Operating Systems Lab

Assessment 1 – Part A – Study Experiment

(Note: Practice all the commands given in this document.

You can practice in any one of the following environment:
1) Oracle VirtualBox,
2) Google Cloud Platform / AWS, **(recommended)**
3) repl.it, **(recommended)**
4) any other online editor
5) Desktop Unix version OS

You have to submit the report of completion in vtop

What Are Unix Files and Directories?

A file is a "container" for data. Unix makes no distinction among file types—a file may contain the text of a document, data for a program, or the program itself.

Directories provide a way to organize files, allowing you to group related files together. Directories may contain files and/or other directories. Directories are analogous to Macintosh and Windows folders.

Naming Unix Files and Directories

Each file and directory has a name. Within a directory, each item (that is, each file or directory) must have a unique name, but items with the same name may exist in more than one directory. A directory may have the same name as one of the items it contains.

File and directory names may be up to 256 characters long. Names may use almost any character except a space. You can divide a multi-word file name using either an underscore or a period (for example, **chapter_one** or **chapter.two**).

Some characters have special meanings to Unix. It is best to avoid using these characters in file names:

/ \ " ' * | ! ? ~ $ < >

Unix is case-sensitive. Each of these is a unique file: **myfile, Myfile, myFile,** and **MYFILE**.

Creating a File

Many people create files using a text editor, but you can use the command **cat** to create files without learning a text editor. To create a practice file (named **firstfile**) and enter one line of text in it, type the following at the **%** prompt:

cat > firstfile
(Press the **Enter/Return** key.)
This is just a test.
(Press the **Enter/Return** key.)

Stop file entry by typing **Control-d** on a line by itself. (Hold down the Control key and type d.) On your screen you will see:

% cat > firstfile
This is just a test.
^D

One way to examine the contents of the file you've just created is to enter this at the **%** prompt:

cat firstfile

Copying a File

To make a duplicate copy of a file, use the command **cp.** For example, to create an exact copy of the file called **firstfile,** you would type:

cp firstfile secondfile

The result is two files with different names, each containing the same information. The **cp** command works by overwriting information. If you create a different file called **thirdfile** and then type the following command:

cp thirdfile firstfile

you will find that the original contents of **firstfile** are gone, replaced by the contents of **thirdfile.**

Renaming a File

Unix does not have a command specifically for renaming files. Instead, the **mv** command is used both to change the name of a file and to move a file into a different directory.

To change the name of a file, use the following command format (where **thirdfile** and **file3** are sample file names):

mv thirdfile file3

The result of this command is that there is no longer a file called **thirdfile,** but a new file called
file3 contains what was previously in **thirdfile.**

Like **cp,** the **mv** command also overwrites existing files. For example, if you have two files,

fourthfile and **secondfile,** and you type the command

mv fourthfile secondfile

mv will remove the original contents of **secondfile** and replace them with the contents of **fourthfile.** The effect is that **fourthfile** is renamed **secondfile,** but in the process **secondfile** is deleted.

Removing a File

Use the **rm** command to remove a file. For example,

rm file3

deletes **file3** and its contents. You may remove more than one file at a time by giving a list of files to be deleted. For example,

rm firstfile secondfile

You will be prompted to confirm whether you really want to remove the files:

rm: remove firstfile (y/n)? y
rm: remove secondfile (y/n)?
n
Type **y** or **yes** to remove a file; type **n** or **no** to leave it.

Creating a Directory

Creating directories permits you to organize your files. The command

mkdir project1

creates a directory called **project1,** where you might store files related to a particular project. The directory that you create will be a subdirectory within your current directory.

Moving and Copying Files Into a Directory

The **mv** and **cp** commands can be used to put files into a directory. Assume that you want to put some files from your current directory into a newly created directory called **project1.** The command

mv bibliography project1

will move the file **bibliography** into the directory **project1.** The command

cp chapter1 project1

will put a copy of the file **chapter1** into the directory **project1,** but leave **chapter1** still in the current directory. There will now be two copies of **chapter1,** one in the current directory and one in **project1.**

Renaming a Directory

You can also use the **mv** command to rename and to move directories. When you type the command

mv project1 project2

the directory called **project1** will be given the new name **project2** as long as a directory called **project2** did not previously exist. If directory **project2** already existed before the mv command was issued, the result of

mv project1 project2

would be to put the directory **project1** and its files into the directory **project2.**

