows/columns) of the matrix but only modify the elements in-place.

2. The input file however contains only one long line with each value and row separated by a space. You have to do certain parsing by yourself. Remember: all matrixes are $n \times n$ matrixes.

3. The `vfs` keeps all versions of a file and there are no file deletion or file overwriting in `vfs`. So, if you already have 5 versions of "testMatrix.txt" in the `vfs` and you just retrieve the 2nd version back as the working copy. If you once again add that working copy back to the `vfs`, that will be the 6th version of the file.

4. Store all your (versioned) data under `/.vfsdata`. Storing versioned data elsewhere is regarded as cheating and penalty will be imposed.

5. Each test case will clear your `/.vfsdata` directory before testing.

6. The grading will be fully automated. The grading platform is our course's VM: Ubuntu 14.04 32-bit.

7. Once we start the grading processing, we reserve the right to deduct marks from you for any request that requires TA's extra manual effort.

8. **Hardcoding won't work. We will use another similar set of test data and expected output when grading**. The good news is that, if you pass all the test cases during the assignment, you should also pass all the test cases during grading (unless you do hardcoding).

2.6 Late Submission Policy

We follow the late submission policy specified in the course outline.

3 Questions

If you have doubts about the assignment, you are encouraged to ask questions on our Facebook group.

4 Academic Honesty

We follow the University guide on academic honesty against any plagiarism.