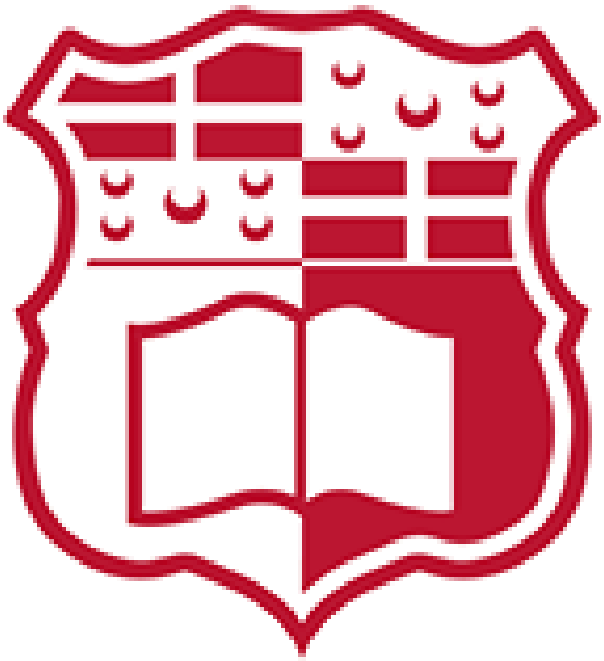
CPS1012 – Operating Systems and Systems Programming 1 Assignment



Faculty of ICT

By Nathan Bonavia Zammit

Course Code: CPS1012

Table of Contents

Bugs and Issues

The sections completed for the assignment are:

- Part 1: Shell Fundamentals
- Part 2:
    - Input, Tokenisation, and Parsing
    - Variables
    - Internal Commands
        - Exit command
        - Echo command
        - Cd command
        - Showvar command
        - Set command
        - Unset command
        - Showenv command – specific variable not attempted

These sections were tested and should work correctly. The report will now proceed by displaying test cases for the sections which work correctly, and give some explanation on the code that was written.

The other sections were not attempted, however the attempted sections hould work correctly with minimal bugs.

Design and Implementation

For my shell I decided to work with nodes, and made a struct as can be seen below:

```
typedef struct _node
{
    char* value;
    char is_pipe;
    struct _node* next;
} node;
```

[1]

I also created a struct for variables alongside space for a number of variables:

```
typedef struct _var
{
    char* name;
    char* value;
} var;

#define g_var_count 1024    // space for 1024 variables
var g_vars[g_var_count];
```

[2]

Input, Tokenisation, and Parsing

The majority of this was set up in the main(). For my assignment I decided to use nodes. For my tokenization I simply created an input buffer of size 4096, and made all the values null.

```
char input_buf[4096];                    // creates 4096 characters
memset(input_buf, 0, sizeof(input_buf));    // sets all characters in the buffer to 0/null
```
[3]

I then set up the input as can be seen below with a limit of 4095 characters:

```
const int rc = read(
    0,
    input_buf,
    sizeof(input_buf)
        - 1);   // reads the input from the keyboard, with a limit of 4095 characters
```
[4]

The first node is created, and another buffer of size 4096 is also created, and cleared/all its values are set to null.

```
node* first = 0;                // first node is created and set to null
char buf[4096];                 // buffer of size 4096 is created
memset(buf, 0, sizeof(buf));    // buffer is cleared
```
[5]

Variables to indicate if a backslash (esc_slash), or a double quote (esc_dblqte) are being used, to indicate if one is in the middle of escaping or qouting. A variable pointer (vp) is also created alongside the integer len which contains the length of input_buf, the buffer which the input is stored in.

```
char esc_dblqte = 0;    // variable to indicate if we are in the middle of quoting (")
char esc_slash = 0;     // variable to indicate if we are in the middle of escaping (\)
char* vp = 0;           // variable pointer
const int len = strlen(input_buf);
```
[6]

A for loop to go through each character that is inputted in the buffer is then declared alongside the pointer for the destination which the nodes will end up.

```
char* d = buf;                    // destination pointer
for (int i = 0; i < len; ++i)   // going through each character
```
[7]

