

OS Lab Report – Week 1

PES1201800366

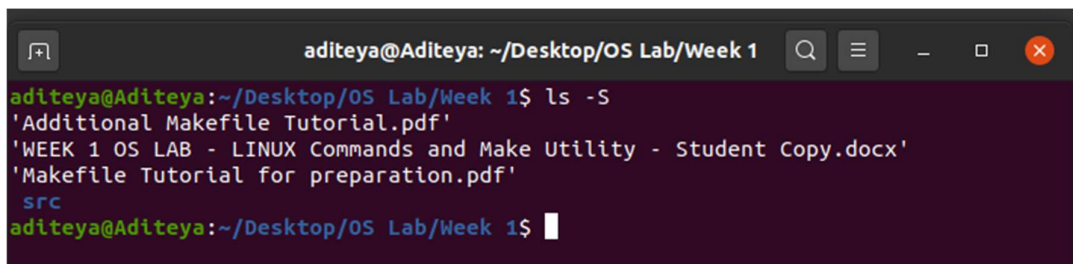
Aditeya Baral

1. Exercise 1 – Linux Commands

- Execute and familiarize with Linux environment and commands. Getting used to basic commands on Linux Operating System – Process creation, Process monitoring, Process states, Linux File system tree, Linux File system commands.
- Basic LINUX commands (ANY 5) executed in the lab should be submitted in the following way:
 - Command: What does the command do?
 - Any two options (i.e. flags or arguments) regarding the command
 - Outcome of the command

1.1 ls

- The **ls** command is a command-line utility for listing the contents of a directory or directories given to it via standard input and writes results to standard output.
- Flags can be provided to the command:
 - **-S**: this flag is used to sort the listed files and directories in decreasing order of size



```
aditeya@Aditeya: ~/Desktop/OS Lab/Week 1
aditeya@Aditeya:~/Desktop/OS Lab/Week 1$ ls -S
'Additional Makefile Tutorial.pdf'
'WEEK 1 OS LAB - LINUX Commands and Make Utility - Student Copy.docx'
'Makefile Tutorial for preparation.pdf'
src
aditeya@Aditeya:~/Desktop/OS Lab/Week 1$
```

- **-R**: this flag is used to display a recursive listing of all files and directories by traversing the entire path

```
aditeya@Aditeya: ~/Desktop/OS Lab/Week 1
aditeya@Aditeya:~/Desktop/OS Lab/Week 1$ ls -R
.:
'Additional Makefile Tutorial.pdf'
'Makefile Tutorial for preparation.pdf'
src
'WEEK 1 OS LAB - LINUX Commands and Make Utility - Student Copy.docx'

./src:
Client.c Client.o Header.h make.mk reverse.out Server.c Server.o
aditeya@Aditeya:~/Desktop/OS Lab/Week 1$
```

1.2 mkdir

- The **mkdir** command is a command line utility used to create a directory or even a directory structure given to it via standard input.
- Flags/Arguments can be provided to the command:
 - **dirname**: this argument specifies the name of the directory to be created

```
aditeya@Aditeya: ~/Desktop
aditeya@Aditeya:~/Desktop$ mkdir testdir -v
mkdir: created directory 'testdir'
aditeya@Aditeya:~/Desktop$
```

- **-p**: this flag is used to create an entire directory path (with parent and child directories) if the parent directory does not exist

```
aditeya@Aditeya: ~/Desktop
aditeya@Aditeya:~/Desktop$ mkdir -p parentdir/childdir -v
mkdir: created directory 'parentdir'
mkdir: created directory 'parentdir/childdir'
aditeya@Aditeya:~/Desktop$
```

1.3 rm

- The **rm** command is a command line utility used to delete files and folders given to it via standard input
- Flags/Arguments can be provided to the command:
 - **-r**: This flag is used to recursively delete a directory and all its contents since rm does not delete directories by default

```
aditeya@Aditeya: ~/Desktop
aditeya@Aditeya:~/Desktop$ rm -r -v parent/
removed directory 'parent/child'
removed directory 'parent/'
aditeya@Aditeya:~/Desktop$
```

- **-i**: This flag confirms with and asks the user before deleting each file

```
aditeya@Aditeya: ~/Desktop
aditeya@Aditeya:~/Desktop$ rm -i test.py
rm: remove regular file 'test.py'? y
aditeya@Aditeya:~/Desktop$
```

1.4 top

- The **top** command is used to show all the running processes on the system. It is like **ps** but provides an additional functionality of updating the processes at regular intervals.
- Flags can be provided to the command:
 - **-u**: This flag is used to show all the processes associated with a user.

```
aditeya@Aditeya: ~/Desktop
top - 13:23:35 up 3:41, 1 user, load average: 0.16, 0.03, 0.01
Tasks: 190 total, 1 running, 189 sleeping, 0 stopped, 0 zombie
%Cpu(s): 10.8 us, 1.4 sy, 0.0 ni, 87.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7962.4 total, 5593.1 free, 1008.7 used, 1360.6 buff/cache
MiB Swap: 1401.6 total, 1401.6 free, 0.0 used, 6678.6 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR  S  %CPU  %MEM   TIME+ COMMAND
 1809 aditeya   20   0 3704980 382996 126896 S   6.6   4.7   0:55.15 gnome-s+
 1520 aditeya   20   0 265844  93468  48352 S   3.6   1.1   0:12.92 Xorg
 6095 aditeya   20   0 969016  50304  39388 S   1.7   0.6   0:01.68 gnome-t+
 4599 aditeya   20   0 996432  72388  47032 S   0.7   0.9   0:04.40 nautilus
 1734 aditeya   20   0   7224   4272   3820 S   0.3   0.1   0:00.05 dbus-da+
 6485 aditeya   20   0  20476   3816   3248 R   0.3   0.0   0:00.02 top
 1462 aditeya   20   0   30088 20232   8204 S   0.0   0.2   0:01.17 systemd
 1467 aditeya   20   0  103460  3544     4 S   0.0   0.0   0:00.00 (sd-pam)
 1473 aditeya    9 -11 1941204 20320  16392 S   0.0   0.2   0:00.26 pulseau+
 1475 aditeya   39  19 519980 24956  16748 S   0.0   0.3   0:00.45 tracker+
 1478 aditeya   20   0  248816  7692   6720 S   0.0   0.1   0:00.51 gnome-k+
 1482 aditeya   20   0   11292  8376   3840 S   0.0   0.1   0:01.81 dbus-da+
 1498 aditeya   20   0  248328  7688   6696 S   0.0   0.1   0:00.05 gvfsd
 1503 aditeya   20   0  378336  6404   5752 S   0.0   0.1   0:00.01 gvfsd-f+
 1504 aditeya   20   0  396272  9236   7540 S   0.0   0.1   0:00.13 gvfs-ud+
 1517 aditeya   20   0  172636  6388   5740 S   0.0   0.1   0:00.00 gdm-x-s+
 1518 aditeya   20   0  245120  6736   6016 S   0.0   0.1   0:00.01 gvfs-go+
```

top -u aditeya

- **-n**: This flag is used to exit the top command after a certain number of update intervals

```
aditeya@Aditeya: ~/Desktop
Tasks: 190 total, 1 running, 189 sleeping, 0 stopped, 0 zombie
%Cpu(s): 5.9 us, 11.8 sy, 0.0 ni, 82.4 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7962.4 total, 5593.4 free, 1008.4 used, 1360.6 buff/cache
MiB Swap: 1401.6 total, 1401.6 free, 0.0 used, 6678.8 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM    TIME+  COMMAND
 1708 aditeya   20   0  319828   7560  5980  S   6.2   0.1   0:03.39 ibus-da+
    1 root      20   0  102196  11620  8408  S   0.0   0.1   0:02.20 systemd
    2 root      20   0        0        0        0  S   0.0   0.0   0:00.01 kthreadd
    3 root       0 -20        0        0        0  I   0.0   0.0   0:00.00 rcu_gp
    4 root       0 -20        0        0        0  I   0.0   0.0   0:00.00 rcu_par+
    6 root       0 -20        0        0        0  I   0.0   0.0   0:00.00 kworker+
    9 root       0 -20        0        0        0  I   0.0   0.0   0:00.00 mm_perc+
   10 root      20   0        0        0        0  S   0.0   0.0   0:00.25 ksoftir+
   11 root      20   0        0        0        0  I   0.0   0.0   0:01.29 rcu_sch+
   12 root      rt    0        0        0        0  S   0.0   0.0   0:00.10 migrati+
   13 root     -51   0        0        0        0  S   0.0   0.0   0:00.00 idle_in+
   14 root      20   0        0        0        0  S   0.0   0.0   0:00.00 cpuhp/0
   15 root      20   0        0        0        0  S   0.0   0.0   0:00.00 kdevtmp+
   16 root       0 -20        0        0        0  I   0.0   0.0   0:00.00 netns
   17 root      20   0        0        0        0  S   0.0   0.0   0:00.00 rcu_tas+
   18 root      20   0        0        0        0  S   0.0   0.0   0:00.03 kauditd
   19 root      20   0        0        0        0  S   0.0   0.0   0:00.00 khungta+
aditeya@Aditeya:~/Desktop$
```

`top -n 1`

1.5 date

- The `date` command is a command line utility that is used to manipulate datetimes on Linux. It can be used to set dates, and perform operations using them.
- Flags/Arguments can be provided to the command:
 - **-d**: This flag is used to operate on a single date, which can also be provided in human readable format

```
aditeya@Aditeya: ~/Desktop
aditeya@Aditeya:~/Desktop$ date -d "tomorrow"
Sunday 30 August 2020 01:35:05 PM IST
aditeya@Aditeya:~/Desktop$ date -d "last week"
Saturday 22 August 2020 01:35:17 PM IST
aditeya@Aditeya:~/Desktop$ date -d "2000-01-04"
Tuesday 04 January 2000 12:00:00 AM IST
aditeya@Aditeya:~/Desktop$
```

- **+format**: This argument is a format string and is used to return results in the date format specified

```
aditeya@Aditeya: ~/Desktop
aditeya@Aditeya:~/Desktop$ date +"%Y-%m-%d"
2020-08-29
aditeya@Aditeya:~/Desktop$ date +"Week number: %V Year: %y"
Week number: 35 Year: 20
aditeya@Aditeya:~/Desktop$ date +%s
1598688667
aditeya@Aditeya:~/Desktop$
```

2. Exercise 2 – Makefile and Reversal of Array

- Write a C program to display an array in reverse using index. Create Makefile (ex: make.mk below) and other files as shown below:
 - Client.c – contains main function to collect input on array elements from the user and calls reverse_array function
 - Server.c – contains reverse_array function and prints the reversed array (use a separate function to print the reversed array)
 - Header.h – contains function prototypes
 - make.mk – contains targets and their dependencies
- Main program and all sub programs (dependency files, header file and Makefile) should be submitted. Steps to execute make and output of the program should be submitted.

2.1 Code

2.1.1 Client.c

```
1. #include "Header.h"
2. int main()
3. {
4.     int n;
5.     printf("Enter size of Array: ");
6.     scanf("%d", &n);
7.
8.     int *a = (int*)calloc(n, sizeof(int));
9.
10.    printf("Enter elements: ");
11.    for(int i = 0; i<n; i++)
12.        scanf("%d", &a[i]);
13.
14.    printf("\nInput Array:\n");
15.    display_array(a, n);
16.
17.    printf("\nReversed Array:\n");
18.    reverse_array(a, n);
19.
20.    free(a);
21.    return 0;
22. }
```

2.1.2 Server.c

```
1. #include "Header.h"
2. void reverse_array(int *a, int n)
3. {
4.     int temp, start = 0, end = n - 1;
5.     while (start < end)
6.     {
```

```

7.     temp = a[start];
8.     a[start] = a[end];
9.     a[end] = temp;
10.    start++;
11.    end--;
12.    }
13.    display_array(a, n);
14. }
15.
16. void display_array(int *a, int n)
17. {
18.     for (int i = 0; i < n; i++)
19.         printf("%d ", a[i]);
20.     printf("\n");
21. }

```

2.1.3 Make.mk

```

1. reverse.out: Server.o Client.o
2.  gcc Server.o Client.o -o reverse.out
3. Server.o: Server.c Header.h
4.  gcc -c Server.c
5. Client.o: Client.c Header.h
6.  gcc -c Client.c
7. clean:
8.  rm -rf *.o
9.  rm -rf *.out

```

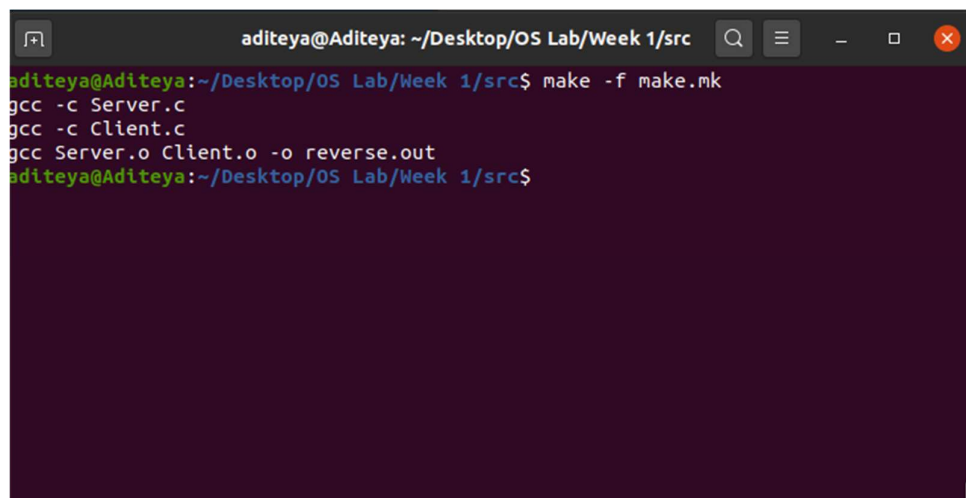
2.1.4 Header.h

```

1. #include <stdio.h>
2. #include <stdlib.h>
3. void display_array(int *, int);
4. void reverse_array(int *, int);

```

2.2 Screenshots



```

aditeya@Aditeya: ~/Desktop/OS Lab/Week 1/src
aditeya@Aditeya:~/Desktop/OS Lab/Week 1/src$ make -f make.mk
gcc -c Server.c
gcc -c Client.c
gcc Server.o Client.o -o reverse.out
aditeya@Aditeya:~/Desktop/OS Lab/Week 1/src$

```



```
aditeya@Aditeya: ~/Desktop/OS Lab/Week 1/src
aditeya@Aditeya:~/Desktop/OS Lab/Week 1/src$ ./reverse.out
Enter size of Array: 5
Enter elements: 1 2 3 4 5

Input Array:
1 2 3 4 5

Reversed Array:
5 4 3 2 1
aditeya@Aditeya:~/Desktop/OS Lab/Week 1/src$ ./reverse.out
Enter size of Array: 6
Enter elements: 6 5 4 3 2 1

Input Array:
6 5 4 3 2 1

Reversed Array:
1 2 3 4 5 6
aditeya@Aditeya:~/Desktop/OS Lab/Week 1/src$
```

3. Exercise 3 – Questions

Answer the following questions (Brief answers only).

1. Why do we use Makefile?

- A makefile is used to automate the software building procedure and other complex tasks which involve a lot of dependencies and requirements by issuing a set of rules or commands required which state when and how to compile a given source program. The makefile is read by the make utility, which understands the project structure and instructions and then performs and executes the commands stated in the makefile.

2. Is Makefile a shell script?

- No, the makefile is not a shell script. The makefile is read by the make command, and not the shell program itself. Make however uses the shell to execute the commands listed for each dependency but is distinguished by its ability to use the modification information such as the edit time to select which dependencies to recompile.

3. What does “clean” do in Makefile?

- “clean” is a phony target which is sometimes included in a makefile. A phony target can have any name and does not correspond to any target file. It is usually used to execute a list of commands before the execution of the target dependencies. “clean” is a conventional phony target name used to remove the outdated object files and executables

before the source code is recompiled.

4. How does make learn about the last modified files to be complied?

- make learns about modified files and other dependencies by inspecting the system's modification meta information. If a certain target file's modification time is older than any of its dependencies, it realises that the file has to be rebuilt or recompiled and follows through by executing the set of commands required to build the target file.

5. What does Cflags in Makefile mean?

- Cflags is a macro defined in a makefile to specify the options that must be passed to the compiler during compilation of dependency files to create the target file. Macros are defined to create shortcuts to code in a makefile and serve to help programmers avoid repeating long text entries. Cflags is a conventional macro name used that is expanded into the full option string when make encounters it while executing the required commands.

6. Why do we use -f option with make command?

- The make command reads or takes in a makefile and executes the required commands for each target file. By default, it looks for a makefile named "makefile.mk" but can also take a filename as an argument. This filename is provided to the make command utility via the -f option. Syntax: make -f <filename>.mk