

Practica 3: Construir un Mini Shell

David Morales Sáez

INTRODUCCIÓN	3
DISEÑO	3
MODIFICADORES	3
PRUEBAS	4
CONCLUSIÓN	8
ANEXO	9

Introducción

En esta práctica se ha propuesto crear una pequeña terminal donde poder lanzar instrucciones.

Diseño

Este programa puede ser segmentado en 3 distintas partes:

1. Obtenemos las opciones del shell (<, >, |, &)
2. Segmentamos la llamada en la instrucción y sus parámetros
3. Lanzamos la llamada del sistema

En primer lugar, buscamos el primer separador(<, >, |, &, \0 o \n) y, en el caso de existir, actuamos de forma consecuente con cada uno. Esto lo vamos haciendo hasta que no encontremos ningún separador en la ristra donde vamos guardando la llamada actual. Una vez conseguido esto, obtenemos el primer espacio, en caso que haya, y obtendremos donde termina el comando a ejecutar.

Tras obtener el comando, vamos obteniendo los distintos parámetros que acompañarán al comando en la llamada y guardándolos en un vector. Finalmente, se creará un hijo y éste lanzará el comando con los parámetros obtenidos.

Modificadores

En función de qué opciones utilicemos, el comportamiento del programa será muy distinto. Para esta práctica se han añadido cuatro modificadores incluidos en la terminal de Linux:

- Pipe

Si nuestro programa se encuentra con el símbolo | , considerará que deseamos crear un pipe entre dos segmentos de la llamada. Para ello, se creará un hijo y tanto el padre como el hijo trabajarán de forma independiente, excepto porque la salida del padre será la entrada del hijo.

- Redireccionamiento de Entrada

Cuando nos encontremos con el símbolo < , nos estarán indicando que todas las entradas de este comando no serán por teclado, sino por el fichero que se le suministrará por la derecha.

- Redireccionamiento de Salida

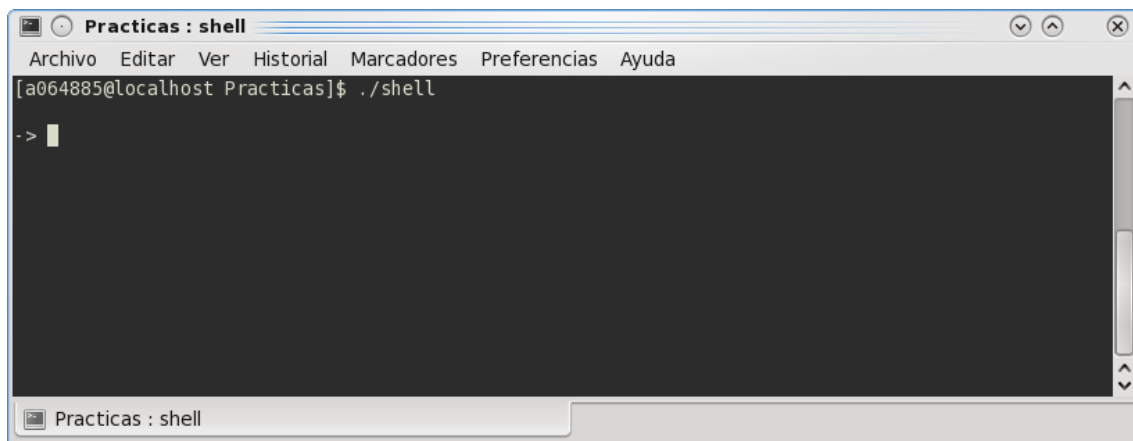
Cuando nos encontremos con el símbolo > , nos estarán indicando que todas las salidas de este comando no serán por pantalla, sino por el fichero que se le suministrará por la derecha.

- Modo tanda

Finalmente, si nos encontramos con el símbolo & , se considerará que se desea llevar a cabo esta orden en modo tanda, es decir, que se irá ejecutando concurrentemente mientras seguimos trabajando. Para ello crearemos un hijo y no esperaremos a que termine, sino que seguiremos ejecutándonos.

