SHELL WITH FUNCTIONALITIES

by

NIMISHA SWAIN – 19BCE1860

UTKARSHA OJHA – 19BLC1110

NANDHINI S – 19BLC1001

A project report submitted to

Dr. Rabindra Kumar Singh

School Of Computer Science Engineering

in partial fulfillment of the requirements for the course

CSE2005: OPERATING SYSTEMS

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vandalur – Kelambakkam Road

Chennai – 600127

WINTER SEMESTER 2020-21

BONAFIDE CERTIFICATE

This is to certify that the Project work titled “Shell with functionalities” is being submitted by

NIMISHA SWAIN – 19BCE1860

UTKARSHA OJHA – 19BLC1110

NANDHINI S – 19BLC1001

for the course is a record of bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University.

Dr. Rabindra Kumar Singh
Associate Professor
SCOPE
VIT University, Chennai
Chennai – 600 127.

ABSTRACT

In this project we will implement our own shell that is a simplified version of a fully featured shell like bash. This will be an interactive shell which is a loop that prompts, executes, and waits. In this project, we will implement a shell with two features: standard input/output redirections and pipelines.

By default, standard input and output refer to the terminal. However, they can be redirected to read from and write to files, respectively. Pipes are another form of redirections. Instead of redirecting to a file, a pipe connects the standard output of one process to the standard input of another.

A pipe is a special unidirectional file descriptor with a single read end and a single write end. To generate a pipe, we will use pipe(2) system call which will return two file descriptors, one for reading and one for writing.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, Dr. RK SINGH for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

OBJECTIVE

We all use the built in terminal window in Linux distributions like Ubuntu, Fedora, etc.

The objective of this project is to learn how they actually work.

We are going to handle some features and algorithms what actually work inside a shell. All Linux operating systems have a terminal window to write in commands. We learn how they are executed properly after they are entered.
We understand how extra features like keeping the history of commands and showing help handled by creating your own shell.

INTRODUCTION

After a command is entered, the following things are done:

1. Command is entered and if length is non-null, keep it in history.
2. Parsing :Parsing is the breaking up of commands into individual words and strings
3. Checking for special characters like pipes, etc is done
4. Checking if built-in commands are asked for.
5. If pipes are present, handling pipes.
6. Executing system commands and libraries by forking a child and calling execvp.
7. Printing current directory name and asking for next input.

WHAT IS A SHELL?

A shell is special user program which provide an interface to user to use operating system services. Shell accept human readable commands from user and convert them into something which kernel can understand. It is a command language interpreter that execute commands read from input devices such as keyboards or from files. The shell gets started when the user logs in or start the terminal.

There are many reasons to write shell scripts –

- To avoid repetitive work and automation
- System admins use shell scripting for routine backups
- System monitoring
- Adding new functionality to the shell etc.

