Casos de Redirecciones en Bash - Válidos e Inválidos

1. REDIRECCIÓN DE SALIDA (OUTPUT REDIRECTION)

1.1 Sintaxis Básica: `>`

VÁLIDO - Redirige stdout a un archivo (trunca si existe):

```bash
echo "hello" > file.txt              # ✓ Crea o sobrescribe file.txt
cat file.txt > output.txt            # ✓ Salida en output.txt
ls > directory_list.txt              # ✓ Lista el directorio
command > file.txt                   # ✓ Salida estándar a archivo
```

VÁLIDO - Redirecciones múltiples:

```bash
echo "test" > file1.txt > file2.txt      # ✓ Solo file2.txt recibe la salida (última)
```

VÁLIDO - Redirección en cualquier posición:

```bash
> file.txt echo "hello"               # ✓ La redirección puede estar al inicio
echo > file.txt "hello"               # ✓ En el medio también funciona
echo "hello" > file.txt               # ✓ Al final (forma más común)
```

INVÁLIDO - Sin archivo destino:

```bash
echo "hello" >                        # ✗ Error: No hay archivo después de >
cat file.txt >                       # ✗ Error: Falta nombre del archivo
command >                            # ✗ Error: Redirección incompleta
```

INVÁLIDO - Redirección sin comando:

```bash
> file.txt                           # ✗ Error: No hay comando antes
> file.txt > file2.txt               # ✗ Error: Ninguno de los dos es válido
```

INVÁLIDO - Operador pipe/lógico después de redirección:

```bash
echo "test" > | cat                    # ✗Error: No se puede poner | después de >
cat > && echo "ok"                     # ✗Error: Falta el archivo y hay &&
```

1.2 Sintaxis: `>>` (Append)

VÁLIDO - Añade contenido al final del archivo:

```bash
echo "line 1" > file.txt               # ✓Crea archivo
echo "line 2" >> file.txt              # ✓Añade al final (no trunca)
date >> log.txt                        # ✓Añade timestamp al log
```

VÁLIDO - En archivos que no existen (los crea):

```bash
echo "first" >> nonexistent.txt        # ✓Crea el archivo
```

INVÁLIDO - Sin archivo:

```bash
echo "test" >>                         # ✗Error: Falta archivo destino
command >>                             # ✗Error: Redirección incompleta
```

1.3 Sintaxis: `>|` (Force overwrite - ignora noclobber)

VÁLIDO - Fuerza sobrescritura aunque noclobber esté activado:

```bash
set -o noclobber
echo "test" >| file.txt                # ✓Sobrescribe aunque esté protegido
```

INVÁLIDO:

```bash
echo "test" >|                         # ✗Error: Falta archivo
```

1.4 Sintaxis: `2>` (Redirect stderr)

VÁLIDO - Redirige errores a archivo:

```bash
ls /nonexistent 2> error.log          # ✓ Error a archivo
command 2> /dev/null                  # ✓ Descarta errores
```

VÁLIDO - Múltiples redirecciones:

```bash
command > output.txt 2> error.txt     # ✓ Salida y error a archivos diferentes
command 1> out.txt 2> err.txt         # ✓ Explícito con números de descriptor
```

INVÁLIDO:

```bash
ls /nonexistent 2>                    # ✗ Error: Falta archivo
command 2> >                          # ✗ Error: Sintaxis inválida
```

1.5 Sintaxis: `&>` o `2>&1` (Redirect stdout y stderr)

VÁLIDO - Ambas salidas a un mismo archivo:

```bash
command &> output.txt                 # ✓ stdout y stderr juntos (bash 4.0+)
command > output.txt 2>&1             # ✓ Forma clásica compatible
command 2>&1 | grep error             # ✓ stderr y stdout al pipe
```

VÁLIDO - En cualquier orden:

```bash
command 2>&1 > output.txt             # ✓ Primero copia stderr a stdout, luego redirige stdout
command > output.txt 2>&1             # ✓ Primero redirige stdout, luego copia stderr a stdout
```

INVÁLIDO:

```bash
command &>                            # ✗ Error: Falta archivo
ls &> > file.txt                      # ✗ Error: Sintaxis duplicada
```

1.6 Sintaxis: `&>>` (Append stdout y stderr)

VÁLIDO:

```bash
command &>> log.txt                    # ✓Añade stdout y stderr al log
```

INVÁLIDO:

```bash
command &>>                            # ✗Error: Falta archivo
```