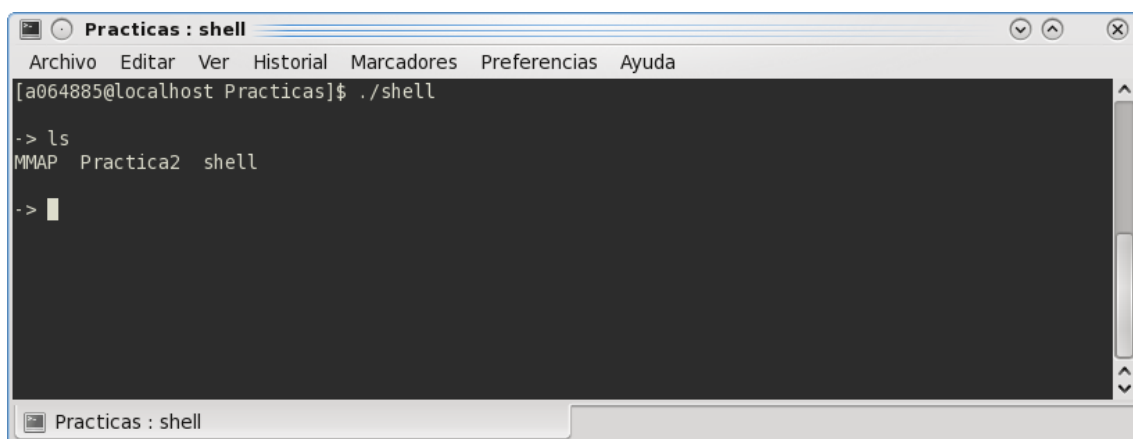
Pruebas

Para comprobar el funcionamiento de este programa, hemos creado una batería de pruebas. En primer lugar, ejecutamos el shell:



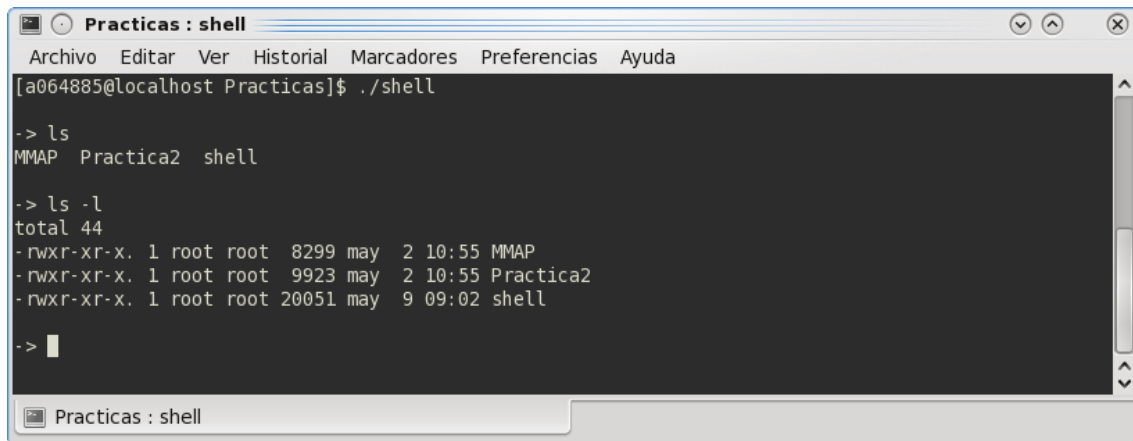
```
Practicas : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda
[a064885@localhost Practicas]$ ./shell
-> 
```

tras lo cual comprobamos lo que tenemos en el lugar donde trabajamos:



```
Practicas : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda
[a064885@localhost Practicas]$ ./shell
-> ls
MMAP Practica2 shell
-> 
```

aunque más información de los ficheros:



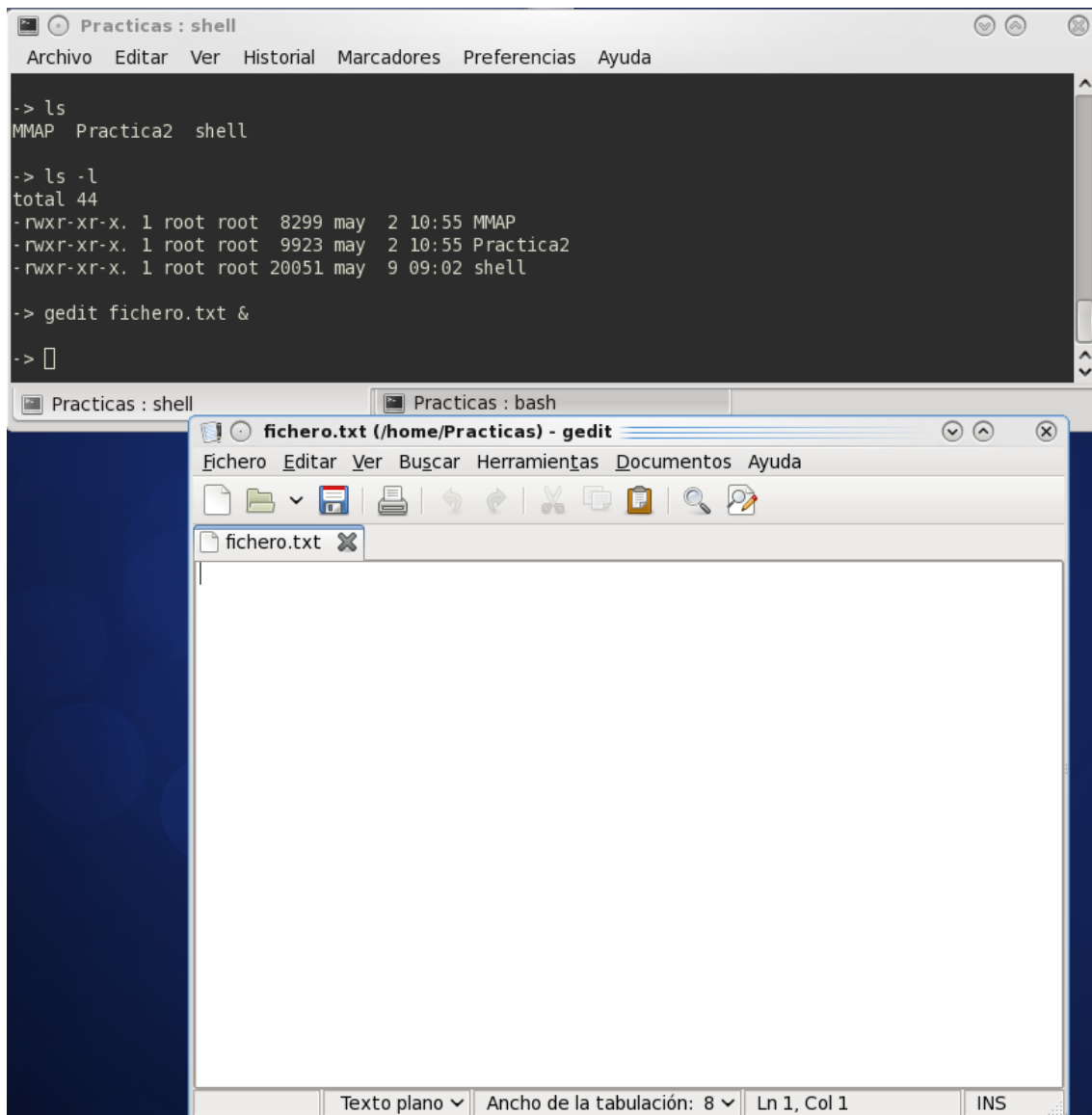
A terminal window titled "Practicas : shell" with a menu bar (Archivo, Editar, Ver, Historial, Marcadores, Preferencias, Ayuda). The prompt is [a064885@localhost Practicas]\$ and the command ./shell has been executed. The user then runs 'ls' and 'ls -l', displaying the following output:

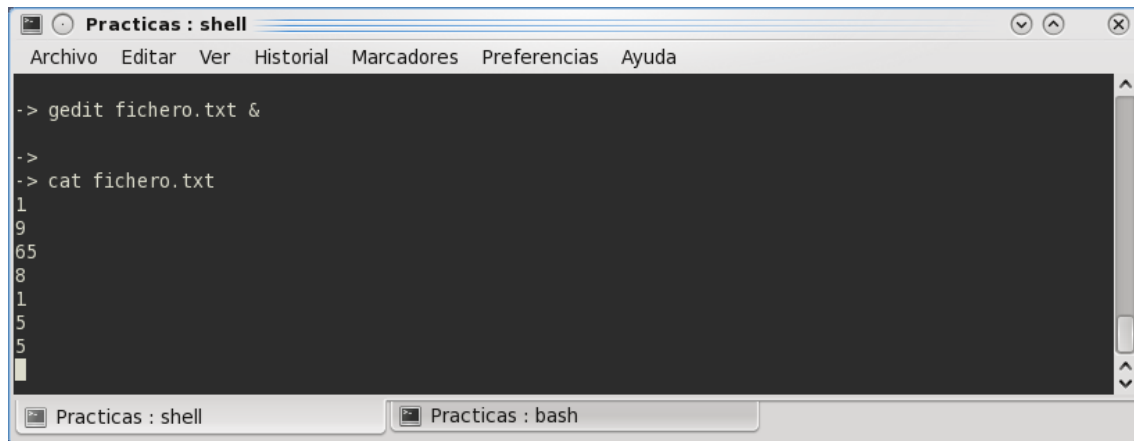
```
[a064885@localhost Practicas]$ ./shell
-> ls
MMAP Practica2 shell

-> ls -l
total 44
-rwxr-xr-x. 1 root root 8299 may  2 10:55 MMAP
-rwxr-xr-x. 1 root root 9923 may  2 10:55 Practica2
-rwxr-xr-x. 1 root root 20051 may  9 09:02 shell

-> 
```

