Linha de comando (bash)

- Comandos úteis
- Sintaxe

Comandos úteis

Comandos úteis do terminal, e a forma como normalmente os usamos.
Esta página não é, e não se propõe a ser, uma documentação exaustiva para esses comandos; consulte as páginas de manual para essa finalidade.

Ajuda

man (MANual)

Mostra a página de manual de um comando. É provavelmente o comando mais útil de todos.

whatis

Mostra uma descrição curta de um comando, que equivale à primeira linha da página de manual.

Comandos básicos

cd (Change Directory)

Muda o diretório atual para o argumento passado, ou para $HOME se nenhum argumento for passado.

ls (LiSt)

Lista os arquivos do diretório atual (se usado sem argumentos), ou dos diretórios passados como argumentos (caso contrário).
A opção -l pode ser usada para imprimir metadados úteis, como permissões e timestamps, e para ver aonde links simbólicos apontam.

echo

Imprime de volta o que lhe é passado como argumento. (`echo` em inglês significa "eco".)

Um comando como esse pode parecer inútil a princípio, mas `echo` tem vários usos. Por exemplo:

- descobrir o valor de uma varáivel de ambiente:

```
$ echo $PATH
```

- criar um arquivo pequeno rapidamente, sem ter que abrir um editor de texto:

```
echo "0O 1IL" > testfont.txt
```

- passar input binário para um programa, com a opção `-e`:

```
echo -e "AAAAAAAA\xff\xff\xff\xffBBBBBBBB" | ./a.out
```

mv (MoVe)

Move ou renomeia arquivos, dependendo de seus argumentos. Por exemplo:

```
$ mv foo bar
```

Se existe uma pasta `bar`, então o arquivo `foo` é colocado nessa pasta, de modo que o caminho para o arquivo passa a ser `bar/foo`. Do contrário, o arquivo `foo` passa a se chamar `bar`.
Note que, se existia um arquivo comum (não um diretório) chamado `bar` antes de esse comando ser executado, o conteúdo do arquivo original é perdido. Para evitar isso, use a opção `-n`.

cp (CoPy)

Copia um arquivo.

rm (ReMove)

Remove um ou mais arquivos. Pode remover também um diretório e todos os arquivos nele com a opção `-R`. (Cuidado com essa opção!)
Note que esse comando

- não envia o arquivo para uma "lixeira" como no Windows: o arquivo não pode ser acessado depois de removido.
- não remove nem sobresecreve o conteúdo do arquivo no disco. (O arquivo não pode ser

acessado porque não se sabe onde ele está, mas poderia teoricamente ser recuperado.) Se você quiser que o conteúdo do arquivo seja realmente apagado, veja 'shred', abaixo.

cat (conCATenate)

Imprime todos os arquivos passados como argumentos, na ordem em que foram passados. (os arquivos são concatenados, ou seja, impressos como se fossem um só.)
Geralmente usado em conjunto com comandos que filtram o input, como `grep` ou `sed`.

mkdir (MaKe DIRectory)

Cria um diretório.

rmdir (ReMove DIRectory)

Remove um diretório, contanto que ele esteja vazio.
Se você deseja remover um diretório e todos os arquivos nele, use `rm -R` em vez disso. (Cuidado ao usar isso!)

Leitores de texto interativos

less

Permite ler um arquivo ou o output de um comando scrollando para cima e para baixo, conforme o usuário queira. Digite `h` para mostrar o menu de ajuda e `q` para sair.

Editores de texto interativos

nano

Editor de texto simples e intuitivo. Os atalhos do editor são mostrados na tela.

vim

Editor de texto muito mais sofisticado, mas com uma interface contra-intuitiva. Caso não esteja instalado, é possível usar o `vi`.

Filtragem/manipulação de input

grep

Filtra o que lhe é passado como input, mostrando apenas as linhas que batem com a expressão regular passada como argumento.
É recomendável sempre usar a opção `-E` para obter expressões regulares "extendidas" (que são o padrão em quase todos os lugares em que se usam regex. Sem essa opção, é preciso usar `\(` e `\)` em vez de `(` e `)`, por exemplo.)
`grep` pode operar recursivamente em todos os arquivos dentro de uma pasta, e dentro de suas subpastas, ..., com a opção `-R`. Por exemplo, para ver todos os comentários com FIXME ou TODO num diretório:

```
$ grep -ER "FIXME|TODO" /usr/include/x86_64-linux-gnu/
/usr/include/x86_64-linux-gnu/sys/gmon.h:  * programs;  will 1<<20 be adequate for long?  FIXME
/usr/include/x86_64-linux-gnu/bits/stab.def:    or something like that.  FIXME.  I have
assigned the values at random
/usr/include/x86_64-linux-gnu/bits/libio.h:  it points to _buf->Gbase()+_pos.  FIXME comment */
/usr/include/x86_64-linux-gnu/bits/libio.h:  /* if _pos < 0, it points to _buf->eBptr()+_pos.
FIXME comment */
```

