Redireccionamento de E/S do Sistema

1. Introdução

Se voltarmos um pouco para os conhecimentos básicos da informática, podemos lembrar que existem dois tipos principais de interfaces entre o usuário e o computador: interface de entrada (teclado, mouse,…) e interface de saída (monitor). Para entender o funcionamento das entradas e saídas dos comandos, basta inicialmente considerarmos que o teclado é o dispositivo padrão de entrada de dados (STANDARD INPUT ou de forma simplificada "STDIN") e o monitor é o dispositivo padrão de saída (STANDARD OUTPUT ou "STDOUT" e STANDARD ERROR ou simplesmente "STDERR").

Por exemplo, quando digitamos o comando *ls*:

```
$ ls
arquivo1  arquivo2  arquivo3
```

O comando retornou como saída na tela à lista dos arquivos contidos na pasta corrente: `arquivo1, arquivo2, arquivo3`, retornou na tela, pois não especificando outra saída, e o monitor é a saída padrão. A mesma situação pode ser analisada com o comando *cat*:

```
$ cat contacto.txt
   ...Informações de Contacto...
```

Neste caso, o comando *cat* recebeu um argumento (contacto.txt), o qual podemos chamar de entrada para o comando. A entrada padrão é o teclado e como não era objectivo digitar, mas sim buscar a informação de entrada a partir de outra fonte, tivemos que informar o nome da entrada a usar, neste caso o arquivo `contacto.txt`. Recebido o argumento, o *cat* mandou para a tela (saída padrão), o conteúdo do arquivo contacto.txt.

2. Redirecionando a saída de comandos

Nos sistemas Linux é possivel redirecionar as saídas dos comandos para outro destino que não é a saída padrão. Para alterar a saída dos comandos utilizamos os sinais de **">"** e **">>"**.

2.1 Redirecionando a saída com o sinal “ >”

O sinal **“>”** redireciona a saída de um programa/comando/script para algum dispositivo ou arquivo ao invés do dispositivo de saída padrão (monitor). Quando é usado com arquivos, este redirecionamento cria ou substitui o conteúdo do arquivo.

Por exemplo, se quisermos alterar a saída do comando *ls* anterior, para ao invés da listagem aparecer na tela, ser gravada num arquivo `lista_de_arquivos.txt temos`:

```
$ ls > lista_de_arquivos.txt
```

Neste caso, ao invés de mostrar na tela, o dispositivo de saída agora é o arquivo lista_de_arquivos.txt e por isso o comando *ls* vai escrever a sua saída neste arquivo. Se olharmos o conteúdo com o comando *cat*, veremos a lista `“arquivo1, arquivo2, arquivo3”.`

Um uso muito comum disto é redirecionar a saída para o */dev/null*, que é um dispositivo/arquivo especial que anula tudo que vai para ele. Ao fazer:

```
$ cat contacto.txt > /dev/null
```

A informação que está dentro do arquivo contacto.txt não vai a lugar algum: nem para a tela, nem para um arquivo ou dispositivo algum.

2.2 Redirecionando a saída com o sinal “ >>”

Este redirecionador é usado também para desviar a saída padrão de um programa/comando/script para algum dispositivo ou arquivo. A diferença entre o redireccionamento simples **“>”** e o duplo **“>>”**, é caso seja usado com arquivos, este último adiciona a saída no final do arquivo, ou seja, o redirecionador >> não apaga o conteúdo do arquivo muito menos recria o arquivo.

Por exemplo, podemos acrescentar a saída do comando *ls* ao arquivo `lista_de_arquivos.txt` do exemplo anterior usando:

```
ls >> lista_de_arquivos.txt.
```

Use o comando *cat* para visualizar o conteúdo do arquivo `lista_de_arquivos.txt`. Se executarmos este comando várias vezes, veremos que dentro do arquivo haverá várias linhas com a mesma listagem de arquivos do comando *ls*.

3. Redireccionando a entrada dos comandos

Para alterar a entrada dos comandos utilizamos os sinais de **“<”** e **“<<”**.

3.1 Redirecionando a entrada com o sinal " <"

Este símbolo "<", faz com que a entrada padrão seja direcionada para um comando ou arquivo. Geralmente é utilizado em comandos que necessitam de arquivos como argumentos. Por exemplo, o comando *sort* ordena as linhas de um arquivo alfabeticamente e sua sintaxe é `sort arquivo`. Podemos redirecionar o conteúdo do arquivo diretamente, sem passá-lo como argumento:

```
$ sort < teste.txt
```

Cada linha do arquivo `teste.txt` será passada para o comando *sort*, que por sua vez irá ordenar e mostrar na tela. Em muitos casos, especificar "<" funciona exactamente como especificar o nome do arquivo como argumento do comando.