The for loop starts to go through every character that is inputted. If the character is a '\', this is indicated by the esc_slash flag we declared earlier. The flag is then reset, and the escaped character after the '\' is taken. If the character is a '|' then piping is being attempted. In this case, a node is added with value null, and the is_pipe flag is given the value 1, which is displayed. If '\\' is inputted then a '\' has been escaped, and as therefore '\' must be taken as a character. If the character '$' is detected or the variable pointer is null, then vp as set to d, this will be explained later on.

```
for (int i = 0; i < len; ++i)   // going through each character
{
    char c = input_buf[i];

    if (esc_slash)
    {
        esc_slash = 0;  // reset '\' flag
        *d++ = c;         // indiscriminately take the escaped character
        continue;
    }

    if (c == '|')
    {
        add_node(&first, 0, 1);
        continue;
    }

    if (c == '\\')
    {
        esc_slash = 1;
        continue;
    }

    if (c == '$' && vp == 0)
    {
        vp = d;
        // is_variable = 1;
        continue;
    }
```

[8]

If the character inputted is a double quote, then first it is determined whether it is the opening quote or the closing quote, this is done through the esc_dblqte flag we made earlier, and then quoting will occur. Reminder that if there is a '\' in front of the double quote then that is taken as a character (this was referenced earlier [8]). If '{' is inputted and the variable pointer is not null, then for now it is skipped, but if it is followed by a '}' , as seen in [10], then expansion will occur.

```
if (c == '"')
{
    if (esc_dblqte) // if double quote is already set
    {
        // closing quote
        esc_dblqte = 0;
    }
    else
    {
        // opening quote
        esc_dblqte = 1;
    }
    continue;
}

if (c == '{' && vp != 0)
{
    // skip the opening { in case of variable
    continue;
}
```

[9]

If '}' is inputted, and the variable pointer is not null, then variable expansion will occur, and the value of the variable is obtained.

```
if (c == '}' && vp != 0)    // variable expansion occurs
{
    // take the variable value
    char* env = get_var(vp);
    if (env != 0 && env[0] != 0)
    {
        const int sz = sizeof(buf) - (vp - buf);
        strncpy(vp, env, sz - 1);
        d = buf + strlen(env);
    }
    else
    {
        // clear/reset the buffer
        memset(buf, 0, sizeof(buf));
        d = buf;
    }
    vp = 0;
    continue;
}
```

[10]

The function is_seperator() is called, this function checks whether the character is a separator or not. If it is, but it is in escape mode, then the literal character is taken. If it is a null or space, it is simple skipped, and if the variable pointer is now not equal to null, then expansion will occur. After all of this, the node is saved, and a token is created. The variable pointer is reset, and the buffer is cleared.

```
if (is_separator(c))
{
    if ((esc_slash || esc_dblqte))
    {
        // if in escape mode, take this character
        *d++ = c;   // take the character
        continue;
    }

    if (is_spaces_or_null(buf))
    {
        // skip white space
        continue;
    }

    if (vp != 0)    // this is where variable expansion is happening
    {
        char* env = get_var(vp);
        if (env != 0 && env[0] != 0)
        {
            const int sz = sizeof(buf) - (vp - buf);
            memset(vp, 0, sz);
            strncpy(vp, env, sz - 1);
        }
        vp = 0;
    }
    first = add_node(&first, buf, 0);
    vp = 0;
    // is_variable = 0; // always reset the is_variable flag when storing a node
    memset(buf, 0, sizeof(buf));    // clear the buffer
    d = buf;
```

[11]

This is for when a character is not a ‘special character’ and simply takes the character as is. The nodes are then printed/processed and then freed (cleared to not allocate a lot of memory).

```
    else
    {
        *d++ = c;   // take the character
        continue;
    }

    // continue;
}
if (buf[0] != 0 && !is_spaces_or_null(buf))
{
    if (vp != 0)
    {
        char* env = get_var(vp);
        if (env != 0 && env[0] != 0)
        {
            const int sz = sizeof(buf) - (vp - buf);
            memset(vp, 0, sz);
            strncpy(vp, env, sz - 1);
        }
    }
    first = add_node(&first, buf, 0);
    // is_variable = 0; // always reset the is_variable flag when storing a node
}
print_nodes(first);
process_nodes(first);
free_nodes(first);
```