Ahora, crearemos un fichero llamando a un editor en modo tando y lo modificamos:



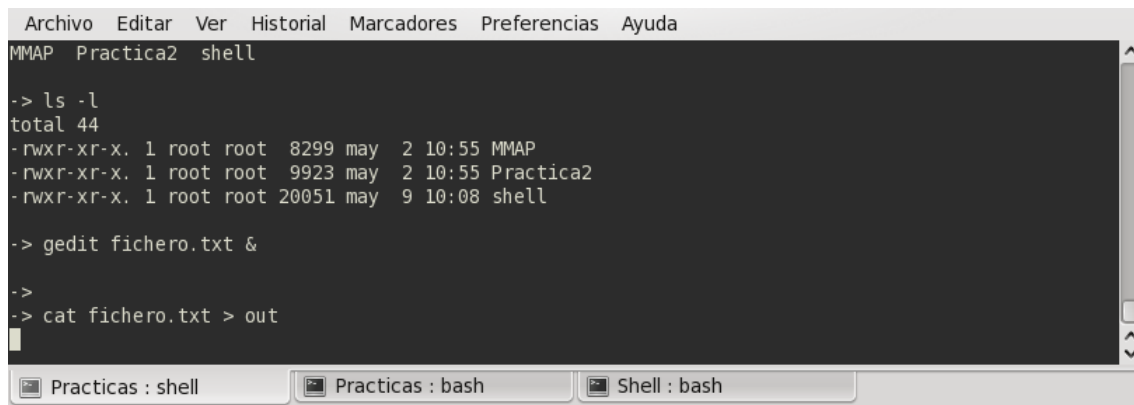


```
Practicas : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda

-> gedit fichero.txt &
->
-> cat fichero.txt
1
9
65
8
1
5
5

```

Una vez creado, vamos a ver lo que tiene, pero redireccionándolo a un fichero:



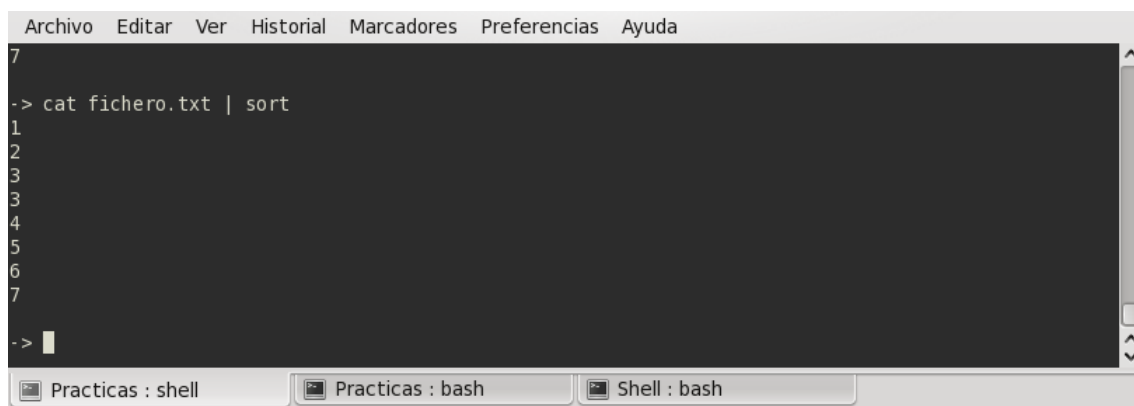
```
Practicas : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda
MMAP Practica2 shell

-> ls -l
total 44
-rwxr-xr-x. 1 root root 8299 may  2 10:55 MMAP
-rwxr-xr-x. 1 root root 9923 may  2 10:55 Practica2
-rwxr-xr-x. 1 root root 20051 may  9 10:08 shell

-> gedit fichero.txt &
->
-> cat fichero.txt > out