Copying a Directory

You can use the **cp** command to make a duplicate copy of a directory and its contents. To copy directory **project1** to directory **proj1copy,** for example, you would type

cp -r project1 proj1copy

If directory **proj1copy** already exists, this command will put a duplicate copy of **directory project1**
into directory **proj1copy.**

Removing a Directory

Use the command **rmdir** to remove an empty directory. Multiple empty directories may be removed by listing them after the command:

rmdir testdir1 testdir2

If you try to remove a directory that is not empty, you will see

rmdir: testdir3: Directory not empty

If you are sure that you want to remove the directory and all the files it contains, use the command

rm -r testdir3

Summary of Commands

Working With Files

- **mv file1 file2**
  Renames **file1** to **file2** (if **file2** existed previously, overwrites original contents of **file2**).
- **cp file1 file2**
  Copies **file1** as **file2** (if **file2** existed previously, overwrites original contents of **file2**).
- **rm file3 file4**
  Removes **file3** and **file4,** requesting confirmation for each removal.

Working With Directories

- **mkdir dir1**
  Creates a new directory called **dir1.**
- **mv dir1 dir2**
  If **dir2** does not exist, renames **dir1** to **dir2.**
  If **dir2** does exist, moves **dir1** inside **dir2.**
- **cp -r dir1 dir2**
  If **dir2** does not exist, copies **dir1** as **dir2.**
  If **dir2** does exist, copies **dir1** inside **dir2.**
- **rmdir dir1**
  Removes **dir1,** if **dir1** contains no files.
- **rm -r dir1**
  Removes **dir1** and any files it contains. Use with caution.

Working With Files and Directories

- **cp file1 dir1**
  Copies file **file1** into existing directory **dir1.**
- **mv file2 dir2**
  Moves file **file2** into existing directory **dir2.**

Listing a Directory's Contents

To see a list of the contents of a directory, type the **ls** command (short for list) at the % prompt.

```
% ls

  AppleVolumes  Private   Public   mail
```

In this example, there are four items in the current directory—AppleVolumes, Private, Public, and mail. We can't tell from this listing which items are files and which are directories.

Using Options on the ls Command

You can learn more about the directory's contents by setting options for the **ls** command. An option is indicated by a hyphen followed by a letter. Options are separated from the command by a space. For example, adding the **-F** option to the **ls** command will produce:

```
% ls -F

  AppleVolumes  Private/   Public/   mail/
```

Now you can tell that three of the items are directories, because they are marked with a */*. If you want to find out what files and directories are in the directory Private, you can specify which directory you want ls to list:

```
% ls -F Private

  bibliography  chapter1   chapter2
```

The **Private** directory contains three files. Another useful option is the **-l** (letter "ell," not one) option, which lists directory contents in a long format.

```
% ls -

l

total

7
-rw-------  1 bjensen   329 Jan 20 17:32 AppleVolumes
drwx------  9 bjensen  2048 May 27 16:28 Private
drwxrwxrwx  3 bjensen 2048 Jun 10 11:07 Public
drwx------  3 bjensen  2048 Jun 24 10:36 mail
```

The total entry shows how much space, measured in 512-byte (.5K) blocks, is used for these files. After that comes one line for each file.

The first character tells if the entry is a file (shown by **-**) or a directory (shown by **d**). Next comes a series of letters and hyphens that lists Unix permissions for each item. The next numeral gives the number of links to a file or directory.

The login name of the owner of these files appears next (bjensen). The next column gives the actual length of the file in bytes (1K=1024 bytes). Then come the date and

time when the file was last changed. The last item on each line is the file or directory name.

A few other options for the ls command are:

-a lists hidden files (which begin with a period or dot)
-d used with -l option to find the status of a directory, instead of listing the directory's contents
-s gives the size of each file in kilobytes (K)
-t lists files sorted by time of most recent modification
-r reverses the order of the sort to get reverse alphabetic (**ls -r**) or oldest first (**ls -rt**)
-R recursively lists contents of any subdirectories

The **-R** option for the **ls** command lists the contents of the current directory and the contents of any subdirectories, as shown below:

```
% ls -R

AppleVolumes    Private    Public    mail

Private:
bibliography    chapter1    chapter2    resume

Public:
00introduction

fyi_24.txt mail:
```

You can give multiple options to the **ls** command. For example, **ls -lt** shows a long listing with the newest (or most recently changed) files first.