Vamos agora redirecionar essa saída para outro arquivo:

```
$ sort < teste.txt > teste_ordenado.txt
```

Outro exemplo muito comum do uso da alteração do STDIN é quando um programa necessita de sub-comandos. Por exemplo, a shell bash, quando executada, fornece um *prompt* para o usuário digitar os comandos via teclado. Podemos automatizar essa digitação colocando todos os comandos num arquivo. Chamaremos este arquivo de *comandos.txt*, com o conteúdo:

```
cd /root
ls
cd /usr/
ls
cd /etc
cat /etc/passwd
cd /usr/local/bin
pwd
```

Agora podemos executar uma shell, passando como entrada este arquivo:

```
$ bash < comandos.txt
```

Ao invés do bash fornecer um prompt para a digitação de comandos, ele irá ler e executar todo o conteúdo do arquivo comandos.txt, depois sair. Este é o conceito de shell-script, mas utilizado de uma forma mais crua. Apesar de ser útil, geralmente usa-se uma forma diferente para criar shell-scripts.

Outro exemplo bastante usado, seria restaurar um arquivo do banco de dados MySQL de forma automática. Nesta situação, temos um arquivo chamado base.sql que contém vários comandos SQL de criação de tabelas e dados. Usa-se então o redirecionador de entrada para que ao invés do usuário digitar todos estes comandos, eles sejam passados para o utilitário do MySQL:

```
$ mysql bancodedados < base.sql
```

3.2 Redirecionando a entrada com o sinal “ <<”

O sinal **“<<”** redireciona para a entrada e mantém a entrada aberta até que seja digitado algum caractere de EOF (fim de arquivo) como, por exemplo, CTRL+D

Este redirecionamento serve principalmente para marcar o fim de exibição de um bloco. Este é especialmente usado em conjunto com o comando `cat`, mas também tem outras aplicações. Por exemplo:

```
cat << final
este arquivo
será mostrado
até que a palavra final seja
localizada no inicio da linha
final
```

4. Redireccionamento de erro padrão

A mensagem de erro gerada por um comando é normalmente direcionada pela *shell* para a saída de erro padrão (STDERR), que é a mesma da saída padrão (STDOUT). A saída de erro padrão também pode ser redirecionada para um arquivo, utilizando o símbolo **“>”**. Uma vez que este símbolo também é utilizado para redirecionar a saído padrão, é necessário fazer uma distinção mais detalhada para evitar ambiguidade.

Os descritores de arquivos a seguir especificam a entrada padrão, saída padrão e saída de erro padrão:

```
0       Entrada padrão;
1       Saída padrão
2       Saída de erro padrão;
```

O descritor do arquivo deve ser colocado imediatamente antes dos caracteres de redirecionamento. Por exemplo, *1>* indica a saída padrão, enquanto *2>* indica a saída de erro padrão. Assim, o comando *mkdir temp 2> errfile* faz a shell direcionar qualquer mensagem de erro para o arquivo errfile.

Seguindo a mesma linha de exemplos anteriores, vemos o STDERR em acção quando tentamos listar um arquivo que não existe:

```
$ ls arquivonaoexistente.txt
ls: cannot access arquivonaoexistente.txt: No such file or directory

$ ls arquivonaoexistente.txt > /dev/null
ls: cannot access arquivonaoexistente.txt: No such file or directory
```

Apesar de no segundo comando *ls* redirecionarmos a saída para ser anulada (/dev/null), mesmo assim o comando retornou na tela a mensagem de que o arquivo não existe. Isto acontece porque esta é uma

mensagem de erro e por isso não é contemplada pelo sinal de “>” ou “1>”. Para redirecionar o STDERR, utilizamos “2>”. Corrigindo o exemplo anterior:

```
$ ls arquivonaoexistente 1> /dev/null 2> /dev/null
```

ou

```
$ ls arquivonaoexistente > /dev/null 2> /dev/null
```

Agora sim, tanto o STDOUT quanto o STDERR serão anulados, pois foram redirecionados para /dev/null.

As indicações da entrada padrão (0>) e saída padrão (1>) são necessárias apenas para evitar ambiguidade.