Our entire project is implemented using C language using a GCC compiler and we have used the following libraries:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/wait.h>
#include <assert.h>
#include <unistd.h>
```

CODE

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/wait.h>
#include <unistd.h>
#include <assert.h>
#include <ctype.h>

//macros
#define BASHPRINT "os_project_shell> "
#define INT_MAX 2147483647
#define MAXVAL 1023
#define READ_PERMISSION 0444
#define WRITE_PERMISSION 0666

/**
 * Control structure for a string tokenizer.  Maintains the
 * tokenizer's state.
 */
typedef struct tokenizer {
  char *str;                    /* the string to parse */
  char *pos;                            /* position in string */
} TOKENIZER;
```

```
/**
 * Initializes the tokenizer
 *
 * @param string the string that will be tokenized.  Should be non-NULL.
 * @return an initialized string tokenizer on success, NULL on error.
 */
TOKENIZER *init_tokenizer( char *string )
{
  TOKENIZER *tokenizer;
  int len;
  assert( string != NULL );

  tokenizer = (TOKENIZER *)malloc(sizeof(TOKENIZER));
  assert( tokenizer != NULL );
  len = strlen(string) + 1;         /* don't forget \0 char */
  tokenizer->str = (char *)malloc(len);
  assert( tokenizer->str != NULL );
  memcpy( tokenizer->str, string, len );
  tokenizer->pos = tokenizer->str;
  return tokenizer;
}



/**
 * Deallocates space used by the tokenizer.
```

```
  * @param tokenizer a non-NULL, initialized string
tokenizer
 */
void free_tokenizer( TOKENIZER *tokenizer )
{
  assert( tokenizer != NULL );
  free( tokenizer->str );
  free( tokenizer );
}



/**
 * Retrieves the next token in the string.  The returned
token is
 * malloc'd in this function, so you should free it when
done.
 *
 * @param tokenizer an initiated string tokenizer
 * @return the next token
 */
char *get_next_token( TOKENIZER *tokenizer )
{
  assert( tokenizer != NULL );
  char *startptr = tokenizer->pos;
  char *endptr;
  char *tok;

  if( *tokenizer->pos == '\0' )  /* handle end-case */
    return NULL;
  /* if current position is a delimiter, then return it */
  if( (*startptr == '|') || (*startptr == '&') ||
      (*startptr == '<') || (*startptr == '>') ) {
    tok = (char *)malloc(2);
    tok[0] = *startptr;
    tok[1] = '\0';
    tokenizer->pos++;
    return tok;
  }

  while( isspace(*startptr) )  /* remove initial white
spaces */
    startptr++;

  if( *startptr == '\0' )
    return NULL;

  /* go until next character is a delimiter */
  endptr = startptr;
  for( ;; ) {
         if( (*(endptr+1) == '|') || (*(endptr+1) == '&') ||
(*(endptr+1) == '<') ||
(*(endptr+1) == '>') || (*(endptr+1) == '\0') ||
(isspace(*(endptr+1))) ) {
      tok = (char *)malloc( (endptr - startptr) + 2 );
      memcpy( tok, startptr, (endptr - startptr) + 1 );
         tok[(endptr - startptr) + 1] = '\0'; /* null-terminate the
string */
      tokenizer->pos = endptr + 1;
         while( isspace(*tokenizer->pos) ) /* remove trailing
white space */
tokenizer->pos++;
   return tok;
  }
  endptr++;
 }

 assert( 0 );                    /* should never reach here */
 return NULL;                    /* but satisfy compiler */
}

pid_t pid;

// Write to the buffer
void writeBuffer(int fd, char *buffer)
{
for (int i = 0; buffer[i] != '\0'; i++)
{
    if (write(fd, &buffer[i], sizeof(char)) < 0)
    {
        perror("invalid write was not successful\n");
        exit(EXIT_FAILURE);
    }
}
}

// Read the input command from the user.
void readBuffer(int fd, char *readInput)
{
char data = '\0';
int readback = 0;
int i = 0;
while (data != '\n')
```

```
{
    // Read byte by byte.
    readback = read(fd, &data, sizeof(char));

    if (readback == -1)
    {
        // Check if read system call gave an error.
        perror("invalid read was not successful\n");
        exit(EXIT_FAILURE);
    }
    else if (readback == 0)
    {
        // Check for EOF and exit.
        writeBuffer(STDOUT_FILENO, "^D\n");
        exit(EXIT_SUCCESS);
    }
    else if (i < MAXVAL)
    { // Store in the buffer till the max value.
        readInput[i] = data;
        i++;
    }
}
// Remove the \r and add a null terminator.
readInput[--i] = '\0';
}

void getCommand(char *input, char **command)
{
TOKENIZER *tokenizer;
char *temp = input;
char *tok;
tokenizer = init_tokenizer(temp);
```

```
tok = get_next_token(tokenizer);
*command = (char *)malloc(sizeof(char) * strlen(tok));
memset(*command, '\0', sizeof(char) * strlen(tok));
memcpy(*command, tok, strlen(tok));
*(*command + strlen(tok)) = '\0';
free(tok);                          /* free the token now that
we're done with it */
free_tokenizer(tokenizer); /* free memory */
}

bool getArgs(char *input, char **args, char **inputArgs,
char **outputArgs, int isFirstCommand, int
isLastCommand)
{
TOKENIZER *tokenizer;
char *tok;
tokenizer = init_tokenizer(input);
int j = 0, inRedirection = 0, outRedirection = 0,
endOfCommand = 0;
int countInputRedirection = 0, countOutputRedirection = 0;

while ((tok = get_next_token(tokenizer)) != NULL)
{
    if (inRedirection == 1)
    {
        *inputArgs = (char *)malloc(sizeof(char) *
strlen(tok));
        memset(*inputArgs, '\0', sizeof(char) *
strlen(tok));
        memcpy(*inputArgs, tok, strlen(tok));
        *(*inputArgs + strlen(tok)) = '\0';
        inRedirection = 0;
    }
    if (outRedirection == 1)
    {
        *outputArgs = (char *)malloc(sizeof(char) *
strlen(tok));
        memset(*outputArgs, '\0', sizeof(char) *
strlen(tok));
        memcpy(*outputArgs, tok, strlen(tok));
        *(*outputArgs + strlen(tok)) = '\0';
        outRedirection = 0;
    }
    if (tok[0] == '<')
    {
        if ((isLastCommand == 1 && isFirstCommand
== 1) || (isFirstCommand == 1 && isLastCommand != 1))
// is the only command without any pipes or is the first
command
        {
            inRedirection = 1;
            endOfCommand = 1;
            countInputRedirection++;
        }
        if (isLastCommand == 1 && isFirstCommand
!= 1) // is the last command
        {
            writeBuffer(STDOUT_FILENO,
"Invalid: Multiple input redirections\n");
            return false;
        }
        if (isLastCommand != 1 && isFirstCommand
!= 1) // is the middle command
        {
            writeBuffer(STDOUT_FILENO,
"Invalid: Multiple input redirections\n");
            return false;
        }
    }
    if (tok[0] == '>')
    {
        if ((isLastCommand == 1 && isFirstCommand
== 1) || (isFirstCommand != 1 && isLastCommand == 1))
// is the only command without any pipes os is the last
command
        {
            outRedirection = 1;
            endOfCommand = 1;
            countOutputRedirection++;
        }
        if (isFirstCommand == 1 && isLastCommand
!= 1) // is the first command
        {
            writeBuffer(STDOUT_FILENO,
"Invalid: Multiple output redirections\n");
            return false;
        }
        if (isLastCommand != 1 && isFirstCommand
!= 1) // is the middle command
        {
            writeBuffer(STDOUT_FILENO,
"Invalid: Multiple input redirections\n");
            return false;
        }
    }
    if (tok[0] != '<' && tok[0] != '>' && endOfCommand
== 0)
    {
        args[j] = (char *)malloc(sizeof(char) *
strlen(tok));
        memset(args[j], '\0', sizeof(char) * strlen(tok));
        memcpy(args[j], tok, strlen(tok));
        *(args[j] + strlen(tok)) = '\0';
        j++;
    }
    free(tok); /* free the token now that we're done with it
*/
}
free_tokenizer(tokenizer); /* free memory */
args[j] = NULL;
if (countInputRedirection > 1)
{
    writeBuffer(STDOUT_FILENO, "Invalid: Multiple
standard input redirects or redirect in invalid location\n");
    return false;
}
if (countOutputRedirection > 1)
{
    writeBuffer(STDOUT_FILENO, "Invalid: Multiple
standard output redirects\n");
    return false;
}
return true;
}

int getCommands(char *input, char **args)
{
```

```
char *temp;
int i, j = 0;
temp = input;

while(*temp == ' ')
        temp++;

while (*temp != '\0')
{
        i = 0;
        //Parse each arguments and save in the args pointer.
        char    *tempcomm    =    (char    *)malloc(MAXVAL    *
sizeof(char));

        while (*temp != '|' && *temp != '\0')
        {
                tempcomm[i++] = *temp;
                temp++;
        }

        if (*temp != '\0')
        {
                temp++;
        }

        if (i != 0)
        {
                tempcomm[i] = '\0';
                args[j++] = tempcomm;
        }

        // Free the memory.
        //free(tempcomm);
}
args[j] = (char *)0;

return j;
}

bool setRedirectionFd(char *inputFd, char *outputFd, int
inPipefd, int outPipeFd)
{
int dupOutput = -1;
if (inPipefd == -1 && outPipeFd == -1) // It is a single
command without any pipes
{
        if (strlen(inputFd) != 0) // Set STDIN_FILENO to
input fd provided in the command
        {
                int fdInput = open(inputFd, O_RDONLY,
READ_PERMISSION);
                if (fdInput == -1)
                {
                        writeBuffer(STDOUT_FILENO, "Invalid
standard input redirect: No such file or directory\n");
                        return false;
                }
                dupOutput = dup2(fdInput, STDIN_FILENO);
                if (dupOutput == -1)
                {
                        perror("Invalid dup");
                        return false;
                }
        }
```

```
    if (strlen(outputFd) != 0) // Set STDOUT_FILENO to output fd provided in the command
    {
        int fdOutput = open(outputFd, O_WRONLY | O_CREAT | O_TRUNC, WRITE_PERMISSION);
        if (fdOutput == -1)
        {
            writeBuffer(STDOUT_FILENO, "Invalid standard output redirect: File creation failed\n");
            return false;
        }
        dupOutput = dup2(fdOutput, STDOUT_FILENO);
        if (dupOutput == -1)
        {
            perror("Invalid dup");
            return false;
        }
    }
    return true;
}
else if (inPipefd != -1 && outPipeFd != -1) // It is a middle command surrounded by pipes
{
    dupOutput = dup2(inPipefd, STDIN_FILENO); //dup STDIN_FILENO to read end of pipe
    if (dupOutput == -1)
    {
        perror("Invalid dup");
        return false;
    }
    dupOutput = dup2(outPipeFd, STDOUT_FILENO); //dup STDOUT_FILENO to write end of pipe
    if (dupOutput == -1)
    {
        perror("Invalid dup");
        return false;
    }
    return true;
}
else if (inPipefd == -1 && outPipeFd != -1) // It is the first
command followed by pipes
{
    if (strlen(inputFd) != 0)
    {
        int fdInput = open(inputFd, O_RDONLY,
READ_PERMISSION);
        if (fdInput == -1)
        {
            writeBuffer(STDOUT_FILENO, "Invalid
standard input redirect: No such file or directory\n");
            return false;
        }
        dupOutput = dup2(fdInput, STDIN_FILENO);
        if (dupOutput == -1)
        {
            perror("Invalid dup");
            return false;
        }
    }
    dupOutput = dup2(outPipeFd, STDOUT_FILENO);
    if (dupOutput == -1)
    {
        perror("Invalid dup");
        return false;
    }
    return true;
}
else if (inPipefd != -1 && outPipeFd == -1) // It is the last
command that gets input from pipe
{
    if (strlen(outputFd) != 0)
    {
        int fdOutput = open(outputFd, O_WRONLY |
O_CREAT | O_TRUNC, WRITE_PERMISSION);
        if (fdOutput == -1)
        {
            writeBuffer(STDOUT_FILENO, "Invalid
standard output redirect: File creation failed\n");
            return false;
        }
        dupOutput = dup2(fdOutput,
STDOUT_FILENO);
        if (dupOutput == -1)
        {
            perror("Invalid dup");
            return false;
        }
    }
    dupOutput = dup2(inPipefd, STDIN_FILENO);
    if (dupOutput == -1)
    {
        perror("Invalid dup");
        return false;
    }
    return true;
}
```

```
else
{
    // should have never come here !
}
return true;
}

// Handle Ctrl+C.
void ctrlcHandle(int sig)
{
}

// Main function call.
int main(int argc, char *argv[])
{
//registers a signal ctrl c
signal(SIGINT, ctrlcHandle);

while (1)
{
    // Initialize the variable.
    char input[MAXVAL];
    char *commAll[MAXVAL];
    int numProcess = 0;

    // Write the Prompt statement: penn-shredder#
    writeBuffer(STDOUT_FILENO,                    (char
*)BASHPRINT);

    // Read the command from the user.
    readBuffer(STDIN_FILENO, input);
    int l = 0;

    while(input[l] == ' ')
        l++;

    int commands = getCommands(input, commAll);

    if(commands == 0)
        continue;

    pid_t pidArray[commands];
    int fd[commands - 1][2], i;

    for (i = 0; i < (commands - 1); i++)
    {
        // Open the pipe.
        if (pipe(fd[i]) < 0)
        {
            perror("invalid pipe failed");
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < commands; i++)
    {
        char *arguments[MAXVAL];
        char *command = NULL, *inputFd = "",
*outputFd = "";
        int isLastCommand = 0, isFirstCommand = 0;

        // Parse the input for the command.
        getCommand(commAll[i], &command);
        if (i == (commands - 1))
            isLastCommand = 1;

        if (i == 0)
            isFirstCommand = 1;

        // Parse the arguments for the command.
        bool validCommand = getArgs(commAll[i],
arguments, &inputFd, &outputFd, isFirstCommand,
isLastCommand);

        if (!validCommand)
            break;

        // Fork a child from parent.
        pid = fork();

        if (pid == 0)
        {
            // Child Process.
            bool allOkay = false;

            if (isLastCommand == 1 &&
isFirstCommand == 1) // is the only command without any
pipes
            {
                allOkay =
setRedirectionFd(inputFd, outputFd, -1, -1);
            }
            else if (i == 0) // is the first command
followed by a pipe
            {
```

```
                        close(fd[i][0]);
                                                      //  close
the read end of the pipe
                        allOkay                              =
setRedirectionFd(inputFd, outputFd, -1, fd[i][1]); // set the
STDOUT to fd write end
                  }
                  else if (i == (commands - 1)) // is the last
command after a pipe
                  {
                        close(fd[i - 1][1]);
                                                      //  close
the write end of the pipe
                        allOkay                              =
setRedirectionFd(inputFd, outputFd, fd[i - 1][0], -1); // set
the STDIN to fd read end
                  }
                  else // is the middle command enclosed
by pipes on both sides
                  {
                        // close the read end of the next
pipe
                        allOkay                              =
setRedirectionFd(inputFd, outputFd, fd[i - 1][0], fd[i][1]);
// set the STDIN to fd read end
                        close(fd[i - 1][1]);
                                                             //
close the write end of the previous pipe
                        close(fd[i][0]);
                  }
                  if (!allOkay)
                        break;

                  for (int j = 0; j < (commands - 1); j++)
```

```
            {

                // Close both the file descriptors.
                close(fd[j][0]);
                close(fd[j][1]);
            }

            int   execvpOutput   =   execvp(command,
arguments);
            if (execvpOutput == -1)
            {
                perror("Invalid execvp");
                exit(1);
            }
        }
        else
        {
            // Parent Process
            // Add pid to the list of childs list
            pidArray[i] = pid;
            numProcess++;
        }
        // free(command);
    }

    // Close all the pipes.
    for (int j = 0; j < (commands - 1); j++)
    {

        // Close both the file descriptors.
        close(fd[j][0]);
        close(fd[j][1]);
        }

        for (int i = 0; i < numProcess; i++)
        {
                int status;
                pid_t w = waitpid(pidArray[i], &status,
WUNTRACED | WCONTINUED);
                if (w == -1)
                {
                        perror("invalid waitpid error");
                        exit(EXIT_FAILURE);
                }
        }
}
}
```