É possível também ver apenas os arquivos que contém a expressão regular, sem imprimir a linha em que ela está, com a opção `-l`:

```
$ grep -ERl "FIXME|TODO" /usr/include/x86_64-linux-gnu/
/usr/include/x86_64-linux-gnu/sys/gmon.h
/usr/include/x86_64-linux-gnu/bits/stab.def
/usr/include/x86_64-linux-gnu/bits/libio.h
```

Com isso, o output fica menos poluído, e podemos abrir cada arquivo seperadamente (com `less`, por exemplo) para entender os comentários. Mas ainda é possível usar o `grep` para isso: as opções `-A` (after) e `-B` (before) podem ser usadas, respectivamente, para imprimir um certo

número de linhas antes e depois do match da expressão regular:

```
$ grep -E -A2 -B2 "FIXME|TODO" /usr/include/x86_64-linux-gnu/bits/stab.def

/* These STAB's are used on Gould systems for Non-Base register symbols
   or something like that.  FIXME.  I have assigned the values at random
   since I don't have a Gould here.  Fixups from Gould folk welcome... */
__define_stab (N_NBTEXT, 0xF0, "NBTEXT")
```

E isso é suficiente para ler todo o comentário em que há o FIXME.

head

imprime apenas o começo de um arquivo (10 linhas por padrão, ou o número especificado com a opção `-n`)

tail

imprime apenas o final de um arquivo (10 linhas por padrão, ou o número especificado com a opção `-n`).
Esse comando também permite imprimir um arquivo à medida em que dados são adicionados para ele. Por exemplo, imagine que você tem um programa que imprime muita coisa na saída padrão (stdout), mas que imprime na saída padrão de erro (stderr) mensagens sobre um erro que você quer depurar. Para que as mensagens não se misturem, você pode ser redirecionar stderr para um arquivo de log:

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

Mas fazendo só isso você não poderia ver as mensagens de erro enquanto o programa está rodando. Para isso, você pode rodar, em outro terminal

```
$ tail -f log.txt
```

cut

Recorta todas as linhas do input, deixando apenas as colunas espeficadas pela opção `-f`. Sintaxe:

- `N`: coluna N, apenas

- `N-M`: da coluna N até a M
- `N-`: da coluna N em diante
- `-M`: até a coluna M

Use a opção `-d` para especificar o delimitador. Por exemplo:

```
$ cat dados.csv
time;gFx;gFy;gFz;TgF
0.004000;-0.1482;0.7379;0.6228;0.977
0.004000;-0.1634;0.7574;0.6683;1.023
0.011000;-0.1678;0.7716;0.6942;1.051
0.013000;-0.1599;0.7765;0.6995;1.057

$ cat dados.csv |cut -d';' -f2-3
gFx;gFy
-0.1482;0.7379
-0.1634;0.7574
-0.1678;0.7716
-0.1599;0.7765
```

```
$ find /usr/lib/ -iname '*gmp*'
/usr/lib/x86_64-linux-gnu/libgmpxx.a
/usr/lib/x86_64-linux-gnu/libgmp.a
/usr/lib/x86_64-linux-gnu/libgmpxx.so.4.5.2
/usr/lib/x86_64-linux-gnu/libgmp.so.10
/usr/lib/x86_64-linux-gnu/libgmp.so.10.3.2
/usr/lib/x86_64-linux-gnu/libgmpxx.so.4
/usr/lib/x86_64-linux-gnu/libgmp.so
/usr/lib/x86_64-linux-gnu/libgmpxx.so
$ find /usr/lib/ -iname '*gmp*' |cut -d'/' -f4-
x86_64-linux-gnu/libgmpxx.a
x86_64-linux-gnu/libgmp.a
x86_64-linux-gnu/libgmpxx.so.4.5.2
x86_64-linux-gnu/libgmp.so.10
x86_64-linux-gnu/libgmp.so.10.3.2
x86_64-linux-gnu/libgmpxx.so.4
x86_64-linux-gnu/libgmp.so
x86_64-linux-gnu/libgmpxx.so
```

```

```

sed (Stream EDitor)

`sed` é um editor de texto não-interativo, que pode modificar o input com uma série de comandos. Quase todos os comandos podem ser precedidos por linhas ou ranges de linhas. Esses comandos serão executados apenas na range especificada.

- `N`: linha `N`, apenas
- `N,M`: das linhas `N` até (inclusive) `M`
- `N,+M`: das linhas `N` até (inclusive) `N+M`

Em que `N` e `M` podem ser números naturais ou `$`, que denota a última linha.

Todos os exemplos a seguir usam o seguinte arquivo (os números de linha não fazem parte do arquivo):

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.
```

1. substituir todas as ocorrências de uma expressão regular pela mesma string (`s`, substitute)

```
$ cat lipsum.txt | sed 's/[uU]t/AH/'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
AH labore et dolore magna aliqua.
AH enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi AH aliquip ex ea commodo consequat.