5. Redireccionador pipe “|”

Além dos redirecionadores anteriores, temos também o pipe, representado pelo caractere “|”. O pipe é um sinal responsável por passar a saída de um comando como a entrada de outro. Em outras palavras, ao se executar um comando, ao invés da saída deste ir para a tela ou para um arquivo, ele se torna a entrada de outro comando, funcionando de forma parecida com a utilização tanto do “>” quanto do “<”.

Utilizando um exemplo parecido com os anteriores, vamos ordenar um arquivo texto:

```
$ cat arquivo.txt | sort
```

O comando *cat*, ao invés de mostrar o conteúdo de arquivo.txt na tela, manda a saída para o comando *sort*, que ordena e envia para a tela. O mesmo comando, mas agora redirecionando o resultado para outro arquivo:

```
$ cat arquivo.txt | sort > arquivo_ordenado.txt
```

Ou então, se quisermos ver uma listagem detalhada de um diretório, podemos facilitar a visualização combinando a listagem e o comando *more*:

```
$ ls -lha /usr/bin | more
```

Ao listar todos os arquivos do diretório /usr/bin, o *ls* manda o resultado para o *more*. Com isso, podemos utilizar os recursos de paginação deste comando.

5.1 Diferença entre o "|" e o ">"

A principal diferença entre o "|" e o ">", é que o pipe envolve processamento entre comandos, ou seja, a saída de um comando é enviado a entrada do próximo e o ">" redireciona a saída de um comando para um arquivo/dispositivo.

Pode notar pelo exemplo acima (`ls -lha | more`) que ambos `ls` e `more` são comandos porque estão separados por um "|". Se um deles não existir ou estiver digitado incorretamente, será mostrada uma mensagem de erro.

Um resultado diferente seria obtido usando um `">"` no lugar do `"|"`; A saída do comando `ls -la > more` seria gravada num arquivo chamado `more`.

6. Redireccionamento múltiplo (comando tee)

O comando tee "divide" a saída de um comando e redireciona-a para múltiplos destinos: para um arquivo especificado e para a saída padrão. Este comando deve ser usado com o pipe "|".

```
comando | tee [iau] arquivo
```

Opções:

```
-a    Faz a saída ser anexada aos arquivos especificados, em vez de
      substituir seus conteúdos;

-i    Ignora o sinal de interrupção;

-u    Impede o uso de buffer;
```

Exemplo 1: `ls -la | tee listagem.txt`, a saída do comando será mostrada normalmente na tela e ao mesmo tempo gravada no arquivo `listagem.txt` Exemplos

Exemplo 2: Liste os arquivos que começam com a substring ‘arq’ no diretório corrente e guarde os arquivos encontrados no arquivo nomes:

```
$ ls arq* |  tee nomes
```

7. Exercícios

1. Usando caneta e papel escreva comandos GNU/Linux que permitem:

   a) Criar um arquivo chamado *list* que contenha uma listagem de todos os arquivos que começam com a letra d ou D e terminam com um número entre 1 e 5. O mesmo comando deve mostrar a saída na tela.

   b) Concatenar os arquivos *perm1* e *perm2* no arquivo *new.file*. Acrescente no final de *new.file* a seguinte frase: Usando desvio de saída padrão.

   c) Desvia para o arquivo *info*, as informações sobre cada usuário logado na sua máquina e adicione ao arquivo o número de usuários que a estão utilizando. Dica: use tee e who.

   d) Liste todos os arquivos e diretórios corrente que foram alterados no dia anterior, armazene a resposta no arquivo *resposta*. Conte o número de arquivos encontrados. Dica: use grep, tee e pipe.

2. Utilizando o nano do sistema GNU/Linux crie dois arquivos chamados *students1* e *students2* com os conteúdos abaixo:

students1	**students2**
Maria, Ricardo	Freitas, Pedro
Silva, João	Garcia, Maria
Souza, Thiago	Matos, Rosa

   a) Concatene os dois arquivos redirecionando a saída para o arquivo students.all.

   b) Remova o arquivo students.all e concatene os arquivos redirecionando a saída para arquivo students.all novamente, desta vez escreva errado o nome do arquivo students1. O que aconteceu? Quais as informações do arquivo students.all?

   c) Repita o exercício anterior, desta vez, redirecione as mensagens de erro para o arquivo students.erro. Qual o conteúdo de cada arquivo?

   d) Concatene os arquivos students1 e students2, salve os resultados no arquivostudents.temp e exiba simultaneamente os resultados no terminal.