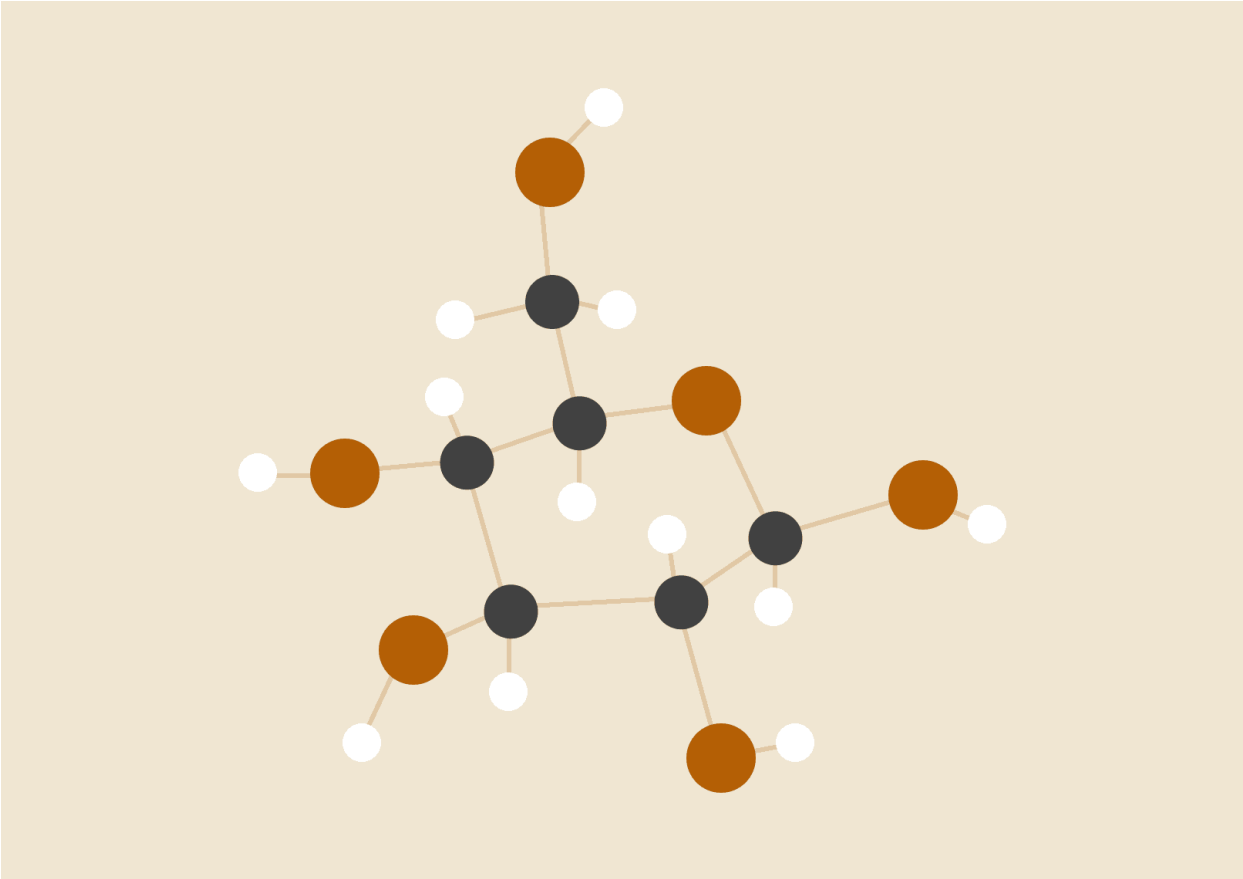
MINISHELL

Intérprete de comandos en C (Proyecto de Sistemas Operativos, UPM)



Alfonso Marín

14/07/2025

4º CURSO, GRADO EN INGENIERÍA INFORMÁTICA

1. DESCRIPCIÓN GENERAL DEL PROYECTO

Desarrollo de un intérprete de mandatos tipo *shell* en lenguaje C sobre sistema Unix, que permite ejecutar comandos, secuencias y redirecciones mediante llamadas al sistema (`fork`, `execvp`, `pipe`, `dup2`, `signal`, etc.).

El objetivo era comprender la gestión de procesos, señales, tuberías y entornos en sistemas POSIX.