[12]

Functions used for Input, Tokenisation & Parsing

char is_separator(const char c)

```
char is_separator(const char c) // function to check whether char is a seperator or not
{
    switch (c)
    {
    case ' ':
    case '\t':
    case '\n':
    case '|':
    case ';':
    case '<':
    case '>':
        return (char)1;

    default:
        return (char)0;
    }
}
```

[13]

The use of this function is to check whether a character is a separator, i.e. a space, a tab, an enter, a pipe, a semi-column, or redirection arrows. Otherwise, the character is none of those and the function is ignored. This is done through a switch case. It can be seen implemented in [11].

char is_spaces_or_null(char* buf)

```
char is_spaces_or_null(char* buf)   // function to check whether there is a space or a null
{
    if (buf == 0 || buf[0] == 0)    // if the contents/the pointer itself is 0
    {
        return 1;
    }
    for (char* p = buf; p != 0 && *p != 0; ++p)
    {
        if (*p != ' ')
        {
            return 0;
        }
    }
    return 1;
}
```

[14]

If a separator is detected, then this function is called to check whether it's a space or a null value. This can be seen in [11].

node* add_node(node** first, char* str, char is_pipe)

```
node* add_node(node** first, char* str, char is_pipe)   // function to add the nodes
{
    // nn = new node
    node* nn = (node*)malloc(sizeof(node)); // creates a node
    if (is_pipe)
    {
        nn->value = 0;      // if it is a pipe then the content of the node is null
        nn->is_pipe = 1;    // notified that it is a pipe
    }
    else
    {
        nn->value = strdup(str);    // string is copied into the node
        nn->is_pipe = 0;            // it is not a pipe
    }

    nn->next = 0;   // the latest node is added to the end, therefore next is set to null

    if (*first == 0)    // if the first node is null, then the new node is set as the first node
    {
        *first = nn;
    }
    else
    {
        // find the last node
        node* ln = *first;
        while (ln->next != 0)
        {
            ln = ln->next;  // move to the next node
        }
        ln->next = nn;  // add the new node to the end of the list of tokens
    }
    return *first;
}
```

[15]

This function is called whenever a new node is being created, i.e. a new token is made. It also is used to move to the next node, and if the is_pipe flag is turned on.

void print_nodes(node* first)

```
void print_nodes(node* first)   // function to print all the nodes
{
    int i = 0;
    node* n = first;
    while (n != 0)
    {
        printf(
            "[%3d]: %-30s [pipe?: %d]\n", i++, n->value, n->is_pipe);   // to left align the items
        n = n->next;
    }
    printf("\n\n\n");
}
```

[16]

This function is used to print the nodes that are created, i.e. the tokens are printed. Ig the is_pipe is on then [pipe?: 1] is printed on the right.

void free_nodes(node* first)

```
void free_nodes(node* first)
{
    node* n = first;
    while (n != 0)
    {
        if (n->value != 0)
        {
            free(n->value);
        }
        node* t = n;
        n = n->next;
        free(t);
    }
}
```

[17]

As mentioned in [12], the free_nodes() function is used to clear the nodes in order to not allocate any memory. This is done via the use of the free() function.

void process_nodes(node* first)

The process_nodes() function is used to go through most of the internal commands, and is of large size. It will be broken down further on in the “Internal Commands” part of the report.

