



**UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN**

**FACULTAD DE INGENIERÍA**

**ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS**

# **SpaceShell: Desarrollo de una Shell básica para Sistemas Operativos Unix**

## **Evaluación de Producto**

**Docente:** MSc. Hugo Manuel Barraza Vizcarra  
**Estudiantes:** William Yak Vargas Chambilla  
Eduardo Xavier Paca Aquino  
**Códigos:** 2023-119066  
2023-119049  
**Curso:** Sistemas Operativos  
**Ciclo:** VI  
**Turno:** A

**Tacna - Perú**  
**Octubre 2025**

# ÍNDICE

1. Objetivos y Alcance .....	1
1.1. Requisitos Funcionales .....	1
1.2. Requisitos No Funcionales .....	2
2. Arquitectura y Diseño .....	3
2.1. Flujo de Trabajo (I/O) .....	3
2.2. Objetos, funciones y relaciones .....	3
3. Detalles de Implementación .....	4
3.1. APIs POSIX Usadas .....	4
3.2. Decisiones clave .....	5
4. Concurrencia y Sincronización .....	6
5. Pruebas y Resultados .....	6
5.1. Prompt Personalizado .....	6
5.2. Resolución de Rutas .....	6
5.2.1. Rutas Absolutas .....	7
5.2.2. Rutas Relativas (/bin) .....	7
5.2.3. Manejo de Errores en Rutas .....	7
5.3. Ejecución mediante Procesos .....	7
5.4. Manejo de Errores .....	7
5.5. Redirección de Salida Estándar (>) .....	8
5.6. Comando de Salida .....	8
5.7. Pipes (Tuberías) .....	8
5.8. Tareas en Segundo Plano .....	9
5.9. Redirección de Entrada y Anexar .....	9
5.10. Comandos Internos (Built-ins) .....	9
5.11. Concurrencia con Hilos .....	10
5.12. Manejo de Señales .....	10
6. Conclusiones y Trabajos Futuros .....	10
7. Anexos .....	11
7.1. Script de Instalación (install.sh) .....	11
7.2. Comandos Externos Básicos .....	11

# 1. Objetivos y Alcance

Nuestro objetivo con este proyecto es desarrollar e implementar SpaceShell, un intérprete de comandos con funcionalidades básicas que sirva como evidencia de los conocimientos adquiridos sobre Sistemas Operativos.

Un intérprete de comandos, comúnmente llamado SHELL, es la interfaz que permite al usuario comunicarse con el Kernel del Sistema Operativo. En este sentido, el intérprete invoca al Kernel, más no lo reemplaza. En el caso de Unix, por ejemplo, el intérprete más conocido es BASH. Los intérpretes de comandos son usualmente presentados en CLI (Interfaz de Línea de Comandos), fácilmente identificables por ser interfaces textuales en una terminal, sin elementos visuales estructurados.

Para ser funcional, el intérprete SpaceShell debe contener las siguientes características básicas

## 1.1. Requisitos Funcionales

### 1. Prompt personalizado

- Mostrar un prompt propio y leer la línea de comando.

### 2. Resolución de rutas

- Si el usuario escribe una ruta absoluta debe ejecutarse tal cual.
- Si no es una ruta absoluta, asumir que se llama desde **/bin**
- Manejar errores cuando el ejecutable no exista o no tenga permisos.

### 3. Ejecución mediante procesos

- La invocación debe realizarse con **fork()** y luego **exec()**, desde el proceso hijo.
- El proceso padre (el intérprete) espera la finalización del hijo con **wait()** / **waitpid()** antes de aceptar el siguiente comando.

### 4. Manejo de Errores

- Mensajes claros cuando el comando no exista, la ruta no sea válida o **exec** falle (incluir error/perror cuando sea útil).

### 5. Redirección de salida estándar (>)

Si el usuario ingresa: `nombrePrograma arg2 > archivo`

- Debe redirigir **stdout** del proceso hijo a archivo (crearlo/truncarlo). Nada debe mostrarse en pantalla para esa ejecución.
- Nota: Todos los tokens (incluido **>**) están separados por espacios.

### 6. Comando de salida

- El intérprete termina al ingresar la palabra **salir** o presionar **Ctrl + D**.

### 7. Pipes

- Detectar y ejecutar pipes simples (**|**): `cmd1 | cmd2`.