$ cat lipsum.txt | sed '4 s/[uU]t/AH/'
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
AH labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.

$ cat lipsum.txt | sed -E 's/([uU]t)/\1\1\1/'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ututut labore et dolore magna aliqua.
UtUtUt enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ututut aliquip ex ea commodo consequat.
```

A sintaxe do comando é `s/<regex>/<substituto>/`.

2. imprimir apenas algumas linhas do input (`p`)

```
$ cat lipsum.txt | sed -n '2 p'
consectetur adipiscing elit,

$ cat lipsum.txt | sed -n '1,+3 p'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Note, nos dois casos, o uso da opção `-n`: ela faz com que apenas as linhas impressas por um comando como o `p` apareçam no output. Se não usarmos essa opção, as linhas impressas com `p` aparecem duas vezes:

```
$ cat lipsum.txt | sed '1,+3 p'
Lorem ipsum dolor sit amet,
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
consectetur adipiscing elit,
```

```
sed do eiusmod tempor incididunt
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
ut labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.
```

3. deletar algumas linhas do input (`d`)

```
$ cat lipsum.txt | sed '3 d'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
ut labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.

$ cat lipsum.txt | sed '3,+2 d'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.
```

4. trocar uma ou mais linhas inteiras por uma string (`c` , change)

```
$ cat lipsum.txt | sed '4,6 c WE COME FROM THE LAND OF THE ICE AND SNOW'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
WE COME FROM THE LAND OF THE ICE AND SNOW
nisi ut aliquip ex ea commodo consequat.
```

5. inserir linhas antes ou depois de uma dada linha (`i` , `a`)

```
$ cat lipsum.txt | sed '4 i WE COME FROM THE LAND OF THE ICE AND SNOW'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
```

```
WE COME FROM THE LAND OF THE ICE AND SNOW
ut labore et dolore magna aliqua.
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.

$ cat lipsum.txt | sed '4 a WE COME FROM THE LAND OF THE ICE AND SNOW'
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
WE COME FROM THE LAND OF THE ICE AND SNOW
Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat.
```

O comando `i` (insert) coloca o input antes da linha 4, enquanto o comando `a` (append) coloca o input depois.

awk (Aho-Weinberger-Kernighan)

tr (TRanslate)

O comando pode ser usado de duas maneiras.
A primeira é a "tradução", em que duas strings de mesmo tamanho são passadas como argumento, que substiutui as letras do input: todas as ocorrências no input da n-ésima letra do primeiro argumento são trocada pela n-ésima letra do segundo argumento. Por exemplo:

```
$ echo jujuba |  tr abju leaz
azazel
```

A tradução pode também eliminar ocorrências repetidas de um caractere no input, com a opção `-s` (squeeze). Isso é útil para eliminar excesso de espaços em branco:

```
$ echo "varios       espacos    em       branco" |  tr -s " " " "
varios espacos em branco
```

A segunda maneira é a deleção, com a opção `-d`, em que uma única string é passada como

argumento, e todas as ocorrências de todos os caracteres dessa string são deletados no input:

```
$ echo "asdasdasdasdadasdasdasaoadsdasdasdasdaiasdasdasdasdasda" | tr -d "asd"
oi
```

Recursos do bash

pushd (PUSH Directory)

Altera o diretório atual para o argumento passado, e o coloca no topo da pilha.
Esse comando tem o mesmo efeito do `cd`, mas permite que se retorne para o diretório em que se estava usando `popd` posteriormente.
É possível rotacionar a pilha de diretórios usando `pushd +n`, em que n é um inteiro.

popd (POP Directory)

Remove o diretório do topo da pilha. O diretório atual passa a ser o que estava imediatamente antes do topo.

dirs (DIRectorieS)

Imprime a pilha de diretórios.

Miscelânia

ln (LiNk)

shred

Sobrescreve o conteúdo de um arquivo com bytes aleatórios. A opção `-u` pode ser usada para remover o arquivo posteriormente.

dd

Apelidado de "Disk Destroyer", esse comando copia uma certa quantidade de bytes de um arquivo para outro. (O nome original é "convert and copy", mas como `cc` já estava em uso como "C

compiler", trocaram a letra `c` por `d`.)
Um bom uso desse comando é sobrescrever todo o conteúdo de um pen-drive ou disco rigido (daí o apelido). Assumindo que esse disco seja /dev/sdb, então

```
$ dd if=/dev/zero of=/dev/sdb
```

copia bytes do arquivo especial /dev/zero (que sempre gera bytes que valem 0, como o nome indica) para /dev/sdb, até que o arquivo acabe (ou seja, até que todo o disco seja sobrescrito com zeros).

du (Disk Usage)

Estima o tamanho de um arquivo no disco. Use a opção '-h' para obter valores "human readable"

pwd (Print Working Directory)

Imprime o diretório atual. Esse comando fazia mais sentido antigamente, quando o prompt do shell não mostrava esse diretório constantemente.

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 `||`)