Validación Previa a Tokenización - Minishell

1. Validación de Comillas y Caracteres Especiales

1.1 Comillas Balanceadas

- **Regla**: Toda comilla abierta debe tener su cierre correspondiente
- **Casos de error**:

```bash
echo "hello       # Falta cierre de comilla doble
echo 'world       # Falta cierre de comilla simple
echo "test'world  # Comillas no coinciden
```

- **Implementación**: Recorre el string rastreando comillas simples/dobles, ignorando caracteres especiales dentro de ellas
- **Nota**: Las comillas pueden estar anidadas: `echo "It's working"` es válido

1.2 Caracteres de Escape

- **Regla**: Si una línea termina con `\` (backslash), la siguiente línea es continuación
- **En minishell**: Generalmente ignoras esto (bash lo maneja en readline)
- **Casos a evitar**: Backslash suelto al final sin propósito válido

---

2. Validación de Pipes

2.1 Pipes en Posiciones Inválidas

- **Error**: Pipe al inicio

```bash
| cat file.txt        # Error
```

- **Error**: Pipe al final

```bash
cat file.txt |        # Error
```

- **Error**: Pipes consecutivos

```bash
cat file.txt || ls      # OK (operador lógico)
cat file.txt || | ls    # Error
```

2.2 Validación

- Detecta `|` fuera de comillas
- Verifica que no esté al inicio (después de espacios/inicio)
- Verifica que no esté al final (antes de espacios/fin)
- Verifica que no haya `||` seguido inmediatamente de `|` (aunque `||` es válido)

---

3. Validación de Redirecciones

3.1 Redirecciones Inválidas

- **Error**: Redireccionamiento sin archivo

```bash
cat file.txt >          # Error: falta archivo destino
cat file.txt < 2>&1     # Depende del contexto (minishell simplificado)
ls >>                   # Error: falta archivo
```

- **Error**: Múltiples redireccionamientos sin comando

```bash
> file1.txt > file2.txt # Error: no hay comando
```

- **Error**: Redireccionamiento en posición inválida

```bash
> cat file.txt          # Error: redireccionamiento sin comando previo
cat > | ls              # Ambiguo: ¿redireccionar a qué?
```

3.2 Tipos de Redirecciones

- `>` (output)
- `>>` (append)
- `<` (input)
- `<<` (heredoc - opcional en minishell)
- `>&` (redireccionar stderr - opcional)
- `<&` (redireccionar stdin - opcional)

3.3 Validación

- Después de `>`, `>>`, `<` debe haber un filename
- No puede haber operadores pipe/lógicos entre redireccionamiento y filename
- Múltiples redirecciones en el mismo comando son válidas: `cat < in.txt > out.txt`

---

4. Validación de Operadores Lógicos

4.1 Operadores Lógicos (`&&` y `||`)

- **Válido**:

```bash
ls && echo "ok"            # AND
ls || echo "error"         # OR
ls && echo "ok" || echo "fail"  # Combinados
```

- **Inválido**:

```bash
&& ls                  # Comienza con operador
ls &&                  # Termina con operador
ls && && echo "test"       # Operadores consecutivos
ls || || echo "test"       # Operadores consecutivos
```

4.2 Validación

- No puede estar al inicio ni al final
- No puede haber `&&` o `||` consecutivos
- Debe haber comando antes y después

---

5. Validación de Paréntesis y Subshells

5.1 Paréntesis Balanceados (si implementas)

- **Válido**:

```bash
(ls && cat file.txt) | grep "test"
```

- **Inválido**:

```bash
(ls && cat file.txt | grep "test"   # Falta cierre
(ls && (cat file.txt))              # OK
ls && (cat || (wc -l))              # OK
```

5.2 Validación

- Cuenta paréntesis abiertos vs cerrados
- Verifica que no haya paréntesis cerrado sin abrir
- Paréntesis debe contener al menos un comando válido

---

6. Validación de Espacios en Blanco

6.1 Input Vacío

- **Error**: String completamente vacío o solo espacios

```bash
""          # Error o ignorar según implementación
"   "       # Solo espacios
```

6.2 Validación

- Trim el input al inicio/final
- Si queda vacío, retorna error o ignora

---

7. Orden de Validación Recomendado

1. Verificar que input no esté vacío (después de trim)
2. Validar comillas balanceadas
3. Validar paréntesis balanceados (si aplica)
4. Validar pipes (posición, no consecutivos)
5. Validar redirecciones (sintaxis, posición)
6. Validar operadores lógicos (posición, no consecutivos)
7. Validar secuencia general (no operador seguido de otro inmediatamente)

8. Casos Especiales a Considerar

8.1 Contenido Dentro de Comillas

- Todo dentro de comillas es literal (no se valida sintaxis)

```bash
echo "| && > <"     # Válido, la sintaxis dentro se ignora
echo "|"           # Válido
```

8.2 Variables y Expansiones (opcional)

- Si implementas expansión de variables: `$VAR`
- Valida que `$` no esté al final sin nombre de variable
- Dentro de comillas simples `''` no se expanden variables

8.3 Comandos Reservados

- En bash existen: `if`, `while`, `for`, etc.
- En minishell simplificado: generalmente no es necesario validarlos

8.4 Caracteres Especiales Dentro de Palabras

- `*`, `?`, `[` (glob patterns - opcional)
- Generalmente se validan pero se procesan en expansión

---

9. Mensajes de Error Coherentes con Bash

```bash
# Bash error examples:
bash: syntax error near unexpected token '|'
bash: syntax error near unexpected token '>'
bash: unexpected EOF while looking for matching '`'
bash: syntax error: unexpected end of file
bash: syntax error: `&&' unexpected
```

Para minishell, mantén mensajes similares:

```
minishell: syntax error near unexpected token '|'
minishell: syntax error: missing closing quote
minishell: syntax error near unexpected token `&&'
```

---

10. Pseudocódigo de Validación Principal

pseudocode

```
function validate_input(input):
    // 1. Trim y verificar vacío
    trimmed = trim(input)
    if empty(trimmed):
        return ERROR_EMPTY_INPUT

    // 2. Validar comillas
    if not validate_quotes(input):
        return ERROR_UNMATCHED_QUOTES

    // 3. Validar paréntesis
    if not validate_parentheses(input):
        return ERROR_UNMATCHED_PARENTHESES

    // 4. Validar operadores sin comillas (ignorar dentro de comillas)
    operators = extract_operators_outside_quotes(input)

    if not validate_pipes(operators):
        return ERROR_INVALID_PIPE

    if not validate_redirects(operators):
        return ERROR_INVALID_REDIRECT

    if not validate_logical_ops(operators):
        return ERROR_INVALID_LOGICAL_OP

    // 5. Validar secuencia general
    if not validate_operator_sequence(operators):
        return ERROR_SYNTAX

    return OK
```

---

11. Funciones Sugeridas a Implementar

c

```c
// Validación básica
int is_empty_input(char *input);
int has_unmatched_quotes(char *input);
int has_unmatched_parentheses(char *input);

// Validación de operadores
int has_invalid_pipes(char *input);
int has_invalid_redirects(char *input);
int has_invalid_logical_ops(char *input);

// Utilidades
int is_inside_quotes(char *input, int pos);
char* get_operator_context(char *input, int pos);
int count_occurrences_outside_quotes(char *input, char *pattern);

// Función principal
int validate_input(char *input, char **error_msg);
```