### 8. Tareas en segundo plano

- Invocando con ampersand (&), el comando no debe bloquear el prompt, mostrar **waitpid no bloqueante** o recolección diferida.

## 9. Redireccionamiento

- Integrar redirecciones de entrada (<) y salida (>).

## 10. Comandos Internos Built-ins

- Comandos básicos como cd, pwd, help, history o alias.

## 11. Concurrencia con hilos

- Built-in PARALLEL que ejecute n comandos en paralelo con pthread\_create, con sincronización.

## 12. Gestión de memoria instrumentada

- Built-in meminfo que muestre uso aproximado de heap o estadístico de asignaciones/liberaciones, usando malloc/free.

## 13. Manejo de señales

- Ignorar o capturar SIGINT en el padre (sin matar la SHELL accidentalmente); pasar señales a hijos cuando corresponda.

# 1.2. Requisitos No Funcionales

## 1. POSIX

- Usar llamadas de sistema como fork, execvp/execve, wait/waitpid, pipe, dup2, open/close, chdir, getcwd, sigaction, etc.

## 2. Estructura modular

- Estructurar archivos(headers .h/.hpp, fuente .c/.cpp) y comentarios útiles.

## 3. Mensajes y errores

- Mensajes en stderr.

## 4. Manejo de memoria

- Evitar leaks y dangling pointers.

## 5. Control de versiones

- Git: Commits atómicos y con mensajes significativos.

## 6. Portabilidad

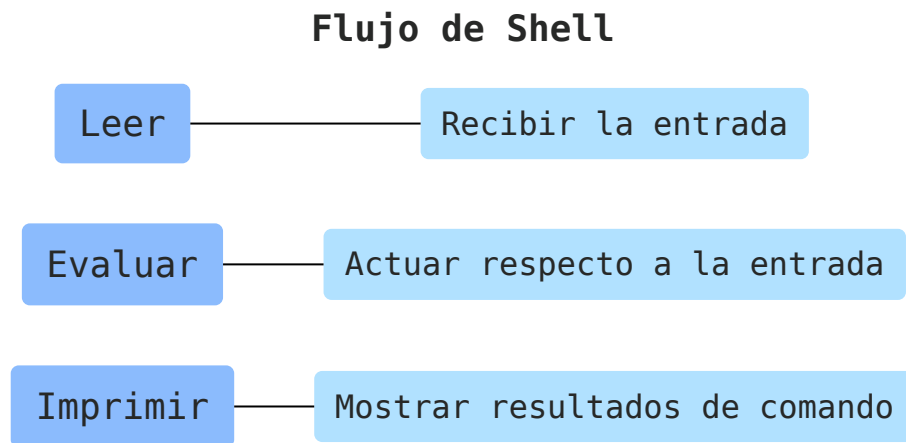
- Probar el funcionamiento del intérprete en Ubuntu y Debian.

## 2. Arquitectura y Diseño

SpaceShell requiere un diseño modular, altamente escalable, con responsabilidades únicas y con acceso a funciones de bajo nivel para administrar procesos, memoria, etc. Por ello, para su implementación se utilizará el potente lenguaje C++, lenguaje integrado con librerías relacionadas al Kernel.

### 2.1. Flujo de Trabajo (I/O)

El funcionamiento clásico de una Shell se basa en REPL (Read-Eval-Print Loop), un bucle que actúa para que la SHELL trabaje con sus entradas y salidas. Con ello, podemos describir el comportamiento de SpaceShell como