1.7 Sintaxis: `n>file` (Redirect descriptor específico)

VÁLIDO:

```bash
echo "output" 1> file.txt              # ✓descriptor 1 (stdout) a archivo
ls 2> errors.txt                       # ✓descriptor 2 (stderr) a archivo
exec 3> debug.log                      # ✓descriptor 3 a archivo
command 3>&1                           # ✓Copia fd 3 a fd 1
```

INVÁLIDO - Descriptor no válido:

```bash
command a> file.txt                    # ✗Error: 'a' no es descriptor válido
command 10> file.txt                   # ✗Potencialmente problemático (fd > 9)
```

---

2. REDIRECCIÓN DE ENTRADA (INPUT REDIRECTION)

2.1 Sintaxis Básica: `<`

VÁLIDO - Lee archivo como stdin:

```bash
cat < file.txt                         # ✓Archivo como entrada
wc -l < data.txt                       # ✓Cuenta líneas del archivo
grep "pattern" < input.txt             # ✓Busca en archivo
```

VÁLIDO - En cualquier posición:

```bash
< file.txt cat                         # ✓Redirección al inicio
cat < file.txt                         # ✓Al final (forma más común)
```

INVÁLIDO - Sin archivo origen:

```bash
cat <                          # ✗ Error: Falta archivo fuente
command <                      # ✗ Error: Redirección incompleta
```

INVÁLIDO - Archivo que no existe:

```bash
cat < /nonexistent/file.txt    # ✗ Error: Archivo no existe
```

2.2 Sintaxis: `n<file` (Redirect descriptor específico)

VÁLIDO:

```bash
exec 3< datafile.txt           # ✓ Asigna fd 3 al archivo
read -u 3 line                 # ✓ Lee desde fd 3
```

INVÁLIDO:

```bash
command a< file.txt            # ✗ Error: 'a' no es descriptor
```

---

3. HERE-DOCUMENTS Y HERE-STRINGS

3.1 Here-Document: `<<`

VÁLIDO - Múltiples líneas:

```bash
cat << EOF
line 1
line 2
EOF                             # ✓ Válido

cat << 'EOF'
No expande $variables
EOF                             # ✓ Válido (comillas simples)

cat <<- EOF
	Tabulaciones removidas
EOF                             # ✓ Válido (<<- elimina tabs)
```

VÁLIDO - Sin espacio antes del delimitador:

```bash
cat <<EOF
content
EOF                             # ✓ Válido

cat <<'DELIMITER'
content
DELIMITER                       # ✓ Válido
```

INVÁLIDO - Delimitador incorrecto:

```bash
cat << EOF
content
NOTEOF                          # ✗ Error: El delimitador debe ser exacto
```

INVÁLIDO - EOF no en nueva línea:

```bash
cat << EOF
content
EOF text                        # ✗ Error: EOF debe estar solo
```

3.2 Here-String: `<<<` (bash only)

VÁLIDO:

```bash
cat <<< "hello world"                    # ✓Pasa string como stdin
wc <<< "count these words"               # ✓Cuenta palabras
grep "pattern" <<< "$variable"           # ✓Busca en variable
```

INVÁLIDO:

```bash
cat <<<                                  # ✗Error: Falta el string
command <<<                              # ✗Error: Redirección incompleta
```

---

4. REDIRECCIÓN BIDIRECCIONAL

4.1 Sintaxis: `<>` (Open for read/write)

VÁLIDO:

```bash
exec 3<> file.txt                        # ✓Abre para lectura y escritura
exec 3<> /dev/tcp/host/port              # ✓Conexión TCP (bash)
```

INVÁLIDO:

```bash
cat <>                                   # ✗Error: Falta archivo
command <>file.txt                       # ✗Generalmente no se usa así
```

---

5. DUPLICACIÓN DE DESCRIPTORES

5.1 Sintaxis: `n>&m` (Duplicate output fd)

VÁLIDO:

```bash
command 1>&2                             # ✓Copia stdout a stderr
exec 3>&1                                # ✓Asigna fd 3 copia de fd 1
command >&2                              # ✓Equivalente a 1>&2
```

VÁLIDO - Cerrar descriptores:

```bash
exec 3>&-                    # ✓ Cierra fd 3
exec 2>&-                    # ✓ Cierra stderr
```

INVÁLIDO:

```bash
command 1>&                  # ✗ Error: Falta descriptor
command >&                   # ✗ Error: Incompleto
```