```

Para probar un poco, vamos a ordenar el fichero y mostrarlo por pantalla:

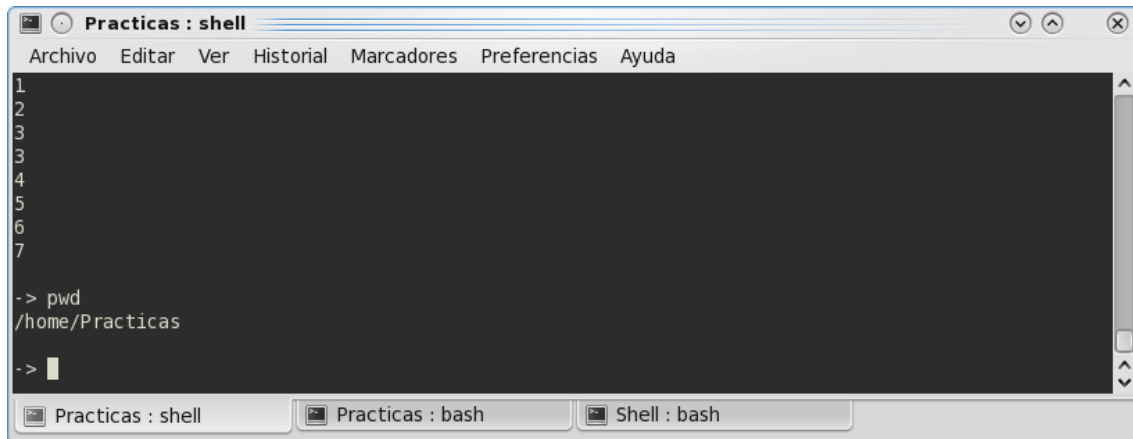


```
Practicas : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda

7
-> cat fichero.txt | sort
1
2
3
3
4
5
6
7
->

```

También podemos movernos por los directorios. Primero, veamos dónde estamos:



```
Practicass : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda

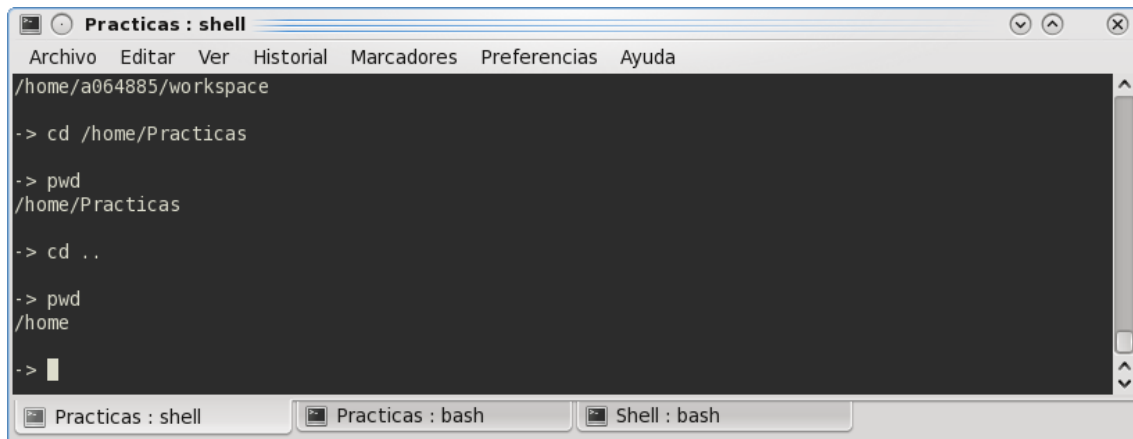
1
2
3
3
4
5
6
7

-> pwd
/home/Practicass

-> 
```

The terminal window shows the current directory as /home/Practicass. The window title is 'Practicass : shell' and it has a menu bar with 'Archivo', 'Editar', 'Ver', 'Historial', 'Marcadores', 'Preferencias', and 'Ayuda'. The bottom of the window shows three tabs: 'Practicass : shell', 'Practicass : bash', and 'Shell : bash'.

