

Sintaxe

Comando simples

Esta é a sintaxe de um único comando em bash:

```
comando arg1 arg2 ... argN
```

Os argumentos são separados por espaços, e podem ou não ser opcionais, dependendo do comando.

Caso o argumento tenha espaços no nome, é preciso colocá-lo entre aspas simples ou duplas para que seja interpretado como um único argumento:

```
rm "Arquivo com espacos no nome.txt"
```

Ou

```
rm 'Arquivo com espacos no nome.txt'
```

Redireção

Todo comando de terminal tem 3 arquivos especiais do qual pode ler ou escrever:

- `stdin` (0): entrada padrão
- `stdout` (1): saída padrão
- `stderr` (2): saída padrão de erro

(Os números entre parênteses são **file descriptors**: inteiros que representam arquivos abertos, e que têm valores fixos para esses 3 arquivos.)

Por padrão, o que se escreve em `stdout` ou `stderr` é impresso no terminal, e o que é lido de `stdin` é lido interativamente pelo terminal: o programa espera o usuário digitar as strings de entrada, até que se digite Ctrl+D.

Porém, é possível redirecionar qualquer um desses 3 arquivos, para que seu conteúdo seja lido de / escrito em um outro arquivo.

Para redirecionar a entrada, usamos `<`:

```
grep abcd 0< alfabeto.txt
```

Redireciona o arquivo de *file descriptor* 0 (ou seja, `stdin`), a entrada padrão) do comando `grep` para o arquivo `alfabeto.txt`.

É possível (e usual) omitir o *file descriptor* de `stdin`:

```
grep abcd < alfabeto.txt
```

Para redirecionar a saída, usa-se `>`:

```
echo abcdefg 1> alfabeto.txt
```

Mas, como no caso de `stdin`, é usual omitir o *file descriptor*:

```
echo abcdefg > alfabeto.txt
```

Se já existia um arquivo com esse nome, seu conteúdo original é sobrescrito:

```
echo abcdefg > alfabeto.txt
echo hijklmn > alfabeto.txt
cat alfabeto.txt
```

Produz a saída

```
hijklmn
```

Para que a redireção adicione o conteúdo ao fim do arquivo em vez de sobrescrevê-lo, usa-se `>>`:

```
echo abcdefg >> alfabeto.txt
echo hijklmn >> alfabeto.txt
cat alfabeto.txt
```

Produz a saída

```
abcdefg
hijklmn
```

Podemos redirecionar tanto a saída quanto a entrada:

```
grep abcd < alfabeto.txt > saida.txt
```

Da maneira análoga a `stdout`, podemos redirecionar a saída padrão de erro (`stderr`), mas para isso é preciso explicitar o *file descriptor* antes do sinal de redireção:

```
./a.out 2>log.txt
```

O uso mais comum da redireção da saída de erro é com `/dev/null` - o buraco negro na forma de arquivo: toda tentativa de leitura ou escrita nesse arquivo falha. Por isso, `2>/dev/null` é usado para omitir as mensagens de erro de um comando.

Podemos redirecionar a saída de erro para a saída padrão:

```
./a.out 2>&1
```

Note o uso de `&` antes do `1`: se não usássemos isso, a saída padrão de erro seria redirecionada para um arquivo cujo nome é realmente `1`.

Pipelines

Pipelines são a forma mais comum de encadear comandos em Unix. Uma pipeline é denotada pela barra vertical `|`, e transforma a saída padrão do primeiro comando na entrada padrão do próximo:

```
comando1 | comando2 | comando3 | ... | comandoN
```

Por exemplo, se queremos imprimir apenas a primeira coluna de um arquivo csv (que tem várias entradas em cada linha separadas por vírgulas),

```
cat dados.csv | cut -d',' -f1
```

Se queremos imprimir a mesma coisa, mas apenas das linhas 2 até 5 do arquivo original,

```
cat dados.csv | sed -n '2,10p' | cut -d',' -f1
```

E assim por diante.

Por padrão, as pipelines não passam a saída de erro para o outro programa. Para fazer isso, é necessário usar uma redireção da saída de erro para a saída padrão (`2>&1`):

```
strace a.out 2>&1 | grep SIGSEGV
```

Porém, é possível usar `| &`, que é sinônimo de `2>&1 |`:

```
strace a.out | & grep SIGSEGV
```

As pipelines refletem bem a filosofia Unix: ter vários programas pequenos que realizam uma única tarefa simples, mas que podem ser combinados de maneira flexível para realizar uma tarefa complexa - que os criadores dos programas individuais não necessariamente previram.

Encadeamento incondicional

TODO (`;`)

Encadeamento condicional

TODO (`&&` e `||`)

Revision #8

Created Sun, Mar 17, 2019 11:34 PM by Luana

Updated Fri, Apr 5, 2019 3:11 AM by Luana