5.2 Sintaxis: `n<&m` (Duplicate input fd)

VÁLIDO:

```bash
exec 3<& 0                   # ✓ Copia stdin a fd 3
exec 0<& 5                   # ✓ Reasigna stdin desde fd 5
```

INVÁLIDO:

```bash
command 0<&                  # ✗ Error: Falta descriptor
```

---

6. CASOS COMPLEJOS Y COMBINACIONES

6.1 Redirecciones Múltiples - VÁLIDO

```bash
# Múltiples redirecciones se procesan de izquierda a derecha
command > out.txt 2> err.txt          # ✓ stdout a out.txt, stderr a err.txt
command 2>&1 | tee log.txt            # ✓ ambos a pipe y a archivo
exec 3< input.txt 4> output.txt       # ✓ múltiples fds
cat < in.txt > out.txt 2> errors.txt  # ✓ entrada y dos salidas
```

6.2 Redirecciones Múltiples - INVÁLIDO

```bash
command > out.txt >                   # ✗ Error: > incompleto al final
command > out.txt 2> 2>               # ✗ Error: 2> incompleto
cat < > file.txt                      # ✗ Error: < sin origen, > necesita comando
```

7. INTERACCIÓN CON PIPES

7.1 Válido - Redirección + Pipe

```bash
cat file.txt > output.txt | grep error      # ✓ Válido (pipe después de redirección)
cat < input.txt | sort > sorted.txt         # ✓ Entrada, pipe, salida
command 2>&1 | tee log.txt                  # ✓ Ambas salidas a pipe
```

7.2 Inválido - Redirección + Pipe

```bash
echo "test" > | cat                 # ✗ Error: | inmediatamente después de >
cat > | grep pattern                # ✗ Error: Falta archivo entre > y |
cat file.txt 2> | sort              # ✗ Error: 2> sin archivo antes de |
```

---

8. INTERACCIÓN CON OPERADORES LÓGICOS

8.1 Válido - Redirección + Lógicos

```bash
command > file.txt && echo "ok"          # ✓ Redirección, luego lógica
command > file.txt || echo "fail"        # ✓ Válido
command 2> error.txt && command2         # ✓ Válido
```

8.2 Inválido - Redirección + Lógicos

```bash
echo "test" > && echo "ok"               # ✗ Error: > sin archivo
cat file.txt > || echo "error"           # ✗ Error: > sin archivo
command 2> && command2                   # ✗ Error: 2> sin archivo
```

---

9. CASOS ESPECIALES CON COMILLAS Y VARIABLES

9.1 Válido

```bash
echo "hello" > "$filename"               # ✓ Variable como nombre archivo
echo "test" > "file with spaces.txt"     # ✓ Archivo con espacios
echo "content" > file$suffix.txt         # ✓ Variable en nombre
```

9.2 Inválido

```bash
echo "test" > $                    # ✗ Error: Variable incompleta
echo "test" > ""                   # ✗ Error: Nombre vacío
echo "test" > 'file.txt            # ✗ Error: Comilla sin cerrar
```

---

10. RESUMEN DE REGLAS GENERALES

1. **Siempre necesitan archivo destino/origen**: `>`, `<`, `>>`, `&>`, `2>`, etc.
2. **No pueden estar al final sin completarse**: `echo >` es inválido
3. **Múltiples redirecciones se procesan left-to-right**: orden importa
4. **El pipe `|` no puede venir inmediatamente después de operador incompleto**
5. **Descriptores válidos**: 0 (stdin), 1 (stdout), 2 (stderr), 3-255 (personalizados)
6. **Las comillas protegen**: Contenido dentro de comillas se trata como literal
7. **Duplicación `>&` y `<&`**: Necesitan descriptor válido
8. **Here-documents `<<`: El delimitador debe estar exacto en su propia línea
9. **Here-strings `<<<`: Solo en bash, necesitan contenido después

---

11. REGLAS PARA MINISHELL

Para validar redirecciones en minishell, debes rechazar:

- Operador redirección sin archivo: `echo test >`
- Operador redirección al final: `cat file.txt >`
- Múltiples operadores sin comando entre ellos: `> file1.txt > file2.txt` (sin comando antes)
- Operadores lógicos/pipes inmediatamente después de redirección incompleta
- Descriptores no numéricos: `echo test a> file`
- Archivos vacíos: `echo test > ""`