Introduction to Linux I/O Redirection, pipes and few useful tools

In your programming course, you should have learnt about I/O streams, specifically *standard output* and *standard input*. For instance, when you use *printf* function in C, it prints to the standard output. When you use *scanf* in C it expects the input from the keyboard. There is another output stream called *standard error;* which we are not going to use today. The three standard streams can be summarized as follows.

Stream	Description	Default
standard output	Standard output is the stream where a program writes its output data	by default, outputs on the terminal
standard input	Standard input is stream data (often text) going into a program	by default, input from the keyboard
standard error	Standard error is another output stream typically used by programs to output error messages or diagnostics	by default, outputs on the terminal

Stream Redirection

Linux allows you to redirect these standard streams. The redirection capabilities provide you with a robust set of tools to make all sorts of tasks easier to accomplish. Whether you're writing a program or performing file management through the command line, knowing how to manipulate the different I/O streams in your environment will greatly increase your productivity.

Redirecting standard output to a file

The following commands write the standard output of a program to a file.

```
$program > filename.txt
```

```
$program >> filename.txt
```

If a non-existent file is targeted (either by a single-bracket > or double-bracket command >>), a new file with that name will be created prior to writing. Commands with a single bracket overwrite the destination's existing contents. Commands with a double bracket do not overwrite the destination's existing contents.

Let's see an example:

the `echo` command in Linux is used to display a line of text. If you issue `echo "Hi!"` the message "Hi" will be written to the standard output. Now redirect the standard output to a file by and observe the content in the text file

```
$echo "Hi!" > write_to_me.txt
```

Now run the following and observe.

```
$echo “Hi for the second time!” >> write_to_me.txt
```

Redirecting standard input from a file

The following command makes the program take its standard input from a file.

```
$program < filename.txt
```

Let's see an example:

Run the command `cat` on the terminal and observe that it displays what you provide to the standard input on the standard output. Now redirect the input from a file as follows and observe.

```
$cat < write_to_me.txt
```

Pipes

Pipes are used for redirecting a stream from one program to another. When a program's standard output is sent to another through a pipe, the first program's output data will be received by the second program, as its input.

The Linux pipe is represented by a vertical bar.

```
$[Command 01] | [Command 02]
```

An example of a command using a pipe:

```
e15XXX@aiken:~$ ls -l /etc | less
total 1516
drwxr-xr-x  3 root root      4096 Jul  8  2015
acpi
-rw-r--r--  1 root root      2981 Jul  9  2015
adduser.conf
-rw-r--r--  1 root root      2981 Jul  8  2015
adduser.conf~
drwxr-xr-x  2 root root     12288 Aug 17 10:47
alternatives
-rw-r--r--  1 root root       395 Jun 20  2010
anacrontab
:
```

This takes the output of `ls`, which displays the contents of /etc, and pipes it to the `less` program. `less` displays the data sent to it one screen at a time.

Note that you can also do the same thing using I/O redirection using following two commands

```
ls -l /etc > tempout.txt
less < tempout.txt
```

But a pipe is easy to write and efficient (you have to write only 1 command and the time to read/write files to the disk are saved).

Few useful tools

grep

The *grep* command is used to search in the text. It searches the given file for lines containing a match to the given strings or words. It is one of the most useful commands on Linux. You can use *grep* in one of the following ways.

```
$grep 'keyword' filename.txt
$grep 'keyword' file1.txt file2.html file3.c
$grep --color 'keyword' filename.txt

$cat findFromMe.txt | grep 'keyword'
```

Try following examples and see results

```
grep 'Cache' /proc/meminfo
grep --color 'Cache' /proc/meminfo
ls /etc | grep '.conf'
```

To see more options about *grep,* follow this article (and don't forget to ***man grep***)

https://www.computerhope.com/unix/ugrep.htm

Note: You can also use a wild card character like '*' with grep command. A wildcard character is a keyboard character such as an **asterisk** (*) or a **question mark** (?) that is used to represent one or more characters when you are searching for files, folders, etc. (grep Ca* /proc/cpuinfo would yield a similar result as grep C /proc/cpuinfo).
A Useful trick: *Imagine you are working on Windows and you want to find only the mp3 files within the folder. In Windows Explorer's search bar (or press F3) type *.mp3 (*.File_extension will do the trick)*

find

Find command is needed on Linux to search through directories for files. This command proceeds to search through all accessible subdirectories from given directory. The filename is usually specified by the `-name` option.

```
$find [path] -name [filename or key word]
```

Use `find` from the command line to locate a specific file by name or extension. The above example searches for *[filename or key word]* files in the *[path]* directory and all sub-directories. Following are some example usage of `find`.

`find . -name testfile.txt` : Find a file called testfile.txt in current and sub-directories.

`find /home -name *.jpg` : Find all .jpg files in the /home and sub-directories.

`find . -type f –empty` : Find an empty file within the current directory.

See man page to see that there are many ways to use `find`.