Internal Commands

Exit Command

As can be seen below, I made the exit command for when the user inputs “exit”, or “quit”, the shell terminates. This was done through the use of the exit() function and if statements as seen below. This is done in the main().

```
if (strcmp(input_buf, "exit") == 0 || strcmp(input_buf, "quit") == 0) //to exit the shell - Internal Command
{
    printf("Exiting...\n");
    exit(0);
}
```

[18]

Echo Command

As can be seen below this is how I implemented the echo command. The quoting, expansion and quote removal that were mentioned before are applied when running said command. This internal command was placed in process_nodes(), where in this case, if the first node is equal to “echo” then it looks at the nodes after it and performs the echo command.

```
if (strcmp(curr_node->value, "echo") == 0) //echo - internal command
{
    node* nx = curr_node->next;
    if (nx != 0 && nx->value[0] != 0)
    {
        printf("%s\n", nx->value);
    }
}
```

[19]

Cd Command

As can be seen below this is how I implemented the cd command. This command can be found in the process_nodes() function, and when the first node inputted is equal to “cd”, then it checks the nodes after and changes the directories as required. If the directory inputted does not exist then an error is displayed to the user, through the use of the perror() function.

```
else if (strcmp(curr_node->value, "cd") == 0) //cd internal command
{
    node* nx = curr_node->next;
    if (nx != 0 && nx->value[0] != 0)
    {
        int rc = chdir(nx->value);
        if (rc == 0)
        {
            // success
            char newdir[2048];
            memset(newdir, 0, sizeof(newdir));
            printf("new dir: %s\n", getcwd(newdir, sizeof(newdir) - 1));
        }
        else
        {
            fprintf(stderr, "%s: ", nx->value);
            perror("");
        }
    }
}
```

[20]

Showvar Command

The use of the showvar command is to print the shell variables as a standard output one by one. This is done by simply inputting in “showvar” in the shell. If a specific variable is sought after, inputting “showvar *variable name*” will print out said variable. This command is found in the process_nodes() function.

```
else if (strcmp(curr_node->value, "showvar") == 0)
{
    // showvar cmd expects another entry after it;
    // get it
    node* next_node = curr_node->next;
    if (next_node != 0)
    {
        if (next_node->value != 0 && next_node->value[0] != 0)
        {
            printf(
                "command is: %s and value for the command is: %s\n",
                curr_node->value,
                next_node->value);
        }
        curr_node = next_node->next;
    }
    else
        print_all_vars();
}
```

[21]

```
void print_all_vars()
{
    for (int i = 0; i < g_var_count; ++i)
    {
        if (g_vars[i].name != 0 && g_vars[i].name[0] != 0)
        {
            printf("%s=%s\n", g_vars[i].name, g_vars[i].value);
        }
    }
}
```

[22]

Set Command

The use of this command is to create new variables.

For example, by inputting “set name = Nathan” then a new variable called name is created, which can be viewed using the showvar command [21]. This command is found in the process_nodes() function.

```
else if (strcmp(curr_node->value, "set") == 0)
{
    // set a=b
    node* n1 = curr_node->next; // n1 would be the 'a'
    node* n2 = n1->next;           // n2 would be the '='
    node* n3 = n2->next;           // n3 would be the 'b'
    if (n1 != 0 && n2 != 0 && n3 != 0)
    {
        char* v1 = n1->value;
        char* v2 = n2->value;
        char* v3 = n3->value;
        if ((v1 != 0 && v1[0] != 0) && (v2 != 0 && v2[0] != 0)
            && (v3 != 0 && v3[0] != 0))
        {
            if (strcmp(v2, "=") == 0)
            {
                add_var(v1, v3);
            }
        }
        // move past the 3 nodes just processed
        curr_node = n3;
    }
}
```

[23]

Unset Command

The use of this command is to do the opposite of the set command [22] and unset a variable from its value. This command is found in the process_nodes() function.

```
else if (strcmp(curr_node->value, "unset") == 0)
{
    node* nx = curr_node->next; // n1 would be the 'a'
    if (nx != 0 && nx->value[0] != 0)
    {
        unset_var(nx->value);
    }
}
```

[24]

Showenv Command

This command is used to print environment variables. The part to print specific environment variables was not attempted. This command can be found in the main().

```
if (strcmp(input_buf, "showenv") == 0)
{
    // if showvar <cmd> is true then print that var
    // else print all i.e. showvar ONLY
    print_env_vars();
}
```