y ahora vayamos a la carpeta padre:



```
Practicass : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda

/home/a064885/workspace

-> cd /home/Practicass

-> pwd
/home/Practicass

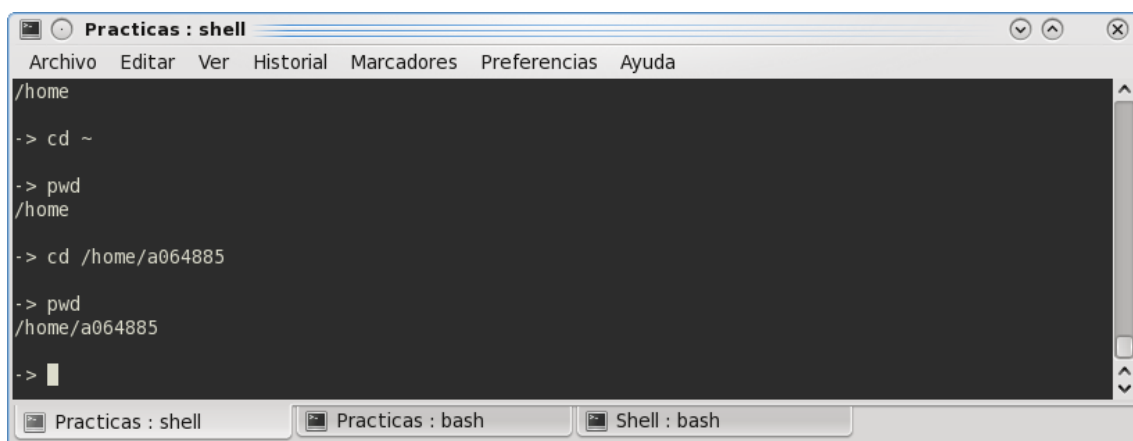
-> cd ..

-> pwd
/home

-> 
```

The terminal window shows the user navigating from /home/a064885/workspace to /home/Practicass and then to /home. The window title is 'Practicass : shell' and it has a menu bar with 'Archivo', 'Editar', 'Ver', 'Historial', 'Marcadores', 'Preferencias', and 'Ayuda'. The bottom of the window shows three tabs: 'Practicass : shell', 'Practicass : bash', and 'Shell : bash'.

aunque también podemos ir a la carpeta personal:



```
Practicass : shell
Archivo  Editar  Ver  Historial  Marcadores  Preferencias  Ayuda

/home

-> cd ~

-> pwd
/home

-> cd /home/a064885

-> pwd
/home/a064885

-> 
```

The terminal window shows the user navigating from /home to /home/a064885. The window title is 'Practicass : shell' and it has a menu bar with 'Archivo', 'Editar', 'Ver', 'Historial', 'Marcadores', 'Preferencias', and 'Ayuda'. The bottom of the window shows three tabs: 'Practicass : shell', 'Practicass : bash', and 'Shell : bash'.

Conclusión

Como se ha comprobado en el transcurso de esta práctica, el trabajo que lleva el terminal es un tanto complejo, ya que no solo ha de lanzar las instrucciones, sino que también ha de definir y variar las distintas formas de ejecución.

Anexo

```
#include <stdio.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>
#include <iostream>
#include <fcntl.h>
using namespace std;

int pos_separador(char* ristra)
{
    int i=0;
    int n=strlen(ristra);
    while((ristra[i]!='\0')&&(ristra[i]!='\n')&&(i<n))
    {
        if((ristra[i]=='|')||(ristra[i]=='&')||(ristra[i]=='>')||(ristra[i]=='<'))
            break;
        i++;
    }
    if(i==n)
        return -1;
    return i;
}

int pos_pipe(char* ristra)
{
    int i=0;
    int n=strlen(ristra);
    while((ristra[i]!='\0')&&(ristra[i]!='\n')&&(i<n))
    {
        if(ristra[i]=='|')
            break;
        i++;
    }
    if(i==n)
        return -1;
    return i;
}

int pos_redirIN(char* ristra)
{
    int i=0;
    int n=strlen(ristra);
    while((ristra[i]!='\0')&&(ristra[i]!='\n')&&(i<n))
    {
        if(ristra[i]=='<')
            break;
        i++;
    }
}
```

```

    }
    if(i==n)
        return -1;
    return i;
}

int pos_redirOUT(char* ristra)
{
    int i=0;
    int n=strlen(ristra);
    while((ristra[i]!='\0')&&(ristra[i]!='\n')&&(i<n))
    {
        if(ristra[i]=='>')
            break;
        i++;
    }
    if(i==n)
        return -1;
    return i;
}

int pos_tanda(char* ristra)
{
    int i=0;
    int n=strlen(ristra);
    while((ristra[i]!='\0')&&(ristra[i]!='\n')&&(i<n))
    {
        if(ristra[i]=='&')
            break;
        i++;
    }
    if(i==n)
        return -1;
    return i;
}

int pos_espacio(char* ristra)
{
    int i=0;
    int n=strlen(ristra);
    while((ristra[i]!='\0')&&(ristra[i]!='\n')&&(i<n))
    {
        if(ristra[i]==' ')
            break;
        i++;
    }
    if(i==n)
        return -1;
    return i;
}

int main ()
{
    string ruta_actual = "/home/a064885/workspace/Shell";