Determining Which Directory You're Currently In

The command pwd (short for print working directory) will show where you are in the directory structure.

```
pwd
/afs/umich.edu/user/b/j/bjensen
```

Reading from right to left, this means: I am in the directory called **bjensen** (my home directory), which is contained in the directory **j**, which is contained in the directory **b**, which is contained in the directory **user**, contained in the cell **umich.edu**, contained in the root directory (**/afs**).

Moving From One Directory to Another

In order to move from my home directory (**bjensen**) to the directory called **Private,** use the command **cd** (short for change directory).

```
cd Private
```

Directory Abbreviations for Faster Navigating

Say you are the user **jansmart** and your current working directory is /afs/umich.edu/user/j/a/jansmart/Private/Paper1/Sources

You are in the directory **Sources**. If you wish to "back up" one level to the directory **Paper1** (the parent of directory **Sources**), you could type

cd Paper1

but a faster way is to use the abbreviation for parent directory (**..**). There is no space between the two periods:

cd ..

Another useful abbreviation is **~** (called tilde). This refers to a user's home directory (**jansmart** in our example). The easiest way to change back to your home directory from any directory is to type

cd ~

A third abbreviation (**.**) means the current working directory. For example, assume that you are in the directory **Sources**. You want to copy the file **intro** from **Paper1** into your current directory, **Sources**. Without using any abbreviations, you would need to type (all on one line):

cp /afs/umich.edu/user/j/a/jansmart/Private/Paper1/intro [space]
/afs/umich.edu/user/j/a/jansmart/Private/Paper1/Sources/intro

Using the abbreviation for our home directory (**~**) and the abbreviation for the current working directory (**.**) we would type only:

cp ~/Private/Paper1/intro .

What makes these abbreviations work are the "**.**" and "**..**" directories that appear when you type **ls**
-a.

Pathnames

Within the home directory **bjensen**, there are three
directories PublicPrivate mail

The directory **Private** contains several files:

bibliography chapter1 chapter2 resume

It is possible that more than one person could have a file called **bibliography**. How does Unix distinguish between files with the same name? The full name of each file includes the "path" through the directory hierarchy to that file.

The full names of two different **bibliography** files might be:

/afs/umich.edu/user/b/j/bjensen/Private/

bibliography and

/afs/umich.edu/user/j/a/jansmart/Private/Paper1/bibliography

The names of the two **bibliography** files shown above are absolute pathnames. An absolute pathname starts with a */* to represent the root directory, then traces the path through various subdirectories to the file.

Another way to describe a file is by its relative pathname. Relative pathnames do not begin with a
/. A relative pathname shows how to get to the file from the current working

directory. If you are in the directory **bjensen**, the relative pathname to your file bibliography is

Private/bibliography

Vi Editor

The vi editor is available on almost all Unix systems. You can use vi from any type of connection to Unix because it does not depend on arrow keys and function keys: it uses the standard alphabetic keys for commands.

vi is short for visual editor; it displays a window into the file being edited that shows 24 lines of text. vi is a text editor, not a "what you see is what you get" word processor. vi lets you add, change, and delete text but does not provide formatting capabilities such as centering lines or indenting paragraphs.

Starting vi

Create a new file by
typing vi *newname*
at the Unix % prompt, where *newname* is the name you wish to give the new file.

On the screen you will see blank lines, each with a tilde at the left, and a line at the bottom giving the name and status of the file:

~
~
"testvi" [New file]
Use vi to open an already existing file by
typing vi *filename*
where *filename* is the name of the existing file. If the file is not in your current directory, you must use the full pathname.

Entering Text

vi has two modes: command mode and insert mode. In command mode, the letters of the keyboard perform editing functions (like moving the cursor, deleting text, and so on). In insert mode, the letters you type form words and sentences. Unlike word processors, vi starts up in command mode.

To begin entering text in an empty file, you must change from command mode to insert mode. To do this, type the letter **i**. You may see INSERT MODE at the bottom right of the screen, or nothing may appear to change, but you are now in insert mode and can begin typing text. In general, you are not required to press the Return key to execute vi commands.

Type a few short lines and press Return at the end of each line. If you type a long line, you will notice that vi does not word wrap; it merely breaks the line unceremoniously at the edge of the screen. If you make a mistake, pressing the Backspace or Delete key may or may not remove the error, depending on the configuration of the software you are using to connect to the Login Service.