[25]

```
void print_env_vars()
{
    char** s = environ;

    for (; *s; s++)
    {
        printf("%s\n", *s);
    }
}
```

[26]

Variables

The shell variables are declared via the use of the vars_on_start() function which also makes use of the add_or_change_vars() function as can be seen below:

```
void vars_on_start()
{
    add_or_change_vars("PATH", getenv("PATH"));
    add_or_change_vars("PROMPT", shellname);
    add_or_change_vars("CWD", getcwd(CWD, sizeof(CWD)));
    add_or_change_vars("USER", getenv("USER"));
    add_or_change_vars("HOME", getenv("HOME"));
    add_or_change_vars("SHELL", "TO/CHANGE");
    add_or_change_vars("TERMINAL", ttyname(STDIN_FILENO));
    add_or_change_vars("EXITCODE", EXITCODE);
}
```

[27]

The add_or_change_vars() function is used to add variables to the vars struct which was declared, [2].

```
int add_or_change_vars(char* shellVarName, char* shellVarContent)
{
    // first run tries to replace
    for (int i = 0; i < g_var_count; ++i)
    {
        if (g_vars[i].name == 0)
        {
            continue;   // empty slot - ignore it
        }
        if (strcmp(g_vars[i].name, shellVarName) == 0)
        {
            // replacement
            free(g_vars[i].value);
            g_vars[i].value = strdup(shellVarContent);
            return 1;
        }
    }
    // addition (as no replacement was done)
    for (int i = 0; i < g_var_count; ++i)
    {
        if (g_vars[i].name == 0)    // empty
        {
            g_vars[i].name = strdup(shellVarName);
            g_vars[i].value = strdup(shellVarContent);
            return 1;
        }
    }
    return 0;
}
```

[28]

In the “main.h” header file, one can find where some of the variables that we were asked to declare can be found, such as for example, the EXITCODE:

```
#define EXITCODE "0"
```

[29]

Testing

Input, Tokenisation, and Quoting

INPUT	OUTPUT
prompt> These are tokens	[0]: These [pipe?: 0] [1]: are [pipe?: 0] [2]: tokens [pipe?: 0]
prompt> "This is a single token"	[0]: This is a single token [pipe?: 0]
prompt> set Name = Nathan prompt> echo "My name is $Name"	My name is Nathan
prompt> echo \$Name	$Name

Variables

INPUT:

prompt> echo $PATH

OUTPUT:

/home/nathan/.vscode-server/bin/507ce72a4466fbb27b715c3722558bb15afa9f48/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/mnt/c/Program Files (x86)/Common Files/Oracle/Java/javapath:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/xampp/php:/mnt/c/ProgramData/ComposerSetup/bin:/mnt/c/Users/natha/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/natha/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/natha/AppData/Roaming/Composer/vendor/bin:/mnt/c/Users/natha/AppData/Local/atom/bin:/snap/bin

INPUT	OUTPUT
prompt> echo $PROMPT	smash 1.0
prompt> echo $CWD	/home/nathan/shell/build
prompt> echo $USER	nathan
prompt> echo $SHELL	TO/CHANGE
prompt> echo $TERMINAL	/dev/pts/0
prompt> echo $EXITCODE	0
prompt> unset USER prompt> set USER = NathanBZ prompt> echo $USER	 NathanBZ
prompt> set age = "Older than 18" prompt> echo $age	 Older than 18
prompt> set c = abc prompt> echo c prompt> echo $c	 c abc

Internal Commands

INPUT	OUTPUT
prompt> exit prompt> quit	Exiting... *Program terminates*
prompt> echo	
prompt> echo $NonExistentVar	NonExistentVar
prompt> echo $CWD prompt> cd ..	/home/nathan/shell/build new dir: /home/nathan/shell
prompt> cd /home prompt> echo $CWD	 /home

INPUT:

prompt> showvar

OUTPUT:

