Linux: Redirecionamento de E/S e Descritores de Arquivos

Redirecionamento de E/S (Entrada/Saída) é um conceito bastante importante para quem trabalha com sistemas Unix-like. Consiste em alterar a saída, ou entrada padrão de um comando.

Vamos considerar o seguinte exemplo:

```
$ cat > arquivo.txt
```

Apenas um arquivo de teste para mostrar um redirecionamento de saída.

^**D** (Pressione Ctrl+D para salvar o arquivo)

O comando acima envia a saída do comando **cat** para o arquivo **arquivo.txt**.

$ ls /home/jaques

O comando acima irá mostrar na tela todos os arquivos do diretório /home/jaques

$ ls /home/jaques > /tmp/ls.txt

Neste momento, enviamos a saída do comando **ls** para o arquivo **/tmp/ls.txt**. Fizemos um redirecionamento de saída **(output redirection)**.

Se fizermos

$ ls /opt >> /tmp/ls.txt

O ">>" é uma maneira de adicionar a sa**ída de ls /opt** ao fim da linha **de /tmp/ls.txt.** O operador ">" apaga tudo que tem no arquivo alvo e adiciona saída. O ">>" é como um "append", adiciona no fim da linha.

$ sort
10
11
1
15
^D

O comando sort, por padrão, pega os dados de entrada do teclado, ordena-os e envia o resultado para a tela. Para pegar a entrada de um arquivo, poderiamos fazer o seguinte:

```
$ cat > numeros.txt
1
4
2
10
^D

$ sort < numeros.txt
1
10
11
2
3
```

Primeiro, criamos um arquivo chamado **numeros.txt** com o comando **cat**. Em seguida, o arquivo foi fornecido como entrada para o comando **sort**, o qual ordenou e imprimiu o resultado. No último comando executado, fizemos um rediecionamento de entrada **(input redirection)**.

DESCRITORES DE ARQUIVOS

Quando um processo precisa acessar um arquivo, ele faz isso através do descritor do arquivo: um número inteiro positivo, que o processo usa para referenciar a um determinado arquivo aberto. Assim, todo processo em um sistema UNIX mantém uma tabela de referências aos descritores de arquivos usados pelo mesmo. Os descritores propriamente dito são mantidos pelo kernel. O tamanho dessa tabela varia de sistema para sistema. Nos linux atuais, o tamanho padrão é 256. Cada processo UNIX dispõe de 20 **descritores de arquivo**, ordenados de **0 a 19**.

Quando um programa é executado, 3 arquivos são automaticamente abertos e associados à esse processo. Abaixo, temos a listagem desses três arquivos:

Arquivo	Número	Uso	Exemplo
stdin	0	entrada padrão	Teclado
stdout	1	saída padrão	Tela
stderr	2	erro padrão	Tela

Obs: **Os outros 17 descritores estão disponíveis para arquivos.**

Na maior parte das vezes, em interrupções direcionamos a saída passando o descritor para ebx (ou rbx, se 64 bits)

É exatamente por isso que quando queremos que alguma coisa seja enviada para tela, usamos mov ebx,1, pois 1 "é o descritor do arquivo tela". Do mesmo modo, se estamos nos referindo ao descritor do arquivo teclado usamos mov ebx, 0. TUDO É ARQUIVO PARA O MUNDO Unix-Like.

Os primeiros dois arquivos(stdin e stdout) já foram vistos nos exemplos anteriores. O arquivo stderr (descritor 2) é usado pelos programas para imprimir as mensagens de erro(geralmente na tela). Podemos redirecionar a saída de um descritor de arquivo para um arquivo com a seguinte sintaxe:

Sintaxe:
numero-descritor> nome_do_arquivo

Exemplos (assumindo que o arquivo teste.txt não existe no diretório corrente):

$ rm teste.txt
rm: imposível remover
`teste.txt': Arquivo ou diretório inexistente

O comando **rm** imprimiu um erro, uma vez que o arquivo não existe.

$ rm teste.txt > erro.log
rm: imposível remover `teste.txt': Arquivo ou diretório inexistente

O que acontecu no exemplo acima? A mensagem de erro não deveria ter sido redirecionada para o arquivo erro.log? A mensagem, no entanto, apareceu na tela, e o arquivo erro.log foi criado vazio. Isso ocorreu por que o comando rm mandou a mensagem de erro para o seu dispositivo de erro(stderr). O **">"**, por padrão, sempre redireciona o que foi mandado para a saída padrão (stdout). Para contornar esse problema, poderiamos usar o comando da seguinte forma:

$ rm teste.txt 2> erro.log

Obs: Não pode haver espaço entre o **"2"** e o **">"**

O **"2>"** direciona o erro padrão para o arquivo **erro.log**. Dessa forma, não vemos a mensagem de erro impresso na tela, mas se olharmos o conteúdo do arquivo erro.log:

$ cat erro.log

rm: imposível remover `teste.txt': Arquivo ou diretório inexistente

Veremos que ele contém a mensagem de erro gerado pela execução do comando rm.

Shell script

Um shell script ou simplesmente script pode ser composto de parâmetros, no qual usa-se as seguintes variáveis:

$# serve para armazenar o número de argumentos ou parâmetros digitados.

$1, $2,...$n seriam os parâmetros que estaria utilizando no meu script.

Vamos criar um shell script, embora pudesse ser feito com um editor de texto:

```
$ cat > teste.sh
if [ $# -ne 2 ]         #se $# diferenteDe 2
then                    #então
echo "Erro : Parametros nao foram fornecidos"
echo "Uso : $0 numero1 numero2"
exit 1                  #saia
fi                      #fim_se
ans=`expr $1 + $2`
echo "Soma: $ans"
^D
```

Adicionando permissão de execução:

$ chmod a+x teste.sh

Executando o arquivo:

```
./teste.sh
Erro : Parametros nao foram fornecidos
Uso : ./teste.sh numero1 numero2
./teste.sh > erro.log
./teste.sh 1 2
Soma: 3
```

No primeiro exemplo, nosso script imprimiu um erro, indicando que não foram fornecidos parâmetros. No segundo exemplo, conseguimos redirecionar a mensagem de erro usando o >, uma vez que a saída padrão do comando echo é o stdout. Assim, devemos mandar o echo escrever na saída de erro, ao invés da saída padrão. Isso pode ser resolvido da seguinte forma:

echo “Erro : Parametros nao foram fornecidos” 1>&2

echo “Uso : $0 numero1 numero2″ 1>&2

Vamos tentar executar nosso script novamente:

$ ./teste.sh > erro.log
Erro : Parametros nao foram fornecidos
Uso : ./teste.sh numero1 numero

Agora, a mensagem de erro está sendo impressa no stderr e não mais no stdout. **O "1>&2" ao final do comando echo redireciona a saída padrão (stdout) para o dispositivo de erro padrão (stderr)**. A sintaxe é:

origem>&destino