Moving the Cursor

To move the cursor to another position, you must be in command mode. If you have just finished typing text, you are still in insert mode. Go back to command mode by pressing the ESC key. (If INSERT MODE was displayed on your screen, it will disappear when you are in command mode.)

TIP: Depending on the software program you are using to connect to the Login Service or the options you have set, you may hear a beep or see the screen flash

when you switch to command mode.

You may find that the arrow keys on your keyboard will move the cursor up, down, left, and right. If they do not appear to work, the cursor can be controlled with the four keys h, j, k, and l, as follows:

Key	Cursor Movement
h	left one space
j	down one line
k	up one line
l	right one space

When you have gone as far as possible in one direction, the cursor stops moving and you may hear a beep or see the screen flash. For example, you can't use **l** to move right and wrap around to the next line; you must use **j** to move down a line.

Basic Editing

Editing commands require that you be in command mode. Many of the editing commands have a different function depending on whether they are typed in upper or lower case. Often, editing commands can be preceded by a number to indicate a repetition of the command.

NOTE: Unless specified otherwise, the instructions in this section assume that you are in the command mode.

Deleting characters. To delete a character from a file, move the cursor until it is on the incorrect letter, then type **x**. The character under the cursor disappears. To remove four characters (the one under the cursor and the next three) type **4x**. To delete the character before the cursor, type **X** (upper case).

Deleting words. To delete a word, move the cursor to the first letter of the word and type **dw**. The command deletes the word and the space following it. To delete three words, type **3dw**.

Deleting lines. To delete a whole line, type **dd**. The cursor does not have to be at the beginning of the line. **dd** deletes the entire line containing the cursor and places the cursor at the start of the next line. **2dd** deletes two lines.

To delete from the cursor position to the end of the line, type **D** (upper case).

Replacing characters. To replace one character with another, move the cursor to the character to be replaced. Type **r**, then the replacement character. The new character will appear, and you will still be in command mode.

Replacing words. To replace one word with another, move to the start of the incorrect word and type **cw**. The last letter of the word to be replaced will turn into a **$**. You are now in change mode and may type the replacement. The new text does not need to be the same length as the original. Press ESC to get back to command mode. **3cw** allows you to replace three words.

Replacing lines. To change text from the cursor position to the end of the line, type **C** (upper case). Type the replacement text and press ESC. Press ESC again to get back to command mode.

Inserting text. To insert text in a line, position the cursor where the new text should go and type **i**. Enter the new text. The text is inserted before the cursor. Press ESC to get back to command mode.

Appending text. To add text to the end of a line, position the cursor on the last letter of the line and type **a**. Enter the new text. This adds text after the cursor. Press ESC to get back to command mode.

Opening a blank line. Type **o** (lower case) to insert, or open, a blank line below the current line. Type **O** (upper case) to insert a blank line above the current line. Press ESC to get back to command mode.

Joining lines. The command **J** (upper case) lets you join two lines together. Put the cursor on the first line to be joined and type **J**. **3J** lets you join 3 lines together.

Undoing. Undo your most recent edit by typing **u** (lower case). **U** (upper case) undoes all the edits on a single line, as long as the cursor stays on that line. Once you move off a line, you can't use **U** to restore it.

Moving Around In a File

These shortcuts, which work in command mode, allow you to move more quickly through a file.

Key(s)	Movement
w	forward word by word
b	backward word by word
$	to end of line
0 (zero)	to beginning of line
H	to top line of screen
M	to middle line of screen
L	to last line of screen
G	to last line of file
1G	to first line of file
ctrl-f	jump forward one screen
ctrl-b	jump backward one screen
ctrl-d	scroll down one-half screen
ctrl-u	scroll up one-half screen

Moving by searching. To move quickly in a file, search for text. In command mode, type a */* (slash) followed by the text to search for. Press Return. The cursor moves to the first occurrence of that text. Repeat the search in a forward direction by typing **n** (lower case), or in a backward direction by typing **N** (upper case).

Closing and Saving a File

With vi, you edit a copy of the file rather than the original file. The original is changed only when you save your edits.

To save the file and quit vi, type **ZZ** (upper case) in command mode.

The vi editor is built on an earlier Unix text editor called ex. ex commands can be used within vi. ex commands begin with a **:** (colon) and end with a Return. The command is displayed on the status line as you type. Some ex commands are useful when saving and closing files.

To save (write) the edits you have made, but leave vi running and your file open, press ESC, then type **:w** and press Return.