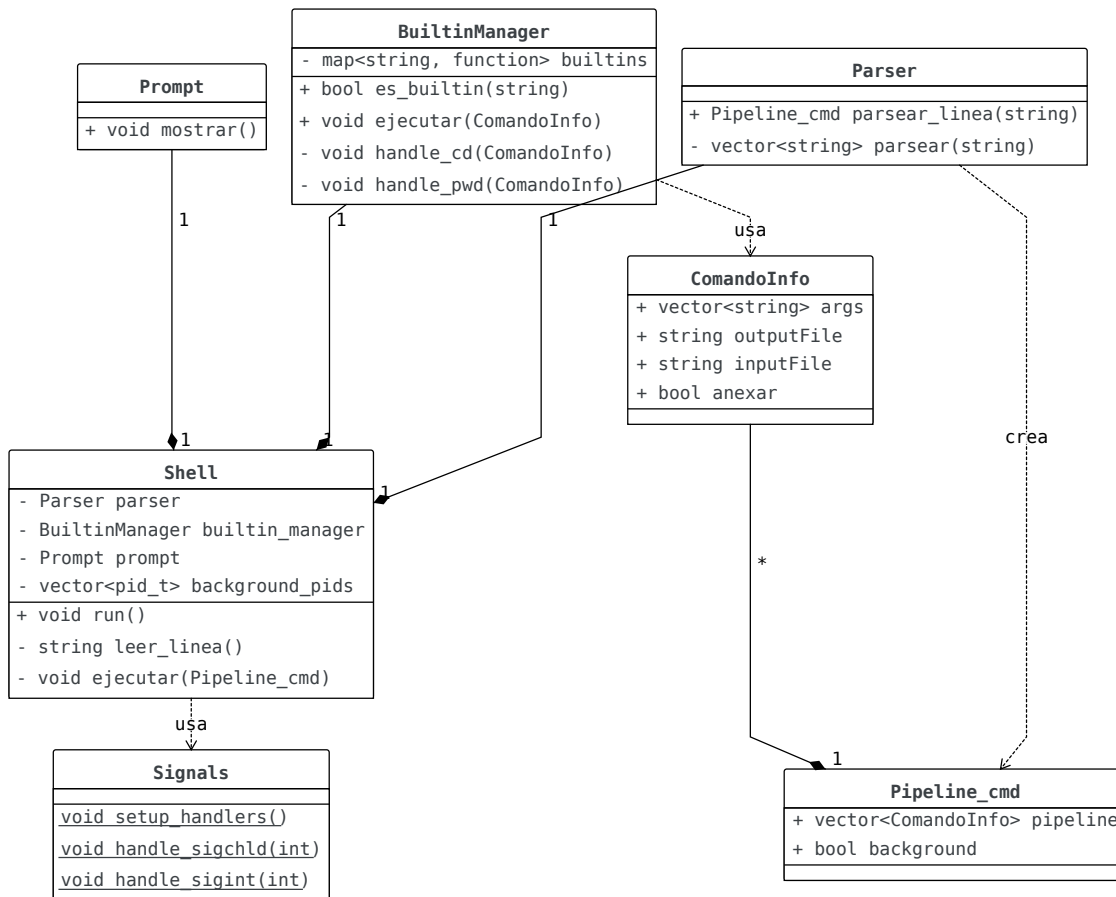


Para lograr este comportamiento, separando los actores en módulos, nos apoyaremos de los objetos de C++ para definir estructuras que interactúen en base a REPL.

### 2.2. Objetos, funciones y relaciones

Los objetos utilizados para la SHELL, así como sus responsabilidades son:

- **Parser:** Clase responsable de analizar la entrada de texto. Convierte una cadena de texto sin procesar en una estructura `Pipeline_cmd` que la Shell puede entender y ejecutar.
- **Pipeline\_cmd:** Estructura que representa una línea de comandos. Contiene una secuencia de `ComandoInfo`, además de si debe ejecutarse en segundo plano.
- **ComandoInfo:** Estructura que almacena la información de un único comando simple.
- **BuiltinManager:** Clase que gestiona el registro y la ejecución de comandos internos (built-ins).
- **Prompt:** Responsable de generar y mostrar el Prompt personalizado de la SHELL.
- **Signals:** Clase de utilidad usada para configurar handles de señales de la SHELL.



Listado 2: Diagrama de clases de uso de SpaceShell

- **Shell:** Clase que representa al intérprete de comandos. Encapsula el bucle principal que contiene al flujo de trabajo principal de la SHELL(REPL). Gestiona los módulos Parser, BuiltinManager.

### 3. Detalles de Implementación

#### 3.1. APIs POSIX Usadas

Nombre de API	Propósito	Librería
fork()	Crear proceso hijo para cada comando en Pipeline	unistd.h
execv()	Ejecutar comando externo	unistd.h
waitpid()	Esperar hijo en foreground	sys/wait.h

Nombre de API	Propósito	Librería
pipe()	Crear tubería entre comandos	unistd.h
dup2()	Redirigir stdout/stdin	unistd.h
open()	Abrir archivo para redirección	fcntl.h
close()	Cerrar archivos	unistd.h
chdir()	Cambiar directorio	unistd.h
getcwd()	Obtener directorio actual	unistd.h
read()	Leer entrada	unistd.h
pthread_create()	Crear hilo de ejecución	pthread.h
pthread_join()	Esperar hilo	pthread.h
sigaction()	Configurar SIGINT/SIGCHLD	signal.h
sigemptyset()	Limpiar máscara de señales	signal.h
signal()	Restaurar SIGINT	signal.h
access()	Verificar directorio actual	unistd.h
getenv()	Obtener variable de entorno	cstdlib

### 3.2. Decisiones clave

Situación	Decisión	Justificación
Resolver y ejecutar comandos externos	Utilizar <code>execv()</code> para direcciones absolutas y agregación a <code>/bin/</code> si no son absolutos	La ruta no depende de la configuración del SO, a diferencia de si se usaran variables de entorno
Implementar pipes simples	Utilizar 2 forks por pipe, apoyados de las APIs <code>pipe()</code> y <code>dup2()</code>	Probar la funcionalidad, en un contexto sencillo, eficiente y escalable

Situación	Decisión	Justificación
Manejar la espera de procesos hijos	Utilizar waitpid(); mientras que, para background, registraremos los PIDs	Conseguir un comportamiento intuitivo de subprocesos como en bash
Comandos internos a implementar	Incluir cd, pwd y parallel	Probar la funcionalidad con pocas opciones que se pueden escalar

## 4. Concurrencia y Sincronización

En SpaceShell, implementamos las concurrencia y sincronización para ejecutar varios comandos, para ello utilizamos hilos de ejecución. Con ello mejoramos la eficiencia de los comandos que se ejecutan.

Entre funcionalidades relacionadas a éste tópico están:

- Utilizando el comando PARALLEL se puede paralelizar procesos, dando paso a ejecución de los comandos que se le pasan como argumentos.
- Soporta Tareas en Segundo Plano utilizando ampersand(&) al final del comando, así se puede ejecutar concurrentemente sin bloquear al SpaceShell.
- Evita la interrupción de procesos por señales, siendo el SpaceShell quien recepciona las señales.

En general, para evitar las condiciones de carrera es que SpaceShell controla la multitarea con hilos de ejecución. Por tanto, esta construido con APIs POSIX diseñadas para gestionar los hilos.

## 5. Pruebas y Resultados

### 5.1. Prompt Personalizado

El prompt de SpaceShell muestra el directorio actual seguido del símbolo @, proporcionando contexto visual al usuario sobre su ubicación en el sistema de archivos.

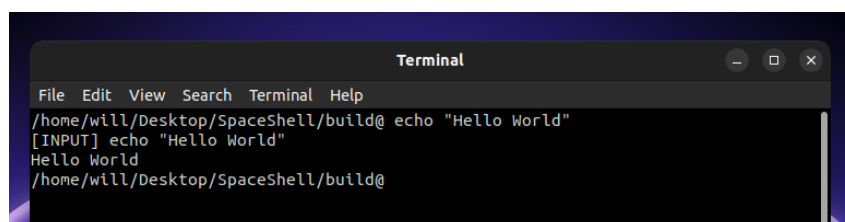


Figura 1: Prompt personalizado mostrando el directorio actual

### 5.2. Resolución de Rutas



### 5.2.1. Rutas Absolutas

SpaceShell detecta rutas que comienzan con / y las ejecuta directamente sin modificación.

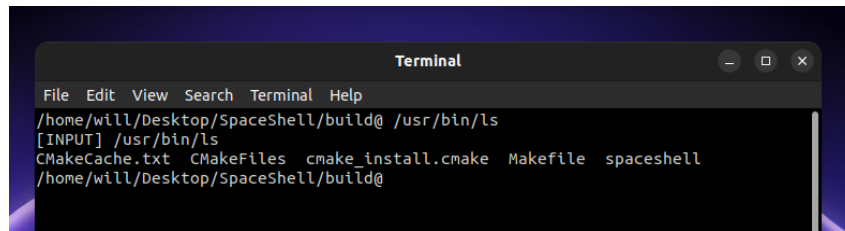


Figura 2: Ejecución de comando con ruta absoluta

### 5.2.2. Rutas Relativas (/bin)

Para comandos sin ruta explícita, SpaceShell antepone /bin/ automáticamente.

