Minishell

OBJETIVO DEL PROYECTO

Construir una shell funcional que:

- Ejecute comandos básicos con historial y manejo de señales.
- Gestione redirecciones (<, >, >>, <<), pipes (|), variables de entorno (\$, \$?), y builtins (cd, echo, pwd, export, unset, env, exit).
- Sea robusta: sin segfaults, leaks, ni errores de norma.

📤 1. Organización del equipo y enfoque de trabajo

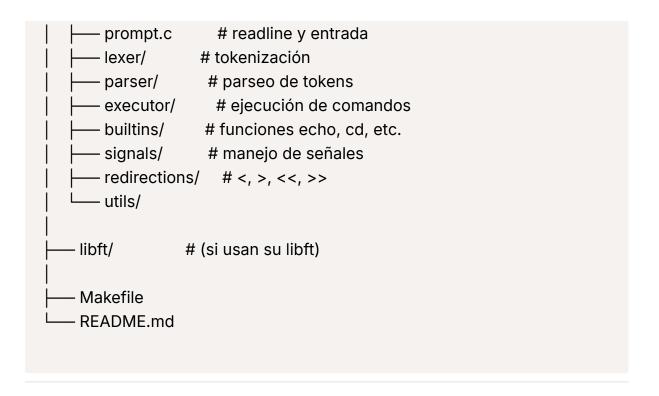
Roles generales sugeridos (intercambiables por etapas):

Persona 1	Persona 2
Parser & lexer	Ejecutador & builtins
Manejo de señales y pipes	Redirecciones y heredocs
Tests, leaks y norma	Makefile y entorno git

🔁 Revisan el código del otro antes de integrar al main branch.

2. Estructura de carpetas

```
makefile
CopiarEditar
minishell/
   – include/
                  # Headers
   — minishell.h
   parser.h
   - src/
   ├── main.c
```



3. Plan de desarrollo por etapas (con división de tareas)

✓ Semana 1: Preparación y estructura base
Estructura de carpetas y Makefile funcional.
☐
☐ implementar control-C, control-D, control-\ básicos.
☐ 🔧 Variable global para señales (única permitida).
Reunión diaria breve (15-30 min) para sincronizar avances y dudas.
✓ Semana 2: Lexer, Parser y estructura de comandos
☐ Control Con
Parser: armar estructura de comandos conectados (pipeline, redirecciones).
☐ S Manejo correcto de comillas y y y casos inválidos.

✓ Semana 3: Ejecución y built-ins
☐ Implementar ejecución básica (fork + execve).
☐ Manejo de PATH y comandos con rutas.
[] implementar built-ins (echo , cd , pwd , env , export , unset , exit).
Probar cada comando con argumentos, con y sin comillas.
✓ Semana 4: Redirecciones y Pipes
Redirecciones >, >>, <.
Pipes cmd1 cmd2 cmd3.
☐ Щ Heredoc <<.
Validar combinaciones: cat < file grep hola > salida.txt
▼ Semana 5: Variables de entorno y estado \$?
✓ Semana 5: Variables de entorno y estado \$?☐ Expansión de VARIABLE , \$? .
Expansión de \$variable, \$?.
 ☐ SEXPANSIÓN de \$VARIABLE , \$? ☐ ☐ Manejo correcto en comillas dobles y simples.
 ☐ SEXPANSIÓN de \$VARIABLE, \$?. ☐ ■ Manejo correcto en comillas dobles y simples. ☐ ■ Implementar actualización de exit status tras cada comando.
 ☐ SEXPANSIÓN de \$VARIABLE, \$?. ☐ Manejo correcto en comillas dobles y simples. ☐ Implementar actualización de exit status tras cada comando. ✓ Semana 6: Limpieza y validación
 ☐ SEXPANSIÓN DE \$VARIABLE, \$?. ☐ Manejo correcto en comillas dobles y simples. ☐ Implementar actualización de exit status tras cada comando. ✓ Semana 6: Limpieza y validación ☐ Comprobación de leaks con valgrind.
 ☐ ➡ Expansión de \$variable , \$? . ☐ ➡ Manejo correcto en comillas dobles y simples. ☐ ➡ Implementar actualización de exit status tras cada comando. ✓ Semana 6: Limpieza y validación ☐ ➡ Comprobación de leaks con valgrind . ☐ ➡ Revisión estricta de norma.
 ☐ Expansión de \$VARIABLE , \$? . ☐ Manejo correcto en comillas dobles y simples. ☐ Implementar actualización de exit status tras cada comando. ✓ Semana 6: Limpieza y validación ☐ Comprobación de leaks con valgrind . ☐ Revisión estricta de norma. ☐ Pruebas manuales con hoja de evaluación. ☐ Test de errores: entradas inválidas, comandos que fallan, múltiples

☑ Ej: echo "hola" | grep h > file.txt

Minishell 3

4. Tests sugeridos (según hoja de evaluación)

Verifica con checklist como:

- ctrl-c en línea vacía o con texto
- echo "hola mundo" y echo '\$USER'
- cat << EOF seguido de contenido
- Is | grep archivo > salida.txt
- export VAR=42 , echo \$VAR , unset VAR
- Comando inexistente jddajdja
- · Heredoc con EOF, comillas dobles, comillas simples

🔧 5. Git y colaboración

- Crear rama main protegida
- Usar ramas tipo feat/parser, fix/signals, etc.
- Hacer merge request con revisión del otro
- Usar issues para bugs/tareas pendientes
- Git log claro: "fix: redirection parsing crash", "feat: implement echo builtin"

Tarea	Responsable	Comentarios
Crear estructura de carpetas	Ambos	include/ , src/ , libft/ si aplica
Crear Makefile básico	Persona 1	Con flags -Wall -Werror -Wextra
Implementar prompt con readline	Persona 2	Mostrar minishell\$
Manejo de Ctrl+D (salida)	Persona 2	readline devuelve NULL
Control básico de señales (Ctrl+C, Ctrl+)	Persona 1	Usar sigaction()
Preparar .gitignore	Ambos	Ignorar a.out , *.o , *.dSYM , etc.

🍣 SEMANA 2 — Lexer & Parser

Tarea	Responsable	Comentarios
Diseñar struct para tokens	Persona 1	Tipos: CMD, ARG, PIPE, REDIR_OUT, etc.
Crear lexer que convierte línea a tokens	Persona 1	Manejar comillas, espacios, pipes
Crear parser que convierte tokens en comandos ejecutables	Persona 2	Comandos en pipes, redirecciones
Detectar errores de sintaxis	Persona 2	Pipes sin comandos, comillas mal cerradas

SEMANA 3 — Execve & Builtins

Tarea	Responsable	Comentarios
Implementar ejecución con execve y PATH	Persona 2	Comandos externos
Implementar echo [-n]	Persona 1	Usar ft_printf o write
Implementar cd con manejo de errores	Persona 1	chdir()
Implementar pwd , env	Persona 1	getcwd() y entorno
Implementar exit , unset , export	Persona 2	Controlar argumentos

SEMANA 4 — Redirecciones y Pipes

Tarea	Responsable	Comentarios
Implementar redirección > y >>	Persona 1	open() con flags
Implementar redirección <	Persona 2	open() + dup2()
Implementar heredoc <<	Persona 2	Leer hasta delimitador
Implementar pipe entre comandos	Persona 1	pipe() , fork() , dup2()
Soporte para combinaciones	Ambos	Ej: `cat < f

💡 SEMANA 5 — Expansión & Status

Tarea	Responsable	Comentarios
Implementar expansión de \$VAR, \$?	Persona 2	getenv() + seguimiento del status
Expansión dentro de comillas dobles	Persona 2	No expandir dentro de

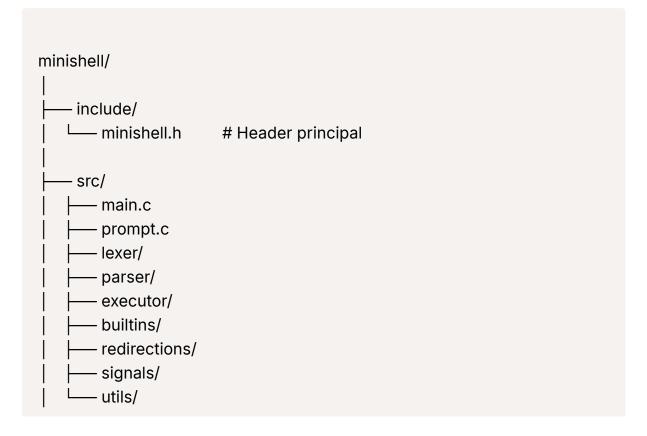
Integrar todo: lexer, parser, executor	Ambos	Refactor para limpiar funciones
Historial de readline con add_history()	Persona 1	

✓ SEMANA 6 — Norma, Tests y Defensa

Tarea	Responsable	Comentarios
Comprobar norma y eliminar warnings	Ambos	norminette , sin -Wall warnings
Valgrind y comprobación de leaks	Persona 1	Comprobar en cada comando
Simulacro de evaluación	Ambos	Usar hoja de evaluación oficial
Documentar proyecto y README.md	Persona 2	Cómo compilar, usar y organizar
Preparar defensa y dividir preguntas	Ambos	Cada uno domina al menos la mitad

1. Estructura inicial del repositorio

Crea la siguiente estructura de carpetas y archivos (puedes copiar este contenido en un script o crear a mano):



2. Makefile básico (estructura mínima)

Asegúrense de que cumpla con los flags y targets requeridos:

```
NAME = minishell
CC = cc
CFLAGS = -Wall -Wextra -Werror
SRCS = src/main.c \
    src/prompt.c \
    # Agrega aquí los .c que vayan apareciendo
OBJS = (SRCS:.c=.o)
all: $(NAME)$(NAME): $(OBJS)
  $(CC) $(CFLAGS) -o $(NAME) $(OBJS) -Ireadline
clean:
  rm -f $(OBJS)
fclean: clean
  rm -f $(NAME)
re: fclean all
.PHONY: all clean fclean re
```