To quit vi, and discard any changes you have made since last saving, press ESC, then

type **:q!**
and press Return.

Displaying Line Numbers

To display the line numbers of the file being edited, type **:set number** and press Return. To cause vi to always display line numbers, add the following line to a file called .exrc in your home directory:

```
set number
```

Searching Files Using UNIX grep

The **grep** program is a standard UNIX utility that searches through a set of files for an arbitrary text pattern, specified through a regular expression. By default, grep is case-sensitive (use -i to ignore case). By default, grep ignores the context of a string (use -w to match *words* only). By default, grep shows the lines that match (use -v to show those that **don't** match).

% grep BOB tmpfile	{search 'tmpfile' for 'BOB' anywhere in a line}
% grep -i -w blkptr *	{search files in CWD for word *blkptr*, any case}
% grep run[-]time *.txt	{find 'run time' or 'run-time' in all txt files}
% who | grep root	{pipe **who** to **grep**, look for *root*}

Understanding Regular Expressions

Regular Expressions are a feature of UNIX. They describe a pattern to match, a sequence of characters, not words, within a line of text. Here is a quick summary of the special characters used in the grep tool and their meaning:

^ (Caret)	=	match expression at the start of a line, as in ^A.
$ (Question)	=	match expression at the end of a line, as in A$.
\ (Back Slash)	=	turn off the special meaning of the next character, as in \^.
[ ] (Brackets)	=	match any **one** of the enclosed characters, as in [aeiou]. Use Hyphen "-" for a range, as in [0-9].
[^]	=	match any one character **except** those enclosed in [ ], as in [^0-9].
. (Period)	=	match a single character of any value, except end of line.
* (Asterisk)	=	match zero or more of the preceding character or expression.
\{x,y\}	=	match **x** to **y** occurrences of the preceding.
\{x\}	=	match exactly **x** occurrences of the preceding.
\{x,\}	=	match **x** or more occurrences of the preceding.

Since you usually type regular expressions within shell commands, it is good practice to enclose the regular expression in single quotes (') to stop the shell from expanding it before passing the argument to your search tool. Here are some examples using **grep**:

grep smug files	{search *files* for lines with 'smug'}
grep '^smug' files	{'smug' at the start of a line}
grep 'smug$' files	{'smug' at the end of a line}
grep '^smug$' files	{lines containing only 'smug'}
grep '\^s' files	{lines starting with '^s', "\" escapes the ^}
grep '[Ss]mug' files	{search for 'Smug' or 'smug'}
grep 'B[oO][bB]'	{search for BOB, Bob, BOb or BoB }

files	
grep '^$' files	{search for blank lines}
grep '[0-9][0-9]' file	{search for pairs of numeric digits}

Back Slash "\" is used to *escape* the next symbol, for example, turn off the special meaning that it has. To look for a Caret "^" at the start of a line, the expression is ^\^. **Period** "." matches any single character. So b.b will match "bob", "bib", "b-b", etc. **Asterisk** "*" does not mean the same thing in regular expressions as in wildcarding; it is a modifier that applies to the preceding single character, or expression such as [0-9]. An asterisk matches **zero** or more of what precedes it.
Thus [A-Z]* matches any number of upper-case letters, including none, while [A-Z][A-Z]* matches
one or more upper-case letters.

The **vi** editor uses \< \> to match characters at the beginning and/or end of a **word boundary**. A word boundary is either the edge of the line or any character except a letter, digit or underscore "_". To look for if, but skip stiff, the expression is \<if\>. For the same logic in **grep**, invoke it with the **-w** option. And remember that regular expressions are **case-sensitive**. If you don't care about the case, the expression to match "if" would be [Ii][Ff], where the characters in **square brackets** define a character set from which the pattern must match one character. Alternatively, you could also invoke **grep** with the **-i** option to ignore case.

Here are a few more examples of **grep** to show you what can be done:

grep '^From: ' /usr/mail/$USER	{list your mail}
grep '[a-zA-Z]'	{any line with at least one letter}
grep '[^a-zA-Z0-9]	{anything not a letter or number}
grep '[0-9]\{3\}-[0-9]\{4\}'	{999-9999, like phone numbers}
grep '^.$'	{lines with exactly one character}
grep '"smug"'	{'smug' within double quotes}
grep '"*smug"*'	{'smug', with or without quotes}
grep '^\.'	{any line that starts with a Period "."}
grep '^\.[a-z][a-z]'	{line start with "." and 2 lc letters}