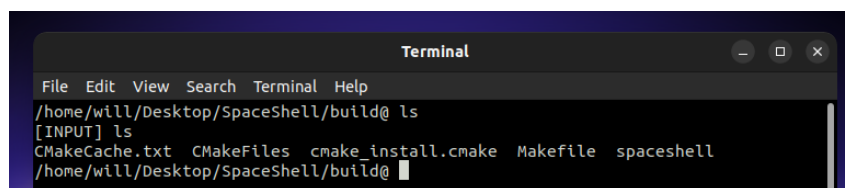


Figura 3: Ejecución de comando asumiendo directorio /bin

### 5.2.3. Manejo de Errores en Rutas

SpaceShell proporciona mensajes claros cuando un comando no puede ejecutarse, indicando la naturaleza del error.

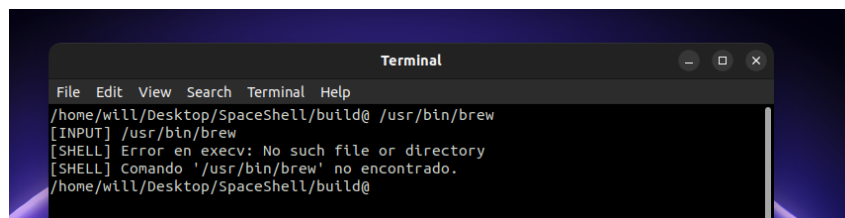


Figura 4: Mensaje de error para comando no encontrado

## 5.3. Ejecución mediante Procesos

Cada comando se ejecuta en un proceso hijo separado, mientras el proceso padre (la shell) espera su finalización antes de mostrar el siguiente prompt.

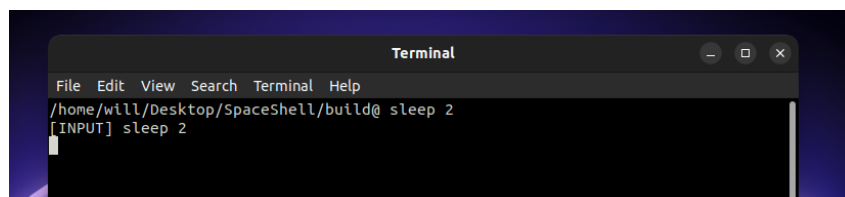


Figura 5: Ejecución secuencial de comandos mediante fork/exec

## 5.4. Manejo de Errores

SpaceShell utiliza `perror()` y `errno` para proporcionar diagnósticos precisos de errores del sistema.

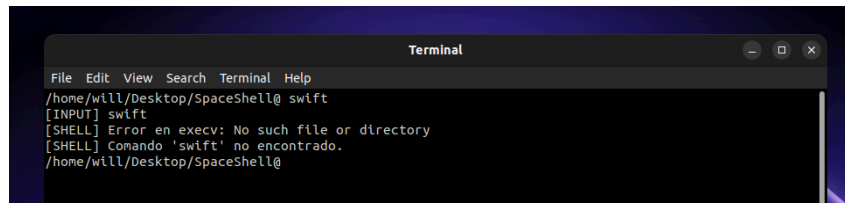


Figura 6: Mensajes de error detallados con información del sistema

## 5.5. Redirección de Salida Estándar (>)

El operador `>` redirige la salida estándar a un archivo, truncándolo si existe o creándolo si no existe.

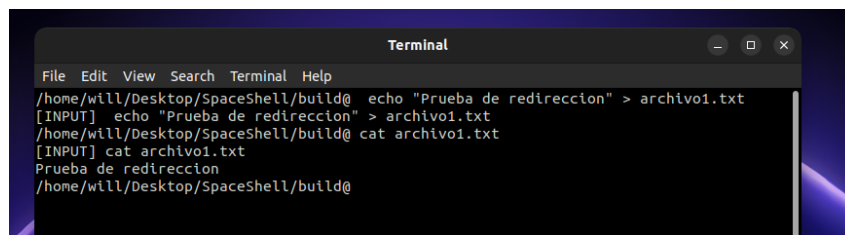


Figura 7: Redirección de salida a archivo con operador `>`

## 5.6. Comando de Salida

SpaceShell proporciona dos formas de terminar la sesión: el comando explícito `salir` o la señal EOF (End of File) mediante `Ctrl+D`.

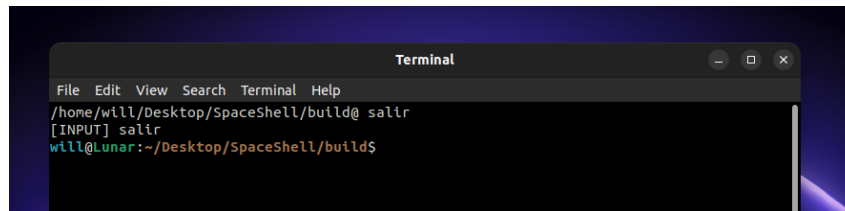
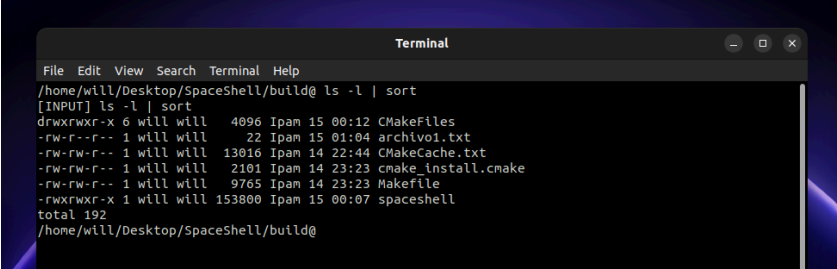


Figura 8: Salida de SpaceShell mediante comando `salir`

## 5.7. Pipes (Tuberías)

Los pipes como `cmd1 | cmd2` permiten conectar la salida estándar de un comando con la entrada estándar del siguiente, creando tuberías de procesamiento de datos.

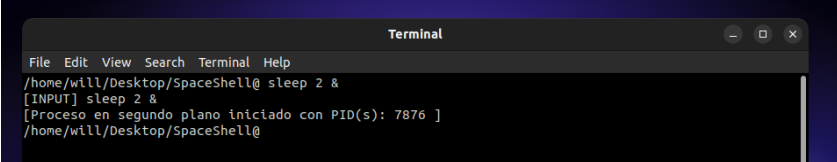
A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "/home/will/Desktop/SpaceShell/build@". The user enters "ls -l | sort". The output shows a list of files with permissions, owner, group, size, and timestamps, sorted by size. The files are: CMakeFiles (4096 bytes), archivo1.txt (22 bytes), CMakeCache.txt (13016 bytes), cmake\_install.cmake (2101 bytes), Makefile (9765 bytes), and spaceshell (153800 bytes). The total size is 192 bytes. The prompt returns to "/home/will/Desktop/SpaceShell/build@".

```
Terminal
File Edit View Search Terminal Help
/home/will/Desktop/SpaceShell/build@ ls -l | sort
[INPUT] ls -l | sort
drwxrwxr-x 6 will will 4096 Ipan 15 00:12 CMakeFiles
-rw-r--r-- 1 will will 22 Ipan 15 01:04 archivo1.txt
-rw-rw-r-- 1 will will 13016 Ipan 14 22:44 CMakeCache.txt
-rw-rw-r-- 1 will will 2101 Ipan 14 23:23 cmake_install.cmake
-rw-rw-r-- 1 will will 9765 Ipan 14 23:23 Makefile
-rwxrwxr-x 1 will will 153800 Ipan 15 00:07 spaceshell
total 192
/home/will/Desktop/SpaceShell/build@
```

Figura 9: Ejecución de comandos con pipes simples

## 5.8. Tareas en Segundo Plano

El operador & permite ejecutar comandos en segundo plano, liberando inmediatamente el prompt para nuevos comandos. SpaceShell muestra los PIDs de los procesos iniciados.

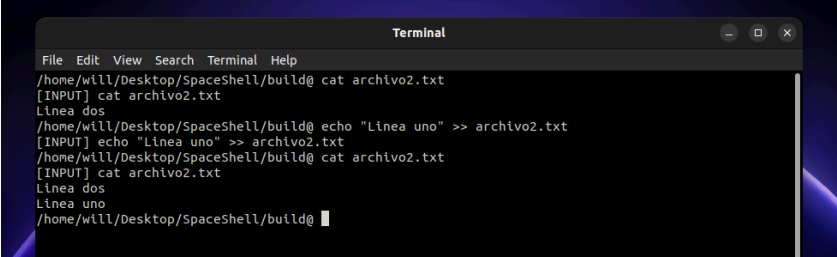
A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "/home/will/Desktop/SpaceShell@". The user enters "sleep 2 &". The output shows "[INPUT] sleep 2 &" and "[Proceso en segundo plano iniciado con PID(s): 7876 ]". The prompt returns to "/home/will/Desktop/SpaceShell@".

```
Terminal
File Edit View Search Terminal Help
/home/will/Desktop/SpaceShell@ sleep 2 &
[INPUT] sleep 2 &
[Proceso en segundo plano iniciado con PID(s): 7876 ]
/home/will/Desktop/SpaceShell@
```

Figura 10: Ejecución de comandos en segundo plano con &amp;

## 5.9. Redirección de Entrada y Anexar

SpaceShell soporta redirección de entrada (<) desde archivos y anexar (>>) contenido a archivos existentes sin truncarlos.

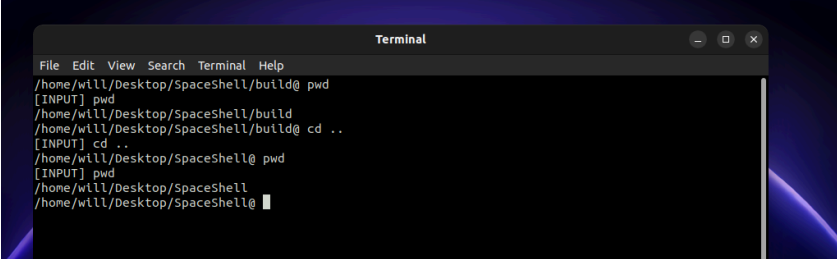
A terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is "/home/will/Desktop/SpaceShell/build@". The user enters "cat archivo2.txt". The output shows "[INPUT] cat archivo2.txt" and "Linea dos". The user enters "echo 'Linea uno' >> archivo2.txt". The output shows "[INPUT] echo 'Linea uno' >> archivo2.txt". The user enters "cat archivo2.txt". The output shows "[INPUT] cat archivo2.txt" and "Linea dos", "Linea uno". The prompt returns to "/home/will/Desktop/SpaceShell/build@".

```
Terminal
File Edit View Search Terminal Help
/home/will/Desktop/SpaceShell/build@ cat archivo2.txt
[INPUT] cat archivo2.txt
Linea dos
/home/will/Desktop/SpaceShell/build@ echo "Linea uno" >> archivo2.txt
[INPUT] echo "Linea uno" >> archivo2.txt
/home/will/Desktop/SpaceShell/build@ cat archivo2.txt
[INPUT] cat archivo2.txt
Linea dos
Linea uno
/home/will/Desktop/SpaceShell/build@
```

Figura 11: Redirección de entrada y salida en modo anexar

## 5.10. Comandos Internos (Built-ins)

Los built-ins como cd y pwd se ejecutan directamente en el proceso de la shell sin crear procesos hijos, permitiendo modificar el estado del intérprete (como el directorio actual).

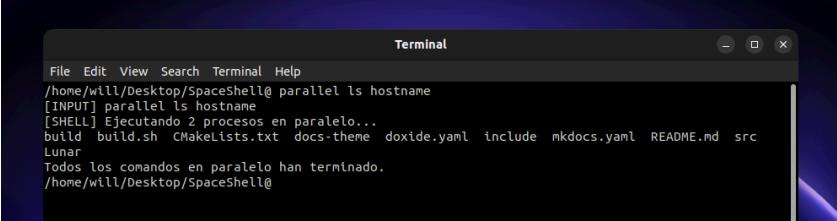


```
Terminal
File Edit View Search Terminal Help
/home/will/Desktop/SpaceShell/build@ pwd
[INPUT] pwd
/home/will/Desktop/SpaceShell/build
/home/will/Desktop/SpaceShell/build@ cd ..
[INPUT] cd ..
/home/will/Desktop/SpaceShell@ pwd
[INPUT] pwd
/home/will/Desktop/SpaceShell
/home/will/Desktop/SpaceShell@
```

Figura 12: Comandos internos cd y pwd

## 5.11. Concurrencia con Hilos

El comando `parallel` crea un hilo por cada comando especificado, ejecutándolos concurrentemente y sincronizándolos al final con `pthread_join`.