```

```

while(1)
{
    char ristra[255];
    for(int i=0; i<255; i++)
        ristra[i]='\0';
    int estado=0;
    int pid=getpid();
    printf("\n-> ");
    fflush(stdin);
    cin.getline(ristra, 255);
    int fd1, fd0;
    bool reset0=false, reset1=false, hijo=false;
    while(pos_separador(ristra)!=-1)
    {
        int k = pos_pipe(ristra);
        if (k!=-1)
        {
            int fd[2];
            pipe(&fd[0]);
            if (fork() != 0)
            {
                // IZQUIERDA
                pid = getpid();
                if(ristra[k-1]==' ')
                    k--;
                char aux[k];
                for(int i=0; i<k; i++)
                    aux[i] = '\0';
                for(int i=0; i<k; i++)
                    aux[i] = ristra[i];
                for(int i=0; i<255; i++)
                    ristra[i] = '\0';
                for(int i=0; i<k; i++)
                    ristra[i] = aux[i];
                close(fd[0]);
                fd1 = dup(1);
                close(1);
                dup(fd[1]);
                close(fd[1]);
                reset1 = true;
            }
            else
            {
                // DERECHA
                pid = getpid();
                int m=strlen(ristra);
                k++;
                if(ristra[k]==' ')
                    k++;
                char aux[m-k];
                for(int i=0; i<m-k; i++)
                    aux[i] = '\0';
                for(int i=0; i<m-k; i++)
                    aux[i] = ristra[k+i];
                for(int i=0; i<255; i++)
                    ristra[i] = '\0';
            }
        }
    }
}

```

```

        for(int i=0; i<m-k; i++)
            ristra[i] = aux[i];
        close(fd[1]);
        fd1 = dup(0);
        close(0);
        dup(fd[0]);
        close(fd[0]);
        hijo = true;
        reset0 = true;
    }
    continue;
}
k = pos_redirOUT(ristra);
if (k!=-1)
{
    int m=strlen(ristra);
    k++;
    if(ristra[k]==' ')
        k++;
    // Obtenemos la ristra de la derecha
    char derecha[m-k];
    for(int i=0; i<m-k; i++)
        derecha[i] = '\0';
    int x = 0;
    while(k<m)
    {
        derecha[x] = ristra[k];
        k++;
        x++;
    }
    derecha[x] = '\0';
    // Obtenemos la ristra de la izquierda
    k = pos_redirOUT(ristra);
    if(ristra[k-1]==' ')
        k--;
    char izquierda[k];
    for(int i=0; i<k; i++)
        izquierda[i] = '\0';
    for(int i=0; i<k; i++)
        izquierda[i] = ristra[i];
    izquierda[k] = '\0';
    fd1 = dup(1);
    close(1);
    fopen(derecha, "w");
    reset1=true;
    for (int s=0; s<m; s++)
    {
        ristra[s]='\0';
    }
    for(int i=0; i<k; i++)
    {
        ristra[i] = izquierda[i];
    }
    continue;
}

```

```

}
k = pos_redirIN(ristra);
if (k!=-1)
{
    int m=strlen(ristra);
    int x = 0;
    k++;
    if(ristra[k]==' ')
        k++;
    // Obtenemos la ristra de la derecha
    char derecha[m-k];
    for(int i=0; i<m-k; i++)
        derecha[i] = '\0';
    while(k<m)
    {
        derecha[x] = ristra[k];
        k++;
        x++;
    }
    derecha[x] = '\0';
    // Obtenemos la ristra de la izquierda
    k = pos_redirIN(ristra);
    if(ristra[k-1]==' ')
        k--;
    char izquierda[k];
    for(int i=0; i<k; i++)
        izquierda[i] = '\0';
    for(int i=0; i<k; i++)
        izquierda[i] = ristra[i];
    izquierda[k] = '\0';
    fd0 = dup(0);
    close(0);
    fopen(derecha, "w");
    reset0=true;
    for (int s=0; s<m; s++)
    {
        ristra[s]='\0';
    }
    for(int i=0; i<k; i++)
    {
        ristra[i] = izquierda[i];
    }
    continue;
}
k = pos_tanda(ristra);
if (k!=-1)
{
    if(ristra[k-1]==' ')
        k--;
    int m=strlen(ristra);
    char aux[k];
    for(int i=0; i<k; i++)
    {
        aux[i] = ristra[i];
    }
}

```

```

    }
    for (int s=0; s<m; s++)
    {
        ristra[s]='\0';
    }
    for(int i=0; i<k; i++)
    {
        ristra[i] = aux[i];
    }
    if(fork()!=0)
    {
        waitpid(0, NULL, WNOHANG);
        estado=-1;
        break;
    }
}
}
if(estado==-1)
    continue;
char parametros[20][20];
int n = pos_espacio(ristra), nparam=1;
char comando[n];
for(int k=0; k<n; k++)
    comando[k]='\0';
if(n!=-1)
{
    strncpy(comando, ristra, n);
    comando[n]='\0';
    int m=strlen(ristra), i=n+1;
    char aux[m-i];
    for(int k=0; k<m-1; k++)
        aux[k]='\0';
    while(i<m)
    {
        aux[i-n-1] = ristra[i];
        i++;
    }
    for (int s=0; s<m; s++) {
        ristra[s]='\0';
    }
    int k;
    for(k=0; k<strlen(aux); k++)
    {
        ristra[k] = aux[k];
    }
    n=strlen(aux);
    int init=0;
    while(pos_espacio(ristra)!=-1)
    {
        int espacio = pos_espacio(ristra);
        for(k=init; k<espacio; k++)
        {
            parametros[nparam][k] = ristra[k];
        }

```

```

        parametros[nparam][k]='\0';
        char aux2[n-espacio-1];
        for(int s=espacio+1; s<n; s++)
        {
            aux2[s-espacio-1]=ristra[s];
        }
        for (int s=0; s<n; s++)
        {
            ristra[s]='\0';
        }
        for(k=0; k<n-espacio-1; k++)
        {
            ristra[k] = aux2[k];
        }
        n=strlen(aux2);
        nparam++;
    }
    for(k=0; k<n; k++)
    {
        parametros[nparam][k] = ristra[k];
    }
    for(i=k; i<20; i++)
    {
        parametros[nparam][i] = '\0';
    }
    nparam++;
    *parametros[nparam]=(char)NULL;
}
else
{
    int k;
    strcpy(comando, ristra);
}
// Caso en el que la orden sea un cd (ir a...)
if(strcmp(comando, "cd")==0)
{
    string objetivo;
    if(parametros[1][0]=='/')
    {
        string aux(parametros[1]);
        objetivo = aux;
    }
    else
    {
        // Si vamos a un lugar desde el padre

        if((parametros[1][0]=='.')&&((parametros[1][1]=='.')))
        {
            int indice =
ruta_actual.find_last_of('/');
            objetivo = ruta_actual.substr(0,indice);
        }
        // Si vamos a un lugar desde la posición actual
    }
    else

```

```

        {
            string aux(parametros[1]);
            objetivo = ruta_actual+'/'+aux;
        }
    }
    chdir(objetivo.c_str());
    ruta_actual = objetivo;
    continue;
}
// Caso estándar
int num = fork();
if(num!=0)
{
    waitpid(0, NULL, 0);
    // Devolvemos al estado inicial la salida
    if(reset1)
    {
        close(1);
        dup(fd1);
        close(fd1);
        reset1=false;
    }
    // Devolvemos al estado inicial la entrada
    if(reset0)
    {
        close(0);
        dup(fd0);
        close(fd0);
        reset0 = false;
    }
    // En caso de ser un hijo del pipe, salimos
    if(hijo)
    {
        return 0;
    }
    else
    {
        waitpid(0, NULL, 0);
        continue;
    }
}
else
{
    for(int i=0; i<strlen(comando); i++)
    {
        parametros[0][i] = comando[i];
    }
    parametros[0][strlen(comando)]='\0';
    char* args[nparam+1];
    for(int i=0; i<nparam; i++)
    {
        args[i]=parametros[i];
    }
    args[nparam]=NULL;

```



```
        execvp(comando, args);
    }
}
```