PATH=/home/nathan/.vscode-server/bin/507ce72a4466fbb27b715c3722558bb15afa9f48/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/mnt/c/Program Files (x86)/Common Files/Oracle/Java/javapath:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/xampp/php:/mnt/c/ProgramData/ComposerSetup/bin:/mnt/c/Users/natha/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/natha/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/natha/AppData/Roaming/Composer/vendor/bin:/mnt/c/Users/natha/AppData/Local/atom/bin:/snap/bin

PROMPT=smash 1.0

CWD=/home/nathan/shell/build

USER=nathan

HOME=/home/nathan

SHELL=TO/CHANGE

TERMINAL=/dev/pts/0

EXITCODE=0

Internal Commands Continued

INPUT	OUTPUT
prompt> showvar CWD	command is: showvar and value for the command is: /home/nathan/shell/build
prompt> set A = ABCD prompt> echo $A prompt> unset A prompt> echo $A	 ABCD A
prompt> unset D	*no changes to a non-existent variable*

INPUT:

prompt> showenv

OUTPUT:

SHELL=/bin/bash

COLORTERM=truecolor

TERM_PROGRAM_VERSION=1.57.1

WSL_DISTRO_NAME=Ubuntu-20.04

NAME=DESKTOP-1B7Q3R6

PWD=/home/nathan/shell/build

LOGNAME=nathan

VSCODE_GIT_ASKPASS_NODE=/home/nathan/.vscode-server/bin/507ce72a4466fbb27b715c3722558bb15afa9f48/node

HOME=/home/nathan

LANG=C.UTF-8

LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.w

mv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:

GIT_ASKPASS=/home/nathan/.vscode-server/bin/507ce72a4466fbb27b715c3722558bb15afa9f48/extensions/git/dist/askpass.sh

LESSCLOSE=/usr/bin/lesspipe %s %s

TERM=xterm-256color

LESSOPEN=| /usr/bin/lesspipe %s

USER=nathan

VSCODE_GIT_IPC_HANDLE=/tmp/vscode-git-13daf57b31.sock

SHLVL=1

WSLENV=VSCODE_WSL_EXT_LOCATION/up

VSCODE_GIT_ASKPASS_MAIN=/home/nathan/.vscode-server/bin/507ce72a4466fbb27b715c3722558bb15afa9f48/extensions/git/dist/askpass-main.js

XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop

PATH=/home/nathan/.vscode-server/bin/507ce72a4466fbb27b715c3722558bb15afa9f48/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/mnt/c/Program Files (x86)/Common Files/Oracle/Java/javapath:/mnt/c/Windows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0/:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/xampp/php:/mnt/c/ProgramData/ComposerSetup/bin:/mnt/c/Users/natha/AppData/Local/Microsoft/WindowsApps:/mnt/c/Users/natha/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/natha/AppData/Roaming/Composer/vendor/bin:/mnt/c/Users/natha/AppData/Local/atom/bin:/snap/bin

HOSTTYPE=x86_64

TERM_PROGRAM=vscode

VSCODE_IPC_HOOK_CLI=/tmp/vscode-ipc-96c69443-6c88-4d95-b38a-2558b86f2d99.sock

_=/home/nathan/shell/build/OS_Assignment

Final Thoughts

Although my implementation does not satisfy all the requirements that were given, the ones that were completed should be relatively correct and with few bugs. I will be inserting a video where I test out what I implemented, as required.

Please find the link of the video here:
https://drive.google.com/file/d/19e6mJwR8wBfwebxU-Im2NYSh-rmQYcUL/view?usp=sharing

FACULTY OF
INFORMATION AND
COMMUNICATION
TECHNOLOGY

Declaration

Plagiarism is defined as "the unacknowledged use, as one's own work, of work of another person, whether or not such work has been published" (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my work, except where acknowledged and referenced.

I understand that the penalties for making a false declaration may include, but are not limited to, loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected, and will be given zero marks.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Nathan Bonavia Zammit	
Student Name	Signature
Student Name	Signature

Student Name	Signature
Student Name	Signature
CPS1012	CPS1012 Assignment : smash
Course Code	Title of work submitted

06/07/2021

Date