COMMANDS IMPLEMENTED

echo hello Nimisha – display user name

pwd – ask for password

ls – for listing files

sleep 5 - to suspend the calling process for a specified time

cat >filename1 - input in file

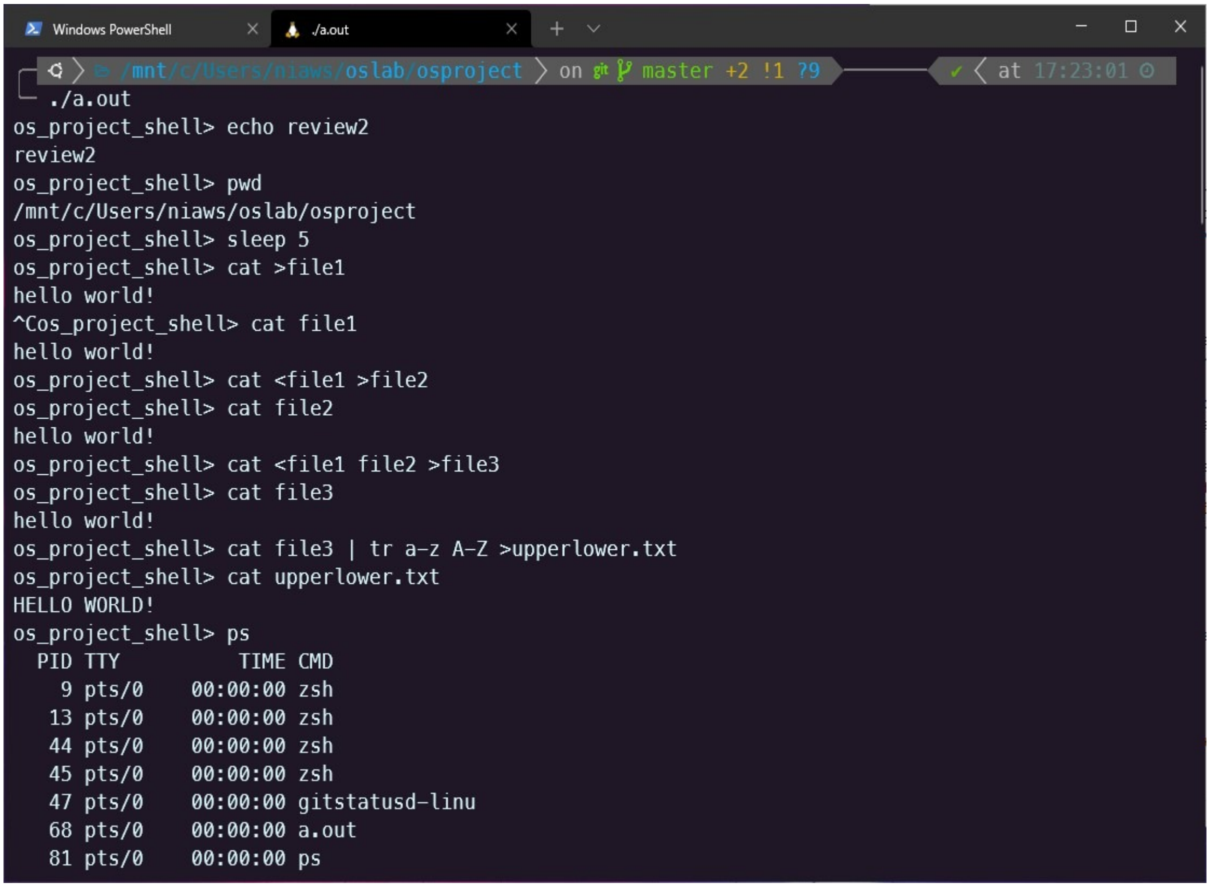
cat filename - display file contents

cat <filename1 - display file contents

cat <filename1 >filename2 - copy contents of file1 into file2

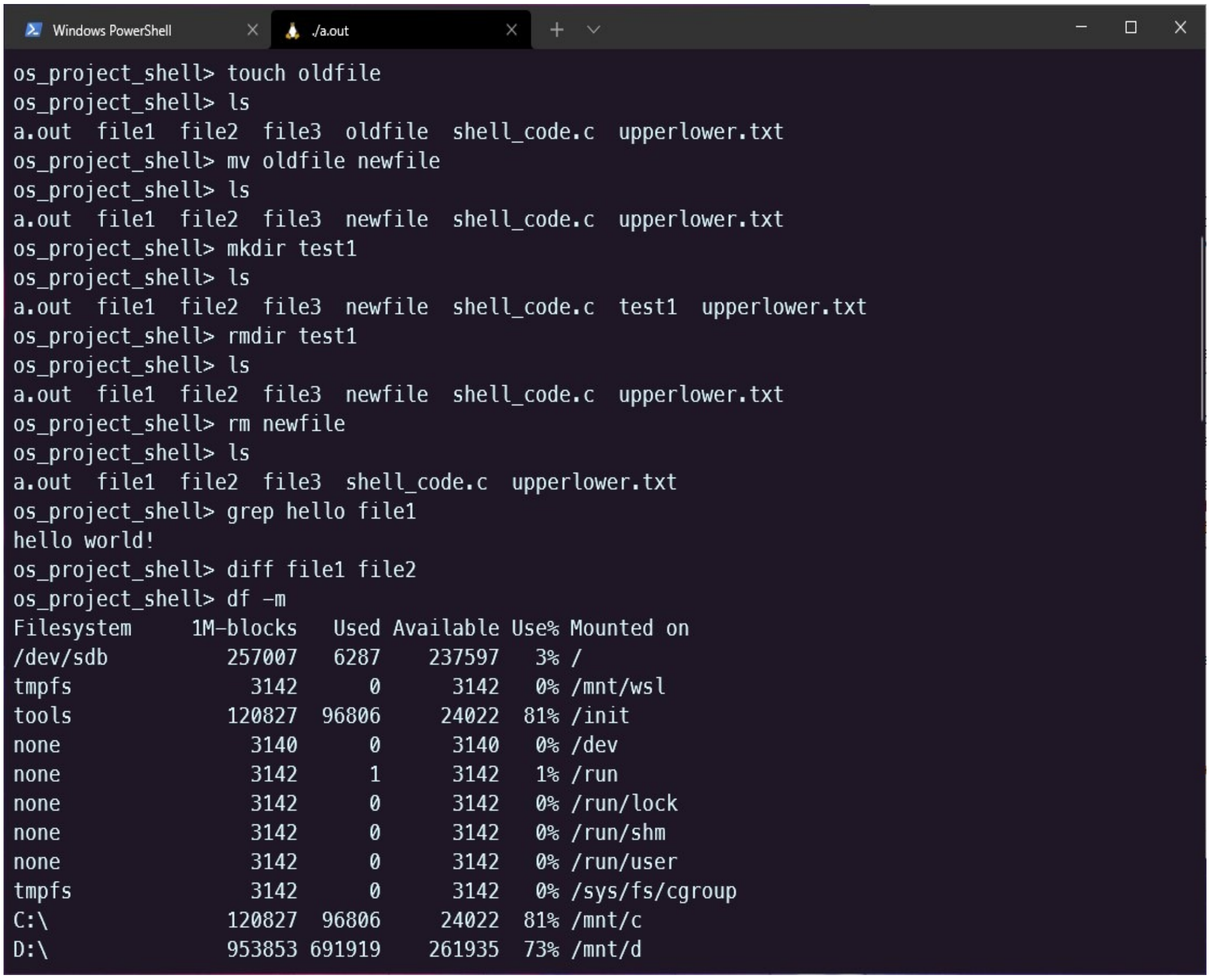
cat <filename1 filename2>filename3 - joins file1 and file2 and copy content in file3

cat review2 | tr a-z A-Z >upperlower.txt - convert lower to upper vice-versa



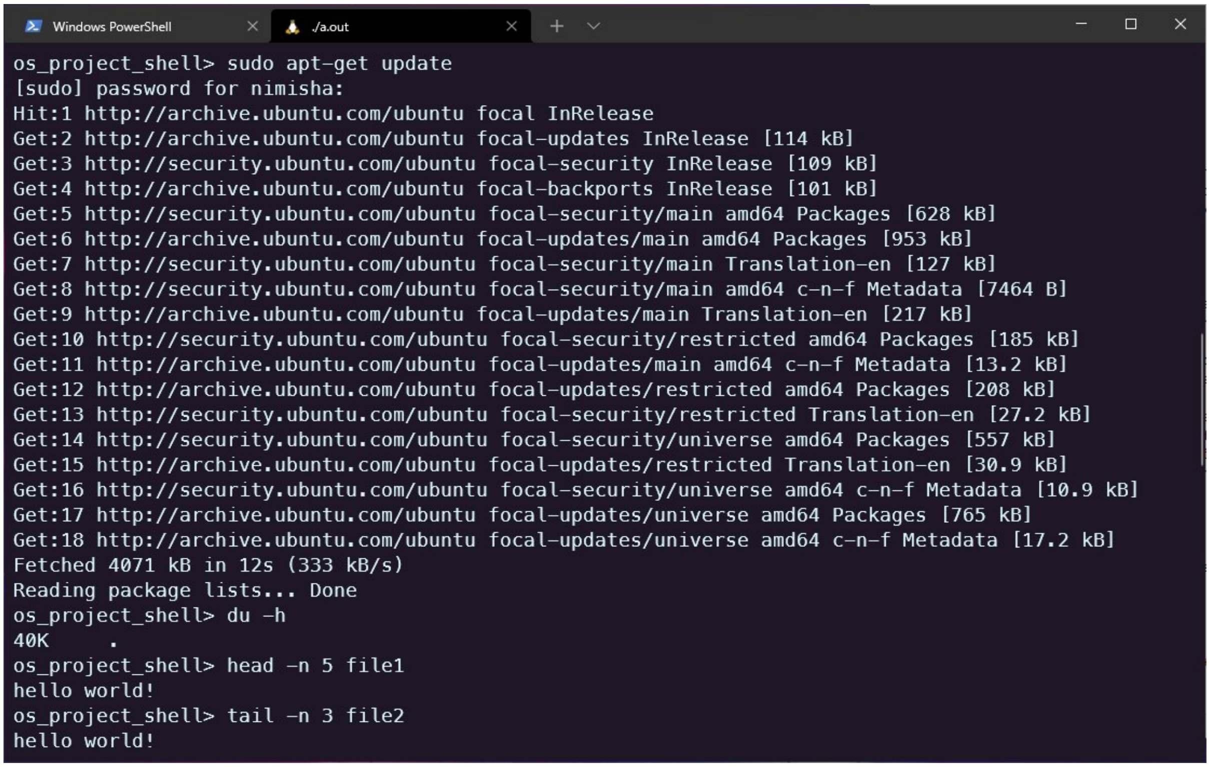
```
/mnt/c/Users/niaws/oslab/osproject on master +2 !1 ?9 ──────── ✓ at 17:23:01
 ./a.out
os_project_shell> echo review2
review2
os_project_shell> pwd
/mnt/c/Users/niaws/oslab/osproject
os_project_shell> sleep 5
os_project_shell> cat >file1
hello world!
^Cos_project_shell> cat file1
hello world!
os_project_shell> cat <file1 >file2
os_project_shell> cat file2
hello world!
os_project_shell> cat <file1 file2 >file3
os_project_shell> cat file3
hello world!
os_project_shell> cat file3 | tr a-z A-Z >upperlower.txt
os_project_shell> cat upperlower.txt
HELLO WORLD!
os_project_shell> ps
  PID TTY          TIME CMD
    9 pts/0    00:00:00 zsh
   13 pts/0    00:00:00 zsh
   44 pts/0    00:00:00 zsh
   45 pts/0    00:00:00 zsh
   47 pts/0    00:00:00 gitstatusd-linu
   68 pts/0    00:00:00 a.out
   81 pts/0    00:00:00 ps
```

- code filename.c create/open filename
- touch filename - create blank filename
- mv oldfile newfile -rename file
- mkdir foldername - make new folder/directory
- rmdir foldername - to delete a folder or directory
- rm filename - delete a file
- grep lineword filename - display line containing word
- df -m - display system's disc space usage



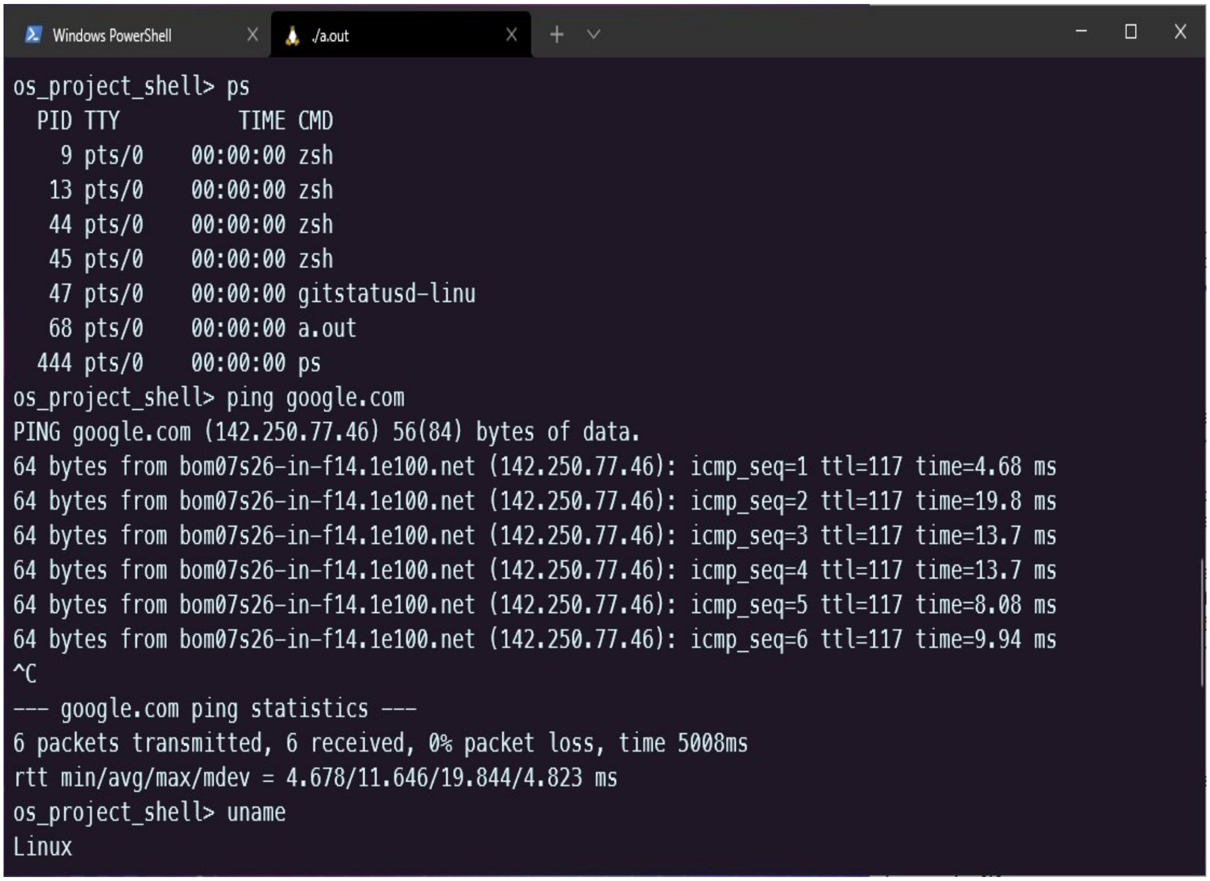
```
os_project_shell> touch oldfile
os_project_shell> ls
a.out  file1  file2  file3  oldfile  shell_code.c  upperlower.txt
os_project_shell> mv oldfile newfile
os_project_shell> ls
a.out  file1  file2  file3  newfile  shell_code.c  upperlower.txt
os_project_shell> mkdir test1
os_project_shell> ls
a.out  file1  file2  file3  newfile  shell_code.c  test1  upperlower.txt
os_project_shell> rmdir test1
os_project_shell> ls
a.out  file1  file2  file3  newfile  shell_code.c  upperlower.txt
os_project_shell> rm newfile
os_project_shell> ls
a.out  file1  file2  file3  shell_code.c  upperlower.txt
os_project_shell> grep hello file1
hello world!
os_project_shell> diff file1 file2
os_project_shell> df -m
Filesystem     1M-blocks   Used Available Use% Mounted on
/dev/sdb          257007   6287    237597   3% /
tmpfs               3142      0      3142   0% /mnt/wsl
tools             120827  96806     24022  81% /init
none                3140      0      3140   0% /dev
none                3142      1      3142   1% /run
none                3142      0      3142   0% /run/lock
none                3142      0      3142   0% /run/shm
none                3142      0      3142   0% /run/user
tmpfs               3142      0      3142   0% /sys/fs/cgroup
C:\               120827  96806     24022  81% /mnt/c
D:\               953853 691919    261935  73% /mnt/d
```

- sudo apt-get update - update
- du -h -how much space a file or a directory takes
- head -n 5 filename   - view
- the first five lines of any text file
- tail -n 5 filename   - view the last five lines of any text file



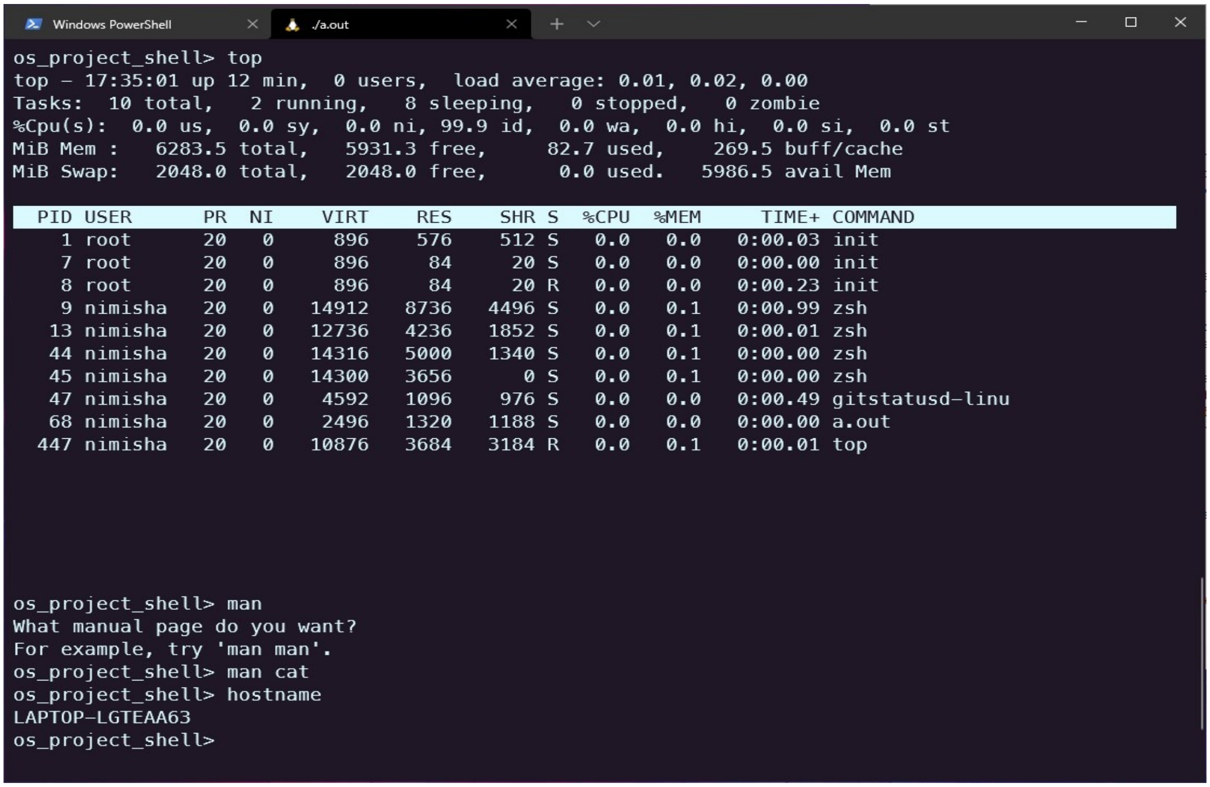
```
os_project_shell> sudo apt-get update
[sudo] password for nimisha:
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security InRelease [109 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [628 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [953 kB]
Get:7 http://security.ubuntu.com/ubuntu focal-security/main Translation-en [127 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [7464 B]
Get:9 http://archive.ubuntu.com/ubuntu focal-updates/main Translation-en [217 kB]
Get:10 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [185 kB]
Get:11 http://archive.ubuntu.com/ubuntu focal-updates/main amd64 c-n-f Metadata [13.2 kB]
Get:12 http://archive.ubuntu.com/ubuntu focal-updates/restricted amd64 Packages [208 kB]
Get:13 http://security.ubuntu.com/ubuntu focal-security/restricted Translation-en [27.2 kB]
Get:14 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [557 kB]
Get:15 http://archive.ubuntu.com/ubuntu focal-updates/restricted Translation-en [30.9 kB]
Get:16 http://security.ubuntu.com/ubuntu focal-security/universe amd64 c-n-f Metadata [10.9 kB]
Get:17 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages [765 kB]
Get:18 http://archive.ubuntu.com/ubuntu focal-updates/universe amd64 c-n-f Metadata [17.2 kB]
Fetched 4071 kB in 12s (333 kB/s)
Reading package lists... Done
os_project_shell> du -h
40K     .
os_project_shell> head -n 5 file1
hello world!
os_project_shell> tail -n 3 file2
hello world!
```

- ps - display running process
- ping website.com - show ping

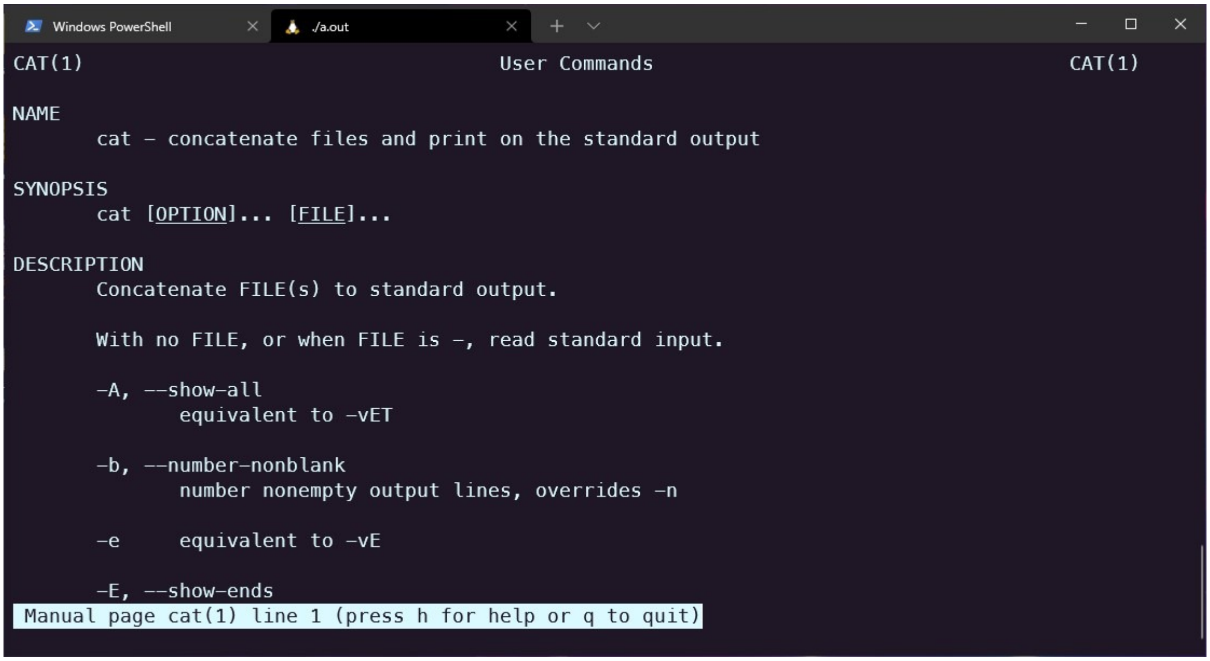


```
os_project_shell> ps
  PID TTY          TIME CMD
    9 pts/0    00:00:00 zsh
   13 pts/0    00:00:00 zsh
   44 pts/0    00:00:00 zsh
   45 pts/0    00:00:00 zsh
   47 pts/0    00:00:00 gitstatusd-linu
   68 pts/0    00:00:00 a.out
  444 pts/0    00:00:00 ps
os_project_shell> ping google.com
PING google.com (142.250.77.46) 56(84) bytes of data.
64 bytes from bom07s26-in-f14.1e100.net (142.250.77.46): icmp_seq=1 ttl=117 time=4.68 ms
64 bytes from bom07s26-in-f14.1e100.net (142.250.77.46): icmp_seq=2 ttl=117 time=19.8 ms
64 bytes from bom07s26-in-f14.1e100.net (142.250.77.46): icmp_seq=3 ttl=117 time=13.7 ms
64 bytes from bom07s26-in-f14.1e100.net (142.250.77.46): icmp_seq=4 ttl=117 time=13.7 ms
64 bytes from bom07s26-in-f14.1e100.net (142.250.77.46): icmp_seq=5 ttl=117 time=8.08 ms
64 bytes from bom07s26-in-f14.1e100.net (142.250.77.46): icmp_seq=6 ttl=117 time=9.94 ms
^C
--- google.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5008ms
rtt min/avg/max/mdev = 4.678/11.646/19.844/4.823 ms
os_project_shell> uname
Linux
```

- top - display list of running processes
- man - manual
- hostname - display hostname



```
os_project_shell> top
top - 17:35:01 up 12 min,  0 users,  load average: 0.01, 0.02, 0.00
Tasks:  10 total,   2 running,   8 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   6283.5 total,   5931.3 free,     82.7 used,    269.5 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   5986.5 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    1 root      20   0     896    576    512 S   0.0   0.0   0:00.03 init
    7 root      20   0     896     84     20 S   0.0   0.0   0:00.00 init
    8 root      20   0     896     84     20 R   0.0   0.0   0:00.23 init
    9 nimisha   20   0   14912   8736   4496 S   0.0   0.1   0:00.99 zsh
   13 nimisha   20   0   12736   4236   1852 S   0.0   0.1   0:00.01 zsh
   44 nimisha   20   0   14316   5000   1340 S   0.0   0.1   0:00.00 zsh
   45 nimisha   20   0   14300   3656      0 S   0.0   0.1   0:00.00 zsh
   47 nimisha   20   0    4592   1096    976 S   0.0   0.0   0:00.49 gitstatusd-linu
   68 nimisha   20   0    2496   1320   1188 S   0.0   0.0   0:00.00 a.out
  447 nimisha   20   0   10876   3684   3184 R   0.0   0.1   0:00.01 top






os_project_shell> man
What manual page do you want?
For example, try 'man man'.
os_project_shell> man cat
os_project_shell> hostname
LAPTOP-LGTEAA63
os_project_shell>
```

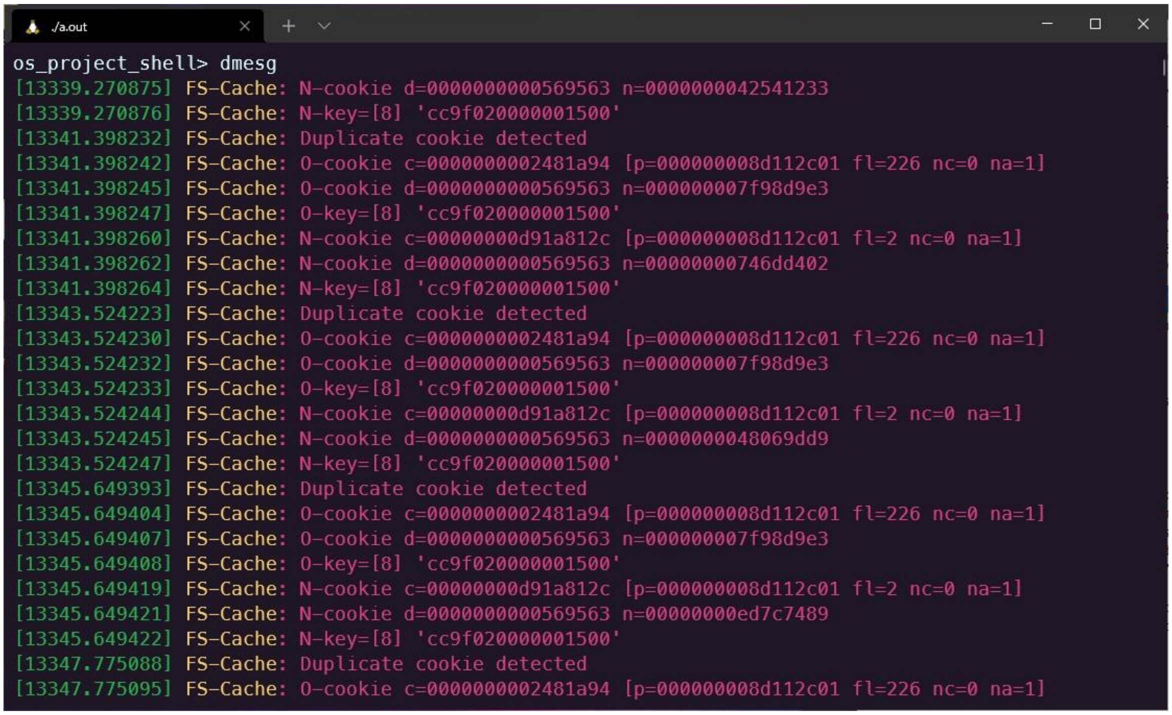


```
CAT(1)                           User Commands                          CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s) to standard output.

       With no FILE, or when FILE is -, read standard input.

       -A, --show-all
              equivalent to -vET

       -b, --number-nonblank
              number nonempty output lines, overrides -n

       -e     equivalent to -vE

       -E, --show-ends
 Manual page cat(1) line 1 (press h for help or q to quit)
```

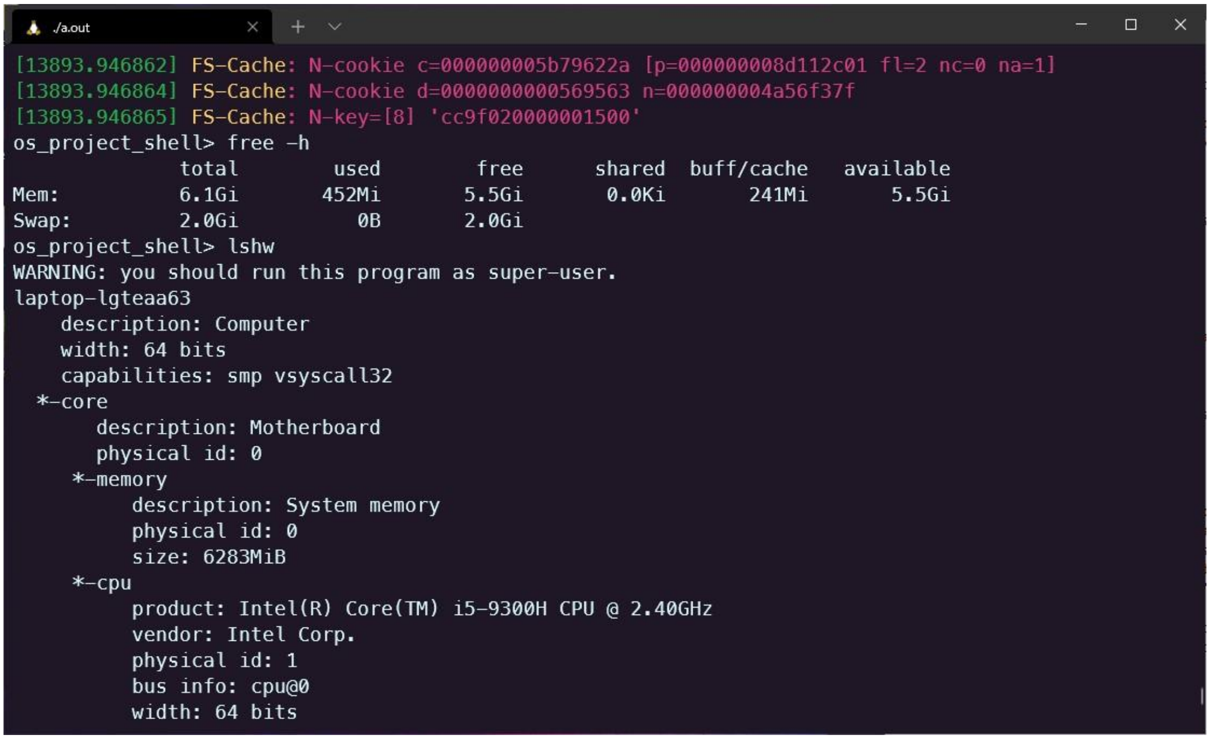
dmesg - show bootup messages.

free -h -free and used memory.

lshw - hardware configuration.



```
os_project_shell> dmesg
[13339.270875] FS-Cache: N-cookie d=0000000000569563 n=0000000042541233
[13339.270876] FS-Cache: N-key=[8] 'cc9f020000001500'
[13341.398232] FS-Cache: Duplicate cookie detected
[13341.398242] FS-Cache: O-cookie c=0000000002481a94 [p=000000008d112c01 fl=226 nc=0 na=1]
[13341.398245] FS-Cache: O-cookie d=0000000000569563 n=000000007f98d9e3
[13341.398247] FS-Cache: O-key=[8] 'cc9f020000001500'
[13341.398260] FS-Cache: N-cookie c=00000000d91a812c [p=000000008d112c01 fl=2 nc=0 na=1]
[13341.398262] FS-Cache: N-cookie d=0000000000569563 n=00000000746dd402
[13341.398264] FS-Cache: N-key=[8] 'cc9f020000001500'
[13343.524223] FS-Cache: Duplicate cookie detected
[13343.524230] FS-Cache: O-cookie c=0000000002481a94 [p=000000008d112c01 fl=226 nc=0 na=1]
[13343.524232] FS-Cache: O-cookie d=0000000000569563 n=000000007f98d9e3
[13343.524233] FS-Cache: O-key=[8] 'cc9f020000001500'
[13343.524244] FS-Cache: N-cookie c=00000000d91a812c [p=000000008d112c01 fl=2 nc=0 na=1]
[13343.524245] FS-Cache: N-cookie d=0000000000569563 n=0000000048069dd9
[13343.524247] FS-Cache: N-key=[8] 'cc9f020000001500'
[13345.649393] FS-Cache: Duplicate cookie detected
[13345.649404] FS-Cache: O-cookie c=0000000002481a94 [p=000000008d112c01 fl=226 nc=0 na=1]
[13345.649407] FS-Cache: O-cookie d=0000000000569563 n=000000007f98d9e3
[13345.649408] FS-Cache: O-key=[8] 'cc9f020000001500'
[13345.649419] FS-Cache: N-cookie c=00000000d91a812c [p=000000008d112c01 fl=2 nc=0 na=1]
[13345.649421] FS-Cache: N-cookie d=0000000000569563 n=00000000ed7c7489
[13345.649422] FS-Cache: N-key=[8] 'cc9f020000001500'
[13347.775088] FS-Cache: Duplicate cookie detected
[13347.775095] FS-Cache: O-cookie c=0000000002481a94 [p=000000008d112c01 fl=226 nc=0 na=1]
```

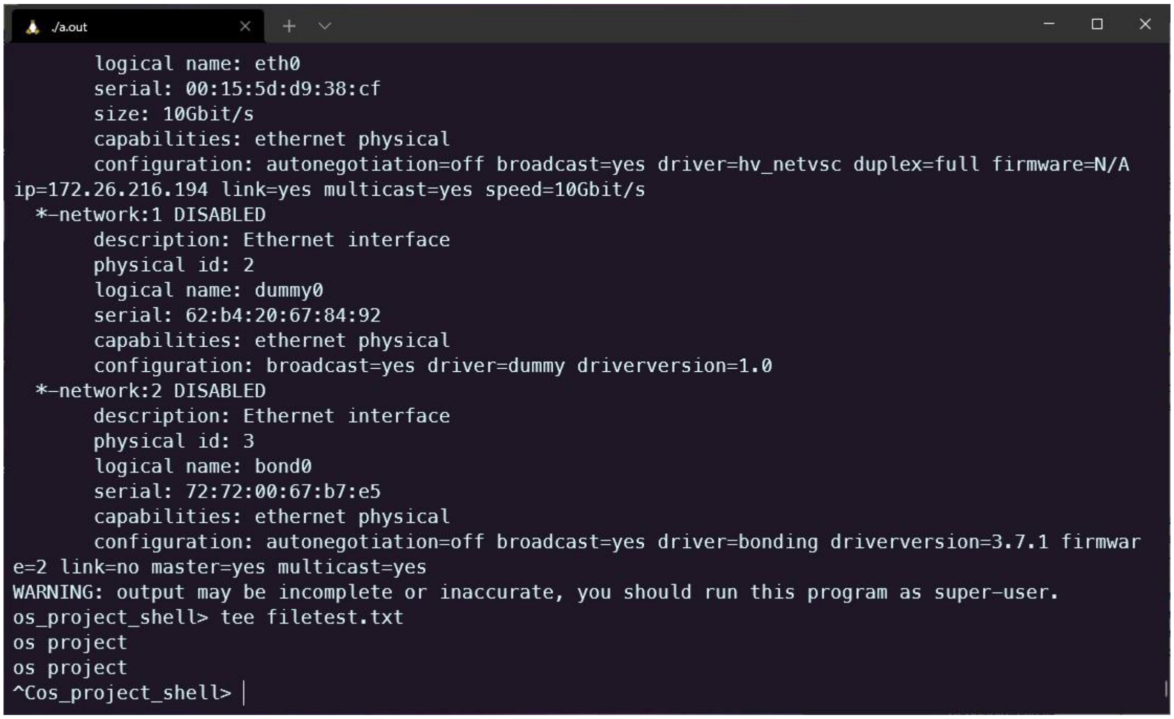


```
[13893.946862] FS-Cache: N-cookie c=000000005b79622a [p=000000008d112c01 fl=2 nc=0 na=1]
[13893.946864] FS-Cache: N-cookie d=0000000000569563 n=000000004a56f37f
[13893.946865] FS-Cache: N-key=[8] 'cc9f020000001500'
os_project_shell> free -h
              total        used        free      shared  buff/cache   available
Mem:          6.1Gi       452Mi       5.5Gi       0.0Ki       241Mi       5.5Gi
Swap:         2.0Gi          0B       2.0Gi
os_project_shell> lshw
WARNING: you should run this program as super-user.
laptop-lgteaa63
    description: Computer
    width: 64 bits
    capabilities: smp vsyscall32
  *-core
       description: Motherboard
       physical id: 0
     *-memory
          description: System memory
          physical id: 0
          size: 6283MiB
     *-cpu
          product: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
          vendor: Intel Corp.
          physical id: 1
          bus info: cpu@0
          width: 64 bits
```

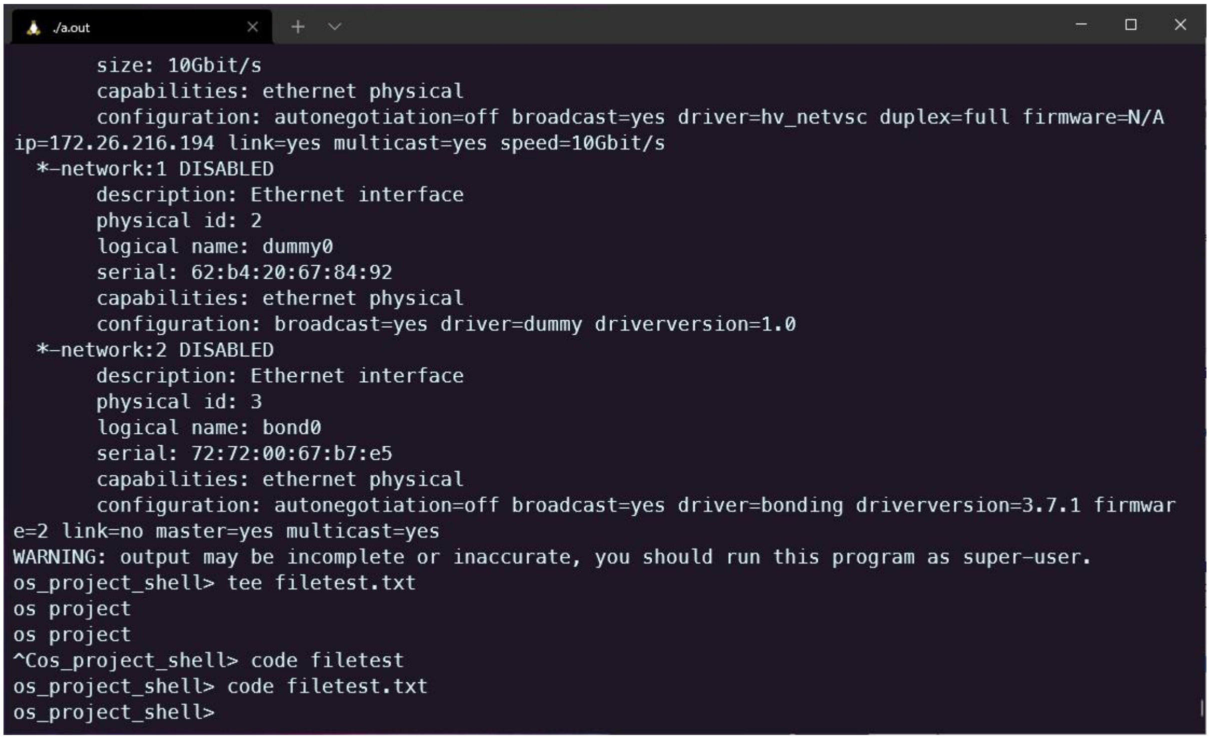
tee filetest.txt - display in terminal as well as write in file.

vi/nano - open file

clear



```
       logical name: eth0
       serial: 00:15:5d:d9:38:cf
       size: 10Gbit/s
       capabilities: ethernet physical
       configuration: autonegotiation=off broadcast=yes driver=hv_netvsc duplex=full firmware=N/A
ip=172.26.216.194 link=yes multicast=yes speed=10Gbit/s
  *-network:1 DISABLED
       description: Ethernet interface
       physical id: 2
       logical name: dummy0
       serial: 62:b4:20:67:84:92
       capabilities: ethernet physical
       configuration: broadcast=yes driver=dummy driverversion=1.0
  *-network:2 DISABLED
       description: Ethernet interface
       physical id: 3
       logical name: bond0
       serial: 72:72:00:67:b7:e5
       capabilities: ethernet physical
       configuration: autonegotiation=off broadcast=yes driver=bonding driverversion=3.7.1 firmwar
e=2 link=no master=yes multicast=yes
WARNING: output may be incomplete or inaccurate, you should run this program as super-user.
os_project_shell> tee filetest.txt
os project
os project
^Cos_project_shell> |
```



```
       size: 10Gbit/s
       capabilities: ethernet physical
       configuration: autonegotiation=off broadcast=yes driver=hv_netvsc duplex=full firmware=N/A
ip=172.26.216.194 link=yes multicast=yes speed=10Gbit/s
  *-network:1 DISABLED
       description: Ethernet interface
       physical id: 2
       logical name: dummy0
       serial: 62:b4:20:67:84:92
       capabilities: ethernet physical
       configuration: broadcast=yes driver=dummy driverversion=1.0
  *-network:2 DISABLED
       description: Ethernet interface
       physical id: 3
       logical name: bond0
       serial: 72:72:00:67:b7:e5
       capabilities: ethernet physical
       configuration: autonegotiation=off broadcast=yes driver=bonding driverversion=3.7.1 firmwar
e=2 link=no master=yes multicast=yes
WARNING: output may be incomplete or inaccurate, you should run this program as super-user.
os_project_shell> tee filetest.txt
os project
os project
^Cos_project_shell> code filetest
os_project_shell> code filetest.txt
os_project_shell>
```

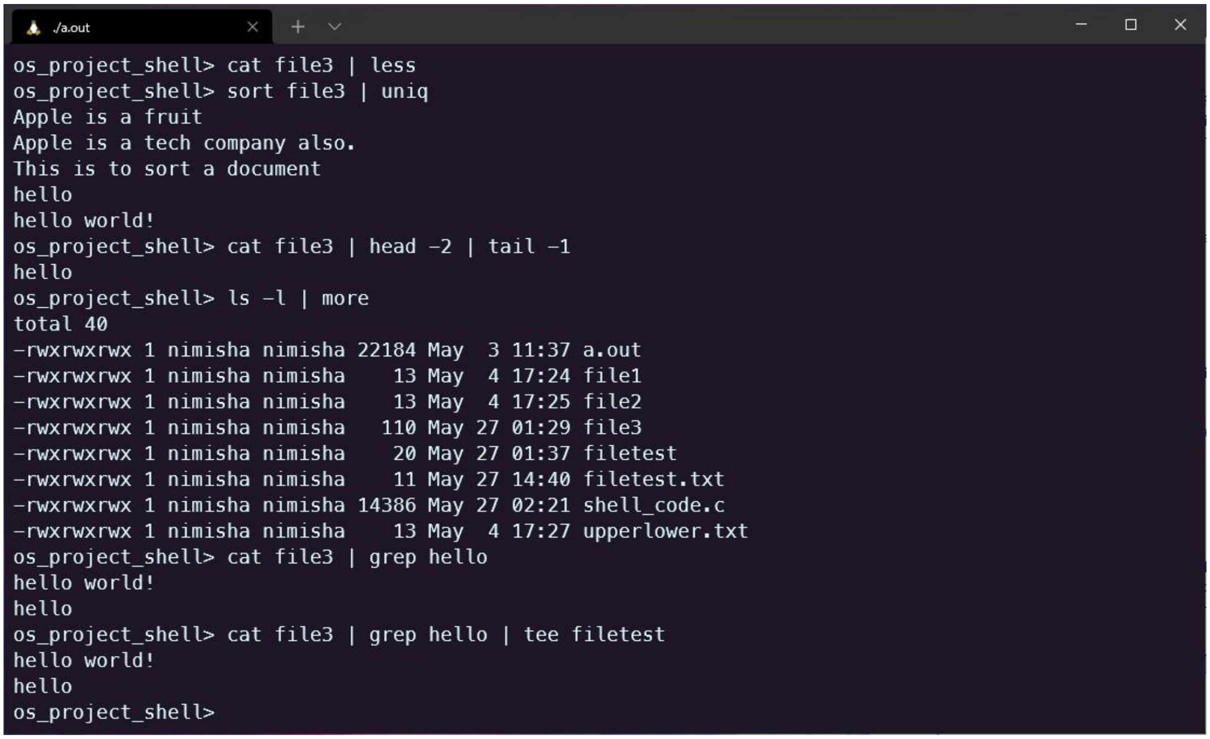
PIPELINE :

This “pipe” command is readily available on UNIX/Linux platforms. This command pipes the output of the previous command to the next command. There are literally TONS of situations where this method offers serious value.Before jumping deeper, there’s something to know of. Every single program in the UNIX/Linux system has 3 built-in data streams.

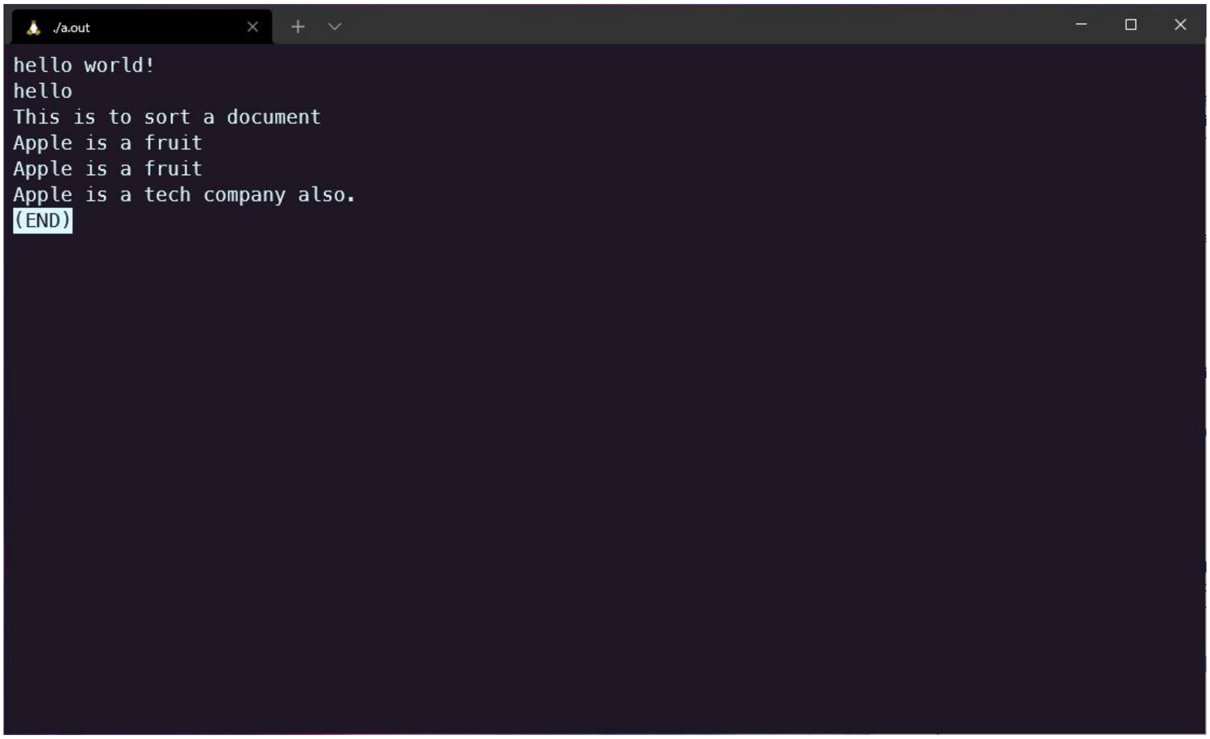
- STDIN (0) – Standard input
- STDOUT (1) – Standard output
- STDERR (2) – Standard error

PIPELINE COMMANDS :

cat file3 | less - less content of file
sort file3 | uniq - sort and display unique values.
cat file3 | head -2 | tail -1 - display lines.
ls -l | more - Listing all files and directories and give it as input to more command.
ls -l | find ./ -type f -name "*.txt" -exec grep "program" {} \;
cat file3 | grep hello
cat file3 | grep hello | tee filetest

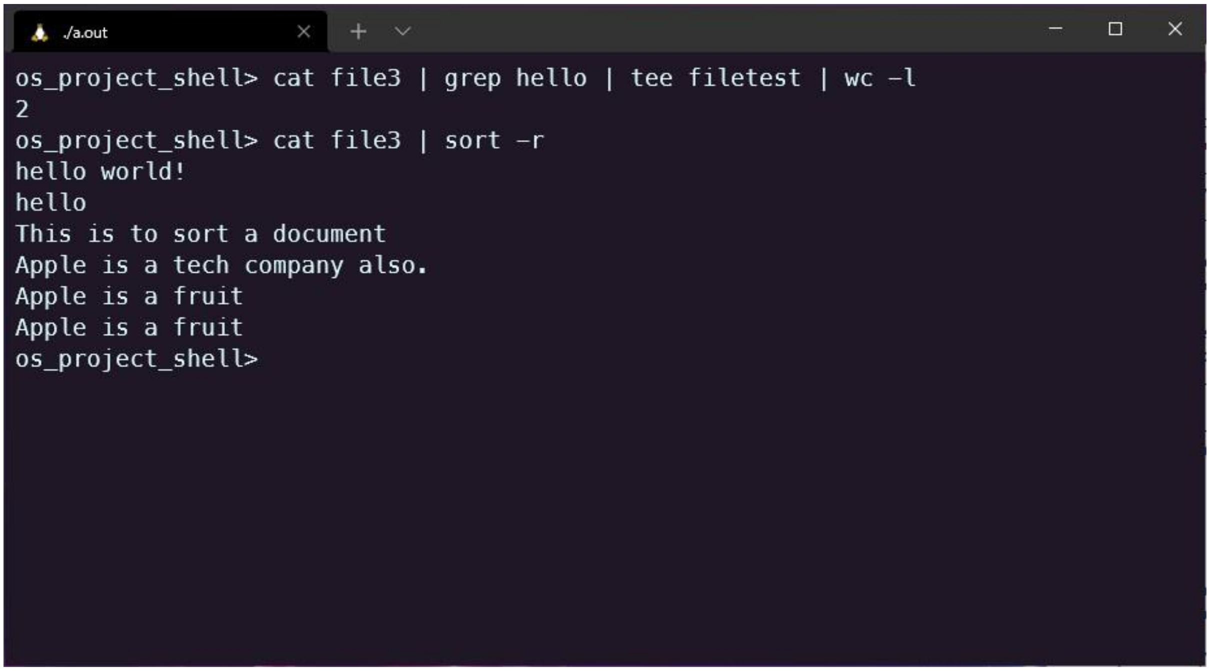


```
os_project_shell> cat file3 | less
os_project_shell> sort file3 | uniq
Apple is a fruit
Apple is a tech company also.
This is to sort a document
hello
hello world!
os_project_shell> cat file3 | head -2 | tail -1
hello
os_project_shell> ls -l | more
total 40
-rwxrwxrwx 1 nimisha nimisha 22184 May  3 11:37 a.out
-rwxrwxrwx 1 nimisha nimisha    13 May  4 17:24 file1
-rwxrwxrwx 1 nimisha nimisha    13 May  4 17:25 file2
-rwxrwxrwx 1 nimisha nimisha   110 May 27 01:29 file3
-rwxrwxrwx 1 nimisha nimisha    20 May 27 01:37 filetest
-rwxrwxrwx 1 nimisha nimisha    11 May 27 14:40 filetest.txt
-rwxrwxrwx 1 nimisha nimisha 14386 May 27 02:21 shell_code.c
-rwxrwxrwx 1 nimisha nimisha    13 May  4 17:27 upperlower.txt
os_project_shell> cat file3 | grep hello
hello world!
hello
os_project_shell> cat file3 | grep hello | tee filetest
hello world!
hello
os_project_shell>
```



```
hello world!
hello
This is to sort a document
Apple is a fruit
Apple is a fruit
Apple is a tech company also.
(END)
```

cat file3 | grep hello | tee filetest | wc -l
cat file3 | sort -r



```
os_project_shell> cat file3 | grep hello | tee filetest | wc -l
2
os_project_shell> cat file3 | sort -r
hello world!
hello
This is to sort a document
Apple is a tech company also.
Apple is a fruit
Apple is a fruit
os_project_shell>
```

CONCLUSION

This project introduced to some features and algorithms what actually work inside a shell. All Linux operating systems have a terminal window to write in commands. We learned how they are executed properly after they are entered. We understand how extra features like keeping the history of commands and showing help handled by creating your own shell.

We also explored a sample script and saw how shell scripts are useful in many circumstances. The difference between interpreted languages like shell scripts and compiled languages like C was discussed. Based on the information about different programming tools, you learned why different tools are chosen to complete different tasks.

In addition, this project discussed how the shell is a command programming language that provides an interface to the UNIX operating system. Its features include

1. control-flow primitives,
2. parameter passing,
3. variables and string substitution.

Constructs such as while, if then else, case and for are available. Two-way communication is possible between the shell and commands.

String-valued parameters, typically file names or flags, may be passed to a command. A return code is set by commands that may be used to determine control-flow, and the standard output from a command may be used as shell input.

The shell can modify the environment in which commands run. Input and output can be redirected to files, and processes that communicate through pipes can be invoked. Commands are found by searching directories in the file system in a sequence that can be defined by the user. Commands can be read either from the terminal or from a file, which allows command procedures to be stored for later use.We also explored the concepts and commands of pipelining which is readily available on UNIX/Linux platforms. This command pipes the output of the previous command to the next command. There are literally TONS of situations where this method offers serious value.

This project has provided us a great learning experience and we wish to explore more about the same in near future.

BIODATA

Name : Nandhini S
Mobile Number : 9632538098
E-mail : nandhini.s2019@vitstudent.ac.in
Permanent Address : SMR Vinay Galaxy, Hoodi ,Whitefield,
Bangalore, Karnataka.

Name : Nimisha Swain
Mobile Number : 9150744137
E-mail ID : nimisha.swain2019@vitstudent.ac.in
Permanent Address : Qrt no . 8/2 , Air Force station lonavala,
Maharashtra.

Name : Utkarsha Ojha
Mobile Number : 7250285004
E-mail ID : utkarsha.ojha2019@vitstudent.ac.in
Permanent Address : N-141,Shyamali colony,Doranda, Ranchi
Jharkhand.