2. FUNCIONALIDADES IMPLEMENTADAS

FUNCIONALIDAD	DESCRIPCIÓN	ESTADO
Ejecución de comandos	Permite ejecutar programas externos usando fork y execvp.	✅
Redirecciones <, >, >&	Implementadas con open, creat, dup2, siguiendo los modos 0666.	✅
Pipelines	Soporte completo de secuencias de comandos conectados con pipe.	✅
Ejecución en background &	Implementada: guarda bgpid y no bloquea la shell.	✅
Comandos internos	cd, umask, time, read implementados conforme al enunciado.	✅
Variables especiales	Manejo de mypid, bgpid, status.	✅
Expansión de variables	Sustitución de $VAR por su valor de entorno (getenv).	✅
Expansión de tildes	Soporte de ~ y ~usuario (usa getpwnam).	✅

Expansión de comodines	Implementación de ? usando glob().	✅
Manejo de señales	Ignora SIGINT y SIGQUIT en la shell principal; los hijos foreground los heredan.	✅
Prompt configurable	Usa la variable prompt, por defecto msh>.	✅
Variables de entorno dinámicas	Permite modificar el entorno con read y putenv	✅

3. TECNOLOGÍAS Y HERRAMIENTAS

Lenguaje: C (C99)

Entorno: Unix/Linux (servidor Triqui, ETSIINF-UPM)

Compilación: `gcc` con `Makefile`

Principales llamadas al sistema: `fork`, `execvp`, `waitpid`, `pipe`, `dup2`, `open`, `creat`, `chdir`, `umask`, `times`, `signal`

Bibliotecas usadas: `stdio.h`, `unistd.h`, `sys/types.h`, `sys/wait.h`, `fcntl.h`, `pwd.h`, `glob.h`

Depuración: `gdb`, `valgrind`

Entrega: `entrega.so msh.2024b`

4. DISEÑO E IMPLEMENTACIÓN

El minishell sigue un ciclo principal basado en obtain_order(), que analiza cada línea introducida por el usuario mediante el parser yacc/lex proporcionado.

Cada comando se ejecuta en un proceso hijo independiente, con control de tuberías y redirecciones.

Se añadieron funciones específicas para expansión de variables (expand_vars), tildes

(expand_tildes) y comodines (expand_wildcards), separando claramente la fase de análisis de la ejecución.

Los comandos internos (cd, umask, read, time) se implementaron dentro del proceso principal para preservar el estado del entorno.

5. EJECUCIÓN Y USO

El minishell se compila y ejecuta en un entorno **Linux/Unix** desde el directorio raíz del proyecto (`msh.2024b`).

1. Abrir una terminal en el directorio del proyecto `msh.2024b`
2. Compilar el programa con **Makefile**: `make`
3. Ejecutar el intérprete: `./msh`
4. Una vez en ejecución, el minishell muestra el prompt configurado (por defecto `msh>`).
5. Desde ahí pueden introducirse comandos del sistema, comandos internos o secuencias encadenadas con tuberías y redirecciones.

6. VALORACIÓN PERSONAL

Este proyecto me permitió comprender de forma práctica el funcionamiento interno de los intérpretes de comandos y la gestión de procesos en Unix.

Fue un reto de bajo nivel, muy útil para afianzar el uso de llamadas al sistema y depuración en C.

Obtuve una calificación de 60/100, demostrando una implementación funcional con margen de mejora en la gestión de errores y formato de salida.

7. CAPTURA Y RESULTADOS

Ejemplo 1 – Ejecución básica de comandos

```
msh> pwd
/mnt/c/Users/amari/Downloads/msh.2024b
msh> ls
Makefile  autores.txt  main.c  main.o  msh  parser.o  parser.y
pwd  scanner.l  scanner.o  y.tab.h
```

Ejemplo 2 – Redirección de salida y error

```
msh> ls inexistente >salida.txt >&error.txt
msh> cat error.txt
ls: cannot access 'inexistente': No such file or directory
```

Ejemplo 3 – Uso de tuberías (|)

```
msh> ls | grep .c
main.c
scanner.l
scanner.o
```

Ejemplo 4 – Ejecución en background (&)

```
msh> sleep 3 &
[1100]
msh> echo $bgpid
1100
```

Ejemplo 5 – Comando interno cd

```
msh> cd /tmp
/tmp
msh> pwd
/tmp
```

Ejemplo 6 – Comando interno umask

```
msh> umask
022
msh> umask 077
077
```

Ejemplo 8 – Comando interno read

```
msh> read VAR1 VAR2
hola mundo
msh> echo $VAR1
hola
msh> echo $VAR2
mundo
```

Ejemplo 9 – Expansión de variables de entorno ($VAR)

```
msh> echo $HOME
/home/alfon
msh> echo $mypid
1002
```

Ejemplo 10 – Expansión de tildes (~ y ~usuario)

```
msh> echo ~
/home/alfon
msh> echo ~root
/root
```

Ejemplo 12 – Variables especiales (status)

```
msh> ls inexistente
ls: cannot access 'inexistente': No such file or directory
msh> echo $status
2
```