Apostila de Linux

Josiney de Souza, Leonardo Gomes

April 20, 2013

Contents

1 Introdução ao Linux

Esta seção é um tutorial básico de Linux para quem tem pouca ou nenhuma familiaridade com o sistema. O Linux é um sistema operacional livre baseado no antigo UNIX, desenvolvido nos anos 60. Nos laboratórios do DInf, a versão do Linux (também chamada de distribuição Linux) adotada é o Ubuntu, porém as técnicas e comandos aqui apresentados servem para quase a totalidade das distribuições.

1.1 *Shell* e *Terminal*

Atualmente, existem excelentes interfaces gráficas muito amigáveis que permitem realizar todo o tipo de operação em Linux, mas ainda assim é muito importante ter domínio sobre o ambiente CLI (*Command Line Interface*) que é uma interface na qual comandos e respostas são transmitidos em forma de texto. *Shell* é a classe de programas responsáveis por implementar a linguagem desta interface (Exemplo: bash, dash, csh, zsh), e *terminal* é a classe de programas que trazem uma interface gráfica ao *shell*. Dentre os programas *terminal* alguns dos mais conhecidos são: gnome-terminal, xterm, konsole.

"ls", "man" (ver seção 1.2), "fpc", "alomamae" (após compilado) são exemplos de comandos que são executados em *shell*. Normalmente, a parte que o usuário vê do *shell* é o *prompt* de comandos, representado por "$ >".

Para executar um terminal dentro dos laboratórios do DInf basta navegar nos menus existentes na parte superior da tela e dentro de "Aplicações" estará o programa chamado terminal.

Para facilitar a utilização do terminal, repare que as setas cima e baixo no teclado podem ser utilizadas para navegar entre os últimos comandos digitados; e, ao pressionar **TAB** duas vezes consecutivas, o terminal muitas vezes pode lhe oferecer opções de como auto-completar nomes de comandos e nomes de arquivos.

1.2 *Man page*

Muitos programas em Linux possuem diversas opções para que o usuário possa realizar seu trabalho. Nem sempre é fácil ou mesmo possível memorizar cada uma das opções existentes dos mais variados programas disponíveis. Devido a isso, os criadores dos programas desenvolvem manuais virtuais para auxiliar no uso de suas aplicações.

A *man page* ou simplesmente *man* é um recurso valioso para se descobrir a forma correta de uso (sintaxe) de programas, suas opções ou entender o que um programa faz. Utilize as setas para navegar no manual e a tecla 'q' para sair. Para executar este programa basta digitar *man* seguido do nome do programa que se deseja inspecionar como no exemplo abaixo:

```
$> man ls

$> man mkdir
```

1.3 Navegação em diretórios

Para navegar entre diretórios em ambiente *shell* é necessário entender dois tipos de endereçamento, os caminhos absolutos e caminhos relativos.

Caminhos absolutos são a localização de diretórios e arquivos a partir da raiz do sistema "/" (barra, *root*). Por exemplo: /home/prof/professor/disciplina/pdfs

Caminhos relativos dependem do diretório corrente. Por exemplo: Assumindo que a cadeia de diretórios (/home/prof/professor/disciplina/pdfs) E assumindo que o usuário, se o usuário esteja no diretório /home/prof/professor, para ele chegar ao diretório "pdfs" bastaria se locomover para a seguinte sequência de caminhos: "disciplina/pdfs".

Comparado ao caminho absoluto, o caminho relativo é bem mais compacta; contudo, depende necessariamente do ponto de partida do usuário. Já o caminho absoluto, pode ser utilizado independente de onde o usuário esteja, pois sua referência é global, enquanto no caminho relativo a referência é local.

1.3.1 Locomoção

O Linux dispõe de ferramentas para locomoção entre diretórios. O comando que faz essa tarefa é o comando "cd" (*change directory*). Ele aceita tanto caminhos absolutos quanto caminhos relativos. Do exemplo da subseções acima, o comando cd seria o seguinte para caminhos absolutos:

```
$> cd /home/prof/professor/disciplina/pdfs
```

Após confirmar o comando com ENTER, é possível notar que o *prompt* de comando muda para:

```
/home/prof/professor/disciplina/pdfs$>
```

Essa mudança é para mostrar que o diretório de origem foi trocado pelo novo diretório de destino. Isso se reflete no *prompt*, para informar o usuário sobre sua atual localização.

Dado isso, o exemplo para caminhos relativos ficaria da seguinte forma:

```
/home/prof/professor$> cd disciplina/pdfs
/home/prof/professor/disciplina/pdfs$>
```

Da mesma forma que é possível ir adiante com caminhos relativos, também é possível ir para trás na sequência. Isso é feito com o caminho especial ".." (ponto-ponto). Ele é como o ".." do *prompt* de comandos do MS-DOS e leva a um diretório anterior na sequência de caminhos. Exemplo:

```
/home/prof/professor$> cd ..
/home/prof$>
```

É possível usar esse comando para construir um caminho relativo maior:

```
/home/prof/professor$> cd ../..
/home$>
```

Assim como o ".." representa o diretório anterior, o '.' (ponto) representa o diretório corrente[1].

1.3.2 Diretório HOME

Em qualquer distribuição Linux, por ser um sistema multiusuