

Operating System

Lab One Simple Shell

```
/ root
├── /usr # shareable and read-only # data
│   ├── /local # local software
│   │   ├── /bin
│   │   ├── /games
│   │   ├── /include
│   │   ├── /lib
│   │   ├── /man
│   │   ├── /sbin
│   │   ├── /share
│   │   ├── /src
│   │   └── /share
│   ├── /share # static data sharable # among all architectures
│   ├── /man # manual pages
│   ├── /man1 # user programs
│   ├── /man2 # system calls
│   ├── /man3 # lib functions
│   ├── /man4 # special file
│   ├── /man5 # file formats
│   ├── /man6 # games
│   ├── /man7 # misc.
│   └── /man8 # system admin.
├── /bin # most user commands
│   ├── /include
│   ├── /lib
│   ├── /obj. bin, lib files
│   └── /sbin # non essential binaries
├── /var # variable data files
│   ├── /cache # application cache data
│   ├── /lib # variable state data # remains after reboot
│   ├── /lock # lock files for shared # resources
│   ├── /opt # variable data of # packages installed
│   ├── /run # info of system since it # was booted
│   ├── /tmp # available for prog.
│   ├── /spool # data awaiting processing
│   ├── /lpd
│   ├── /mqueue
│   ├── /news
│   ├── /rwho
│   └── /uucp
├── /sbin # system binaries
│   ├── fastboot
│   ├── fastboot
│   ├── fsck
│   ├── halt
│   ├── ifconfig
│   ├── init
│   ├── mkfs
│   ├── mksmap
│   ├── reboot
│   ├── route
│   ├── swapoff
│   └── update
├── /tmp # temporary files deleted on # bootup
├── /dev # location of special or # device files # [contains makedev]
├── /home # user home directories
├── /lib # library and kernel modules
├── /mnt # mount files for temporary # filesystems
├── /opt # add-on application # filesystems
└── /root # home dir. for root user
```

Presented by:

Mostafa Hakam

5525

CONTENTS

Summary2

Code3

Sample Runs4

Implementing a simple UNIX shell that supports:

1. Commands with no arguments.
2. Commands with arguments.
3. Commands with (no) arguments executed in the background.
4. Exit command.

Language Used:

- C/C++ Language.

Libraries used:

1. Library "unistd".
2. Library "stdio".
3. Library "string".
4. Library "srdlib".
5. Library "signal".
6. Library "sys/wait".

Prcedures:

1. Scanning command from user.
2. Tokenising command (and arguments) into strings.
3. Dynamically allocating array of command (and arguments).
4. Using fork(), execev(), wait(), and exit() functions to create processes.

```

#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/wait.h>

// Handles the signals for commands executed in the
background
void handles() {
    int x, y;
    y = waitpid(-1, &x, WNOHANG);
    if (y > 0) {
        FILE* f = fopen("history.log", "a+");

        if(f == NULL)
            return;

        fprintf(f, "Process %d: terminated successfully in
the background.\n", y);
        fclose(f);
        fgetc(f);
    }
}

int main() {

    // Erase all data in the log
    FILE* f = fopen("history.log", "w");
    fclose(f);

    char** commandArray;

    char userInput[100];
    char* temp, * delimiter = " \n";

    char token[50];

    int commandSize;
    int i, flag, frk, proc;

    // Create signal struct
    struct sigaction sig;
    sig.sa_handler = handles;
    sig.sa_flags = SA_RESTART;

    // Terminal loop until EXIT is called
    while(1) {
        flag = 1;

        printf("> $ ");
        fgets(userInput, 100, stdin);

        // If no command was entered
        if(!strcmp(userInput, "\n"))
            flag = 0;
        else
            strcpy(userInput, strtok(userInput, "\n"));

```

```

// If command was entered
if(flag) {
    strcpy(token, userInput);
    temp = strtok(token, delimiter);

    i = 0;
    commandSize = 0;

    // Count command size
    while (temp != NULL) {
        commandSize += 1;
        temp = strtok(NULL, delimiter);
    }

    commandArray = malloc(sizeof(char *) * commandSize);
    strcpy(token, userInput);
    temp = strtok(token, delimiter);

    // Dynamically allocate command into array of
pointers
    while (temp != NULL) {
        commandArray[i] = (char *)
malloc(sizeof(char) * strlen(temp));
        strcpy(commandArray[i], temp);
        temp = strtok(NULL, delimiter);
        i++;
    }

    i = 0;

    // If EXIT command was called
    if (!strcmp(commandArray[i], "exit")) {
        printf("Proceeding EXIT ...\n");
        exit(0);
    } else if (commandArray[0] == NULL) {
        continue;
    }

    // If command is set to run in background
    if(!strcmp(commandArray[commandSize - 1], "&")) {
        flag = 0;
        commandArray[commandSize - 1] = NULL;
    }

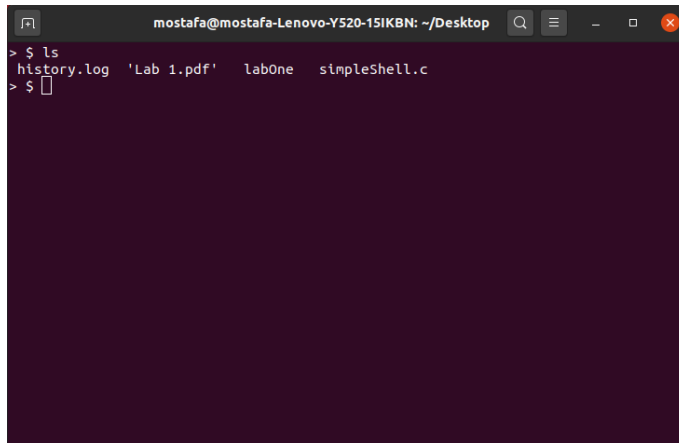
    // Creating processes from command
    frk = fork();
    if (frk >= 0) {
        if (!frk) {
            proc = execvp(commandArray[0],
commandArray);

            if (proc == -1)
                printf("Error: %s: command not
found\n", userInput);
            exit(0);
        } else if(flag) {
            wait(NULL);
        } else
            sigaction(SIGCHLD, &sig, NULL);
    } else
        printf("Forking Error");
    usleep(100000);
}
}
}

```

Sample Runs

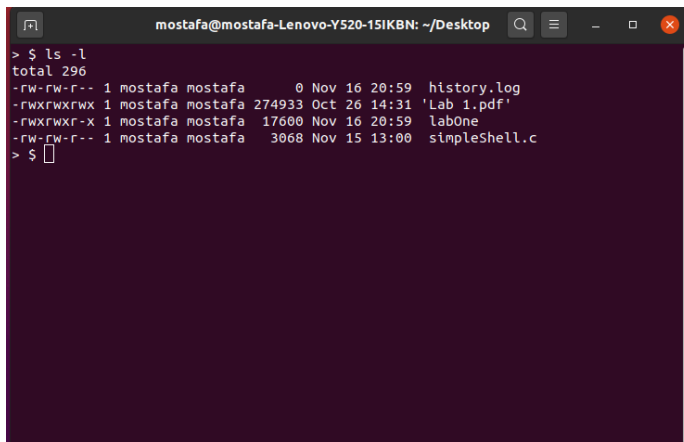
Commands with no arguments:



```
mostafa@mostafa-Lenovo-Y520-15IKBN: ~/Desktop
> $ ls
history.log  'Lab 1.pdf'  lab0ne  simpleShell.c
> $
```

Using command “ls” with no arguments to list directory in UNIX shell.

Commands with arguments:

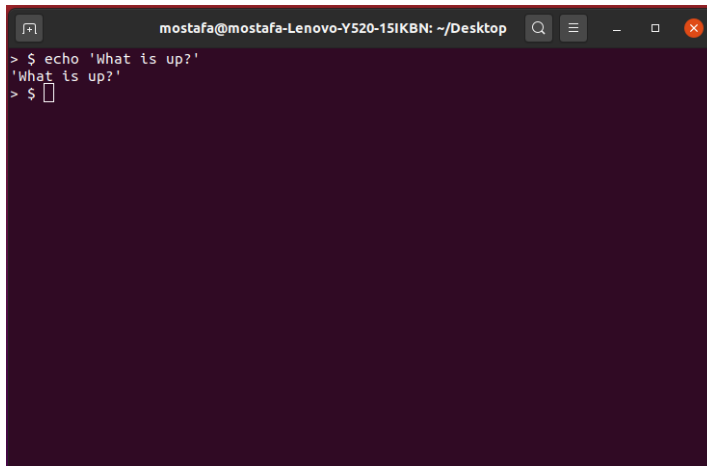


```
mostafa@mostafa-Lenovo-Y520-15IKBN: ~/Desktop
> $ ls -l
total 296
-rw-rw-r-- 1 mostafa mostafa    0 Nov 16 20:59 history.log
-rwxrwxrwx 1 mostafa mostafa 274933 Oct 26 14:31 'Lab 1.pdf'
-rwxrwxr-x 1 mostafa mostafa  17600 Nov 16 20:59 lab0ne
-rw-rw-r-- 1 mostafa mostafa   3068 Nov 15 13:00 simpleShell.c
> $
```

Using command “ls” with argument “-l” to list directory in UNIX shell with long format.

Sample Runs

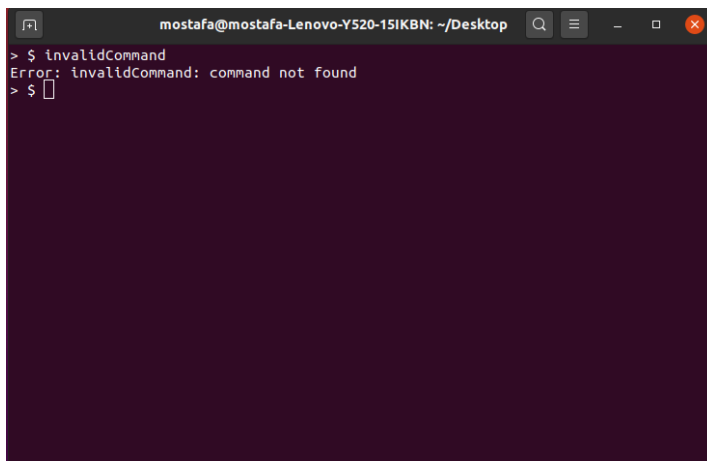
Commands with variable arguments:

A terminal window titled 'mostafa@mostafa-Lenovo-Y520-15IKBN: ~/Desktop' with search, menu, and window control icons. The prompt is '> \$'. The command 'echo 'What is up?'' is entered and executed, resulting in the output 'What is up?'. The prompt returns to '> \$' with a cursor.

```
> $ echo 'What is up?'  
'What is up?'  
> $
```

Using “echo” command is used to display line of text “What is up?” that are passed as an argument.

Invalid commands:

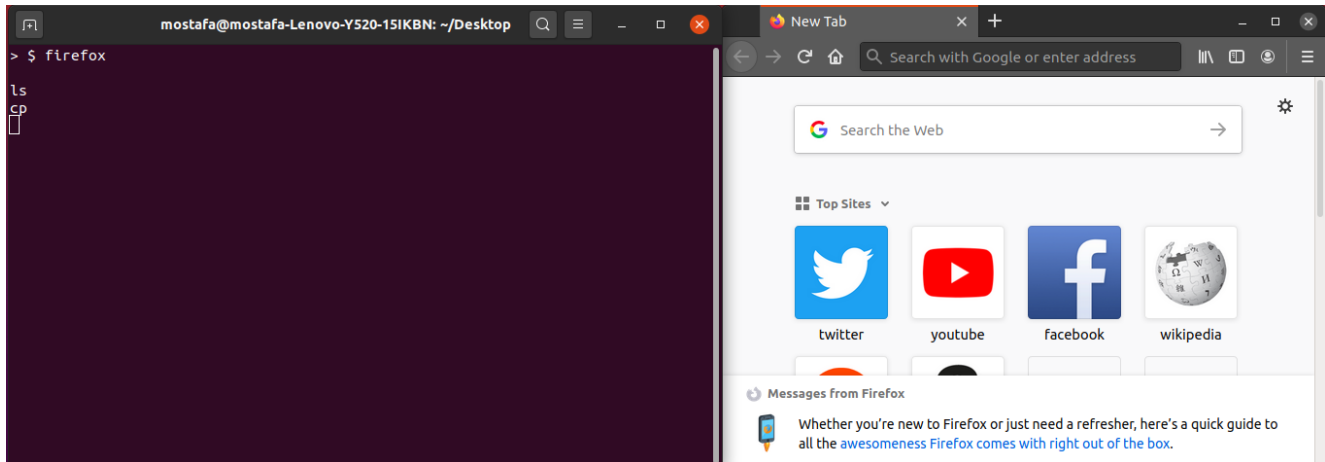
A terminal window titled 'mostafa@mostafa-Lenovo-Y520-15IKBN: ~/Desktop' with search, menu, and window control icons. The prompt is '> \$'. The command 'invalidCommand' is entered and executed, resulting in the error message 'Error: invalidCommand: command not found'. The prompt returns to '> \$' with a cursor.

```
> $ invalidCommand  
Error: invalidCommand: command not found  
> $
```

Entering invalid command to receive Error “Command not found”.

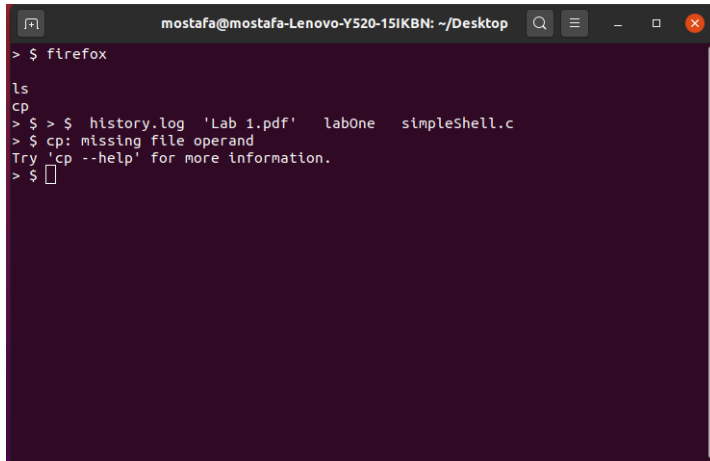
Sample Runs

Commands stopping other processes:



Firefox was process was born after command “Firefox” was called. It’s run in the foreground. Thus, all next commands are put in the queue waiting for the first process to terminate.

Even though “ls” and “cp” are called, no action occurred because firefox is still running.



As soon as firefox process was terminated, all pending operations in queue, “ls” and “cp”, take action.

Sample Runs

Commands running in background:

The screenshot shows a terminal window on the left and a system monitor on the right. The terminal window displays the following commands and output:

```

> $ firefox &
> $ ls
history.log 'Lab 1.pdf' labOne simpleShell.c
> $

```

The system monitor on the right shows a list of processes. The 'firefox' process is highlighted, indicating it is running in the background.

Process Name	User	% CPU	ID	Memory	Disk read t
gnome-terminal-server	mostafa	0	4171	10.5 MiB	1.3 M
bash	mostafa	0	4179	1.5 MiB	25.8 M
labOne	mostafa	0	4218	84.0 KiB	124.2 M
firefox	mostafa	0	4646	103.2 MiB	396.0 K
Privileged Con	mostafa	0	4700	41.6 MiB	N
WebExtension	mostafa	0	4760	23.2 MiB	92.0 K
Web Content	mostafa	0	4814	14.3 MiB	N

Adding an "&" to the end of the "firefox" command forces it to run in the background allowing all next processes to take action even though the firefox process isn't terminated yet.

Firefox process (ID = 4646) is running in the background as a child of the UNIX shell.

The history log is empty because no processes in the background were terminated.

Sample Runs

Terminating process in background:

The screenshot shows a Linux desktop environment. On the left, a terminal window displays the following commands and output:

```

> $ firefox &
> $ ls
history.log 'Lab 1.pdf' labOne simpleShell.c
> $

```

Below the terminal, a text editor window titled 'history.log' shows the message: '1 Process 4646: terminated successfully in the background.'

On the right, a process manager window displays a list of running processes. The process 'labOne' is highlighted in orange, indicating it is the process being terminated. The process list includes columns for Process Name, User, % CPU, ID, Memory, and Disk read to.

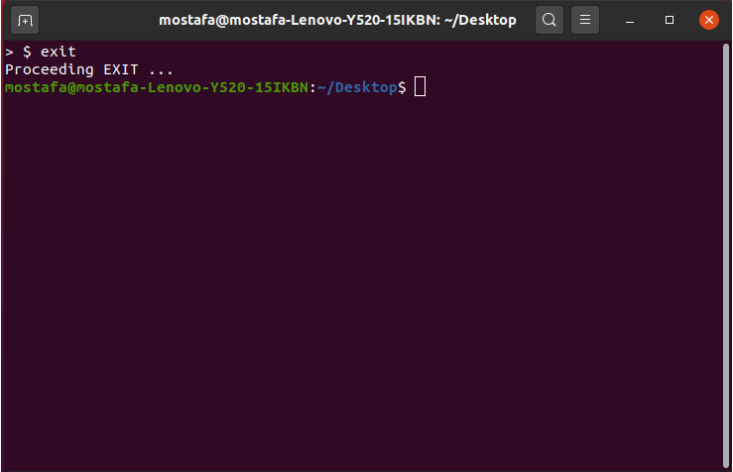
Process Name	User	% CPU	ID	Memory	Disk read to
gsd-media-keys	mostafa	0	2121	7.2 MiB	324.0 KiB
gsd-power	mostafa	0	2124	6.5 MiB	232.0 KiB
gsd-print-notifications	mostafa	0	2126	1.4 MiB	N/A
gsd-rfkill	mostafa	0	2127	620.0 KiB	N/A
gsd-screensaver-proxy	mostafa	0	2128	452.0 KiB	N/A
gsd-sharing	mostafa	0	2131	1.7 MiB	N/A
gsd-smartcard	mostafa	0	2132	1.0 MiB	N/A
gsd-sound	mostafa	0	2135	1.2 MiB	N/A
gsd-usb-protection	mostafa	0	2137	968.0 KiB	44.0 KiB
gsd-wacom	mostafa	0	2141	6.3 MiB	12.0 KiB
gsd-wwan	mostafa	0	2142	1.1 MiB	40.0 KiB
gsd-xsettings	mostafa	0	2143	6.4 MiB	116.0 KiB
gsd-printer	mostafa	0	2200	1.9 MiB	N/A
snap-store	mostafa	0	2270	171.8 MiB	30.1 MiB
xdg-document-portal	mostafa	0	2282	628.0 KiB	220.0 KiB
gvfsd-metadata	mostafa	0	2341	656.0 KiB	212.0 KiB
evolution-calendar-factor	mostafa	0	2355	11.4 MiB	5.2 MiB
evolution-addressbook-f	mostafa	0	2383	4.0 MiB	1.7 MiB
gnome-terminal-server	mostafa	0	4171	10.5 MiB	1.3 MiB
bash	mostafa	0	4179	1.5 MiB	25.8 MiB
labOne	mostafa	0	4218	84.0 KiB	124.6 MiB
tracker-store	mostafa	0	4922	11.4 MiB	4.0 KiB
gedit	mostafa	0	4986	17.3 MiB	4.0 KiB
gnome-keyring-daemon	mostafa	0	1820	936.0 KiB	N/A
gdm-x-session	mostafa	0	1830	644.0 KiB	N/A
gnome-session-binary	mostafa	0	1903	1.8 MiB	3.6 MiB
ssh-agent	mostafa	0	1976	452.0 KiB	N/A

Firefox process was terminated in the background and its parent process has no childs anymore.

The UNIX shell won't mark any changes, but the terminated process sends a signal to add note in the history log that it was terminated.

Sample Runs

Exiting shell:

A terminal window titled 'mostafa@mostafa-Lenovo-Y520-15IKBN: ~/Desktop' with standard window controls. The terminal shows a user entering the command '\$ exit', followed by the system response 'Proceeding EXIT ...'. The prompt then changes to 'mostafa@mostafa-Lenovo-Y520-15IKBN:~/Desktop\$' with a cursor. The terminal background is dark purple.

```
> $ exit
Proceeding EXIT ...
mostafa@mostafa-Lenovo-Y520-15IKBN:~/Desktop$
```

EXIT command terminates the shell and return to original terminal.