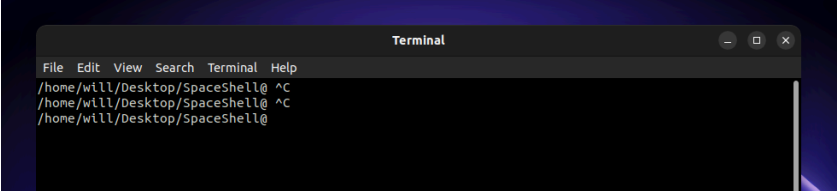


```
Terminal
File Edit View Search Terminal Help
/home/will/Desktop/SpaceShell@ parallel ls hostname
[INPUT] parallel ls hostname
[SHELL] Ejecutando 2 procesos en paralelo...
build build.sh CHakeLists.txt docs-theme doxide.yaml include nkdocs.yaml README.md src
lunar
Todos los comandos en paralelo han terminado.
/home/will/Desktop/SpaceShell@
```

Figura 13: Ejecución paralela de múltiples comandos con hilos

## 5.12. Manejo de Señales

SpaceShell captura `SIGINT` (Ctrl+C) para evitar terminación accidental de la shell, pero permite que los procesos hijos la reciban normalmente.



```
Terminal
File Edit View Search Terminal Help
/home/will/Desktop/SpaceShell@ ^C
/home/will/Desktop/SpaceShell@ ^C
/home/will/Desktop/SpaceShell@
```

Figura 14: Manejo de señales SIGINT sin terminar la shell

## 6. Conclusiones y Trabajos Futuros

Desarrollar SpaceShell fue un desafío, para diseñar, implementar, probar. No es un tipo de sistema que se desarrolle constantemente, por lo cuál requiere de una investigación previa (Revisión de Shells similares, compatibilidad en Sistemas UNIX, programación concurrente y paralela, etc). Sin embargo, permite abordar más técnicamente los conceptos aprendidos e interactuar con librerías de bajo nivel muy poderosas. SpaceShell fue construido priorizando la escalabilidad; pues, como Shell Básico, tiene un alto potencial a para integrar menús detallados, personalización de interfaces o manejo de niveles de usuario (Seguridad), funcionalidades relacionadas al usuario que agregan valor al SHELL.

## 7. Anexos

### 7.1. Script de Instalación (install.sh)

El proyecto incluye un script de instalación automatizado que configura el entorno de compilación y ejecuta SpaceShell:

```
#!/bin/bash

BUILD_TOOL="make"
TARGET=${1:-all}

mkdir -p build
cd build

if [ ! -f "Makefile" ]; then
    echo "[Setup] Generando archivos de compilación con CMake..."
    cmake ..
fi

echo "[Setup] Ejecutando target: $TARGET ..."
$BUILD_TOOL $TARGET
BUILD_EXIT_CODE=$?

if [ "$TARGET" = "all" ]; then
    if [ $BUILD_EXIT_CODE -eq 0 ]; then
        echo "[Setup] Corriendo SpaceShell..."
        ./spaceshell
    else
        echo "[SETUP] ERROR de compilación"
    fi
else
    if [ $BUILD_EXIT_CODE -eq 0 ]; then
        echo "[Setup] Target '$TARGET' ejecutado exitosamente"
    else
        echo "[SETUP] ERROR ejecutando target '$TARGET'"
    fi
fi
```

### 7.2. Comandos Externos Básicos

```
# Navegación de archivos
ls
ls -la
ls -lh /home
pwd
cd /tmp
cd ..
```

```
cd ~  
cd  
  
# Manipulación de archivos  
cat archivo.txt  
echo "Contenido de prueba"  
echo "Hola Mundo"  
cp origen.txt destino.txt  
mv archivo.txt nuevo_nombre.txt  
rm archivo_temp.txt  
touch nuevo_archivo.txt  
  
# Información del sistema  
date  
hostname  
top
```