Example:

```
e15XXX@aiken:~$ find /usr/include -name *.h
```

diff

The `diff` command compares text files. It can compare single files or the contents of directories.

Note: The `diff` command only works with input files that are text files.

If the Directory1 and Directory2 parameters are specified, the `diff` command compares the text files that have the same name in both directories. Binary files that differ, common subdirectories, and files that appear in only one directory are listed.

When the `diff` command is run on regular files, and when comparing text files that differ during directory comparison, the `diff` command tells what lines must be changed in the files to make them agree. If neither the File1 nor File2 parameter is a directory, then either may be given as – (minus sign), in which case the standard input is used. If the File1 parameter is a directory, then a file in that directory whose file name is the same as the File2 parameter is used.

Lines from the first file are preceded by a less than symbol (<) and lines from the second file by a greater than symbol (>). A dashed line (--) is used to separate output from the two files. The letters can be used to convert file1 into file2:

c	Replace lines from file1 with those from file2.
d	Delete lines from file1..
a	Add lines from file2 to file1.

Examples:

```
e15XXX @aiken:~$ cat > A.txt
abc
def
ghi
ctrl+d
e15XXX @aiken:~$ cat > B.txt
abc
ghi
klm
ctrl+d
e15XXX @aiken:~$ diff A.txt B.txt
2d1
< def
3a3
> klm
```

It means one needs to delete the 2nd line in the first file and add the 3rd line from the second file at 3rd position in the first file to make them same.

awk

The awk command is a powerful method for processing or analyzing text files—in particular, data files that are organized by lines (rows) and columns.
Simple awk commands can be run from the command line. More complex tasks should be written as awk programs (so-called awk scripts) to a file.

The basic format of an awk command looks like this:

```
awk 'pattern {action}' input-file
```

take each line of the input file; if the line contains the pattern apply the action to the line and write the resulting line to the output file.

Let's see an example that contains only an action:
Given below is the content of a file called "table1.txt". You can find this file somewhere in */home/e00000*. Use `find` tool to find it.

```
1, Justin Timberlake, Title 545, Price $7.30
2, Taylor Swift, Title 723, Price $7.90
3, Mick Jagger, Title 610, Price $7.90
4, Lady Gaga, Title 118, Price $7.30
5, Johnny Cash, Title 482, Price $6.50
6, Elvis Presley, Title 335, Price $7.30
7, John Lennon, Title 271, Price $7.90
8, Michael Jackson, Title 373, Price $5.50
```

Run the following command.

```
awk '{ print $5 }' table1.txt
```

This statement takes the element of the 5th column of each line and writes it as a line in the output. The variable '$5' refers to the fifth column. Similarly, you can access the first, second, and third column, with $1, $2, $3, etc. By default, columns are assumed to be separated by spaces or tabs (so-called white space). So, the command would write the following lines as the output.

```
545,
723,
610,
118,
482,
335,
271,
373,
```

Now run the following command.

```
awk -F, '{ print $3 }' table1.txt
```

-F, is to specify that the separator is the comma. This will output the following.

```
Title 545
Title 723
Title 610
Title 118
...
```

Now consider the following example that has a pattern and an action.

```
awk '/30/ { print $3 }' table1.txt
```

The string between the two slashes ('/') is the regular expression. In this case, it is just the string "30." This means if a line contains the string "30", the system prints out the element at the 3rd column of that line. The output in the above example would be:

```
Timberlake,
Gaga,
Presley,
```

To see more options about *awk,* follow this article:

https://linuxconfig.org/learning-linux-commands-awk

There are many other such useful tools. However, it is impossible to learn all of them in few days. The best way to learn them is to keep using Linux. When the need for a certain task comes, you can get the help from the system itself (as you learnt a few weeks back) or use the Internet. Here are few other tools that might be useful for you.

- head
- tail
- sort
- uniq
- tr

Assignment

This assignment should be done on aiken.ce.pdn.ac.lk logged in through SSH. You should record what you do using `ttyrec` and submit the `ttyrec` recording. If you have any doubts you should ask them in the discussion forum.

1. Create *output.txt* file with last 10 logins into the system (use output redirection).

2. Append system date at the end of that file (not allowed to use a text editor, use output redirection).

3. Replace that text file with e15 computer students' e-numbers. (Tip: Each Computer engineering student has directory named e-number in /home, filter the list and use output redirection)

4. Use `find` command to find where the file called 'libz.so' resides in */usr* on Aiken.

5. Use `find` command to find all the files in */home/e00000* that ends with '.txt'

6. Print lines which contain 'nobody' from following text file */home/e00000/week4/pg730.txt* .

7. Extract the first and the last column of the output of `ls -l /home`

8. Sort the word list in */home/e00000/week4/words.txt* in alphabetical order and write to a file named *sortedwords.txt*

References:

https://www.digitalocean.com/community/tutorials/an-introduction-to-linux-i-o-redirection

https://www.computerhope.com/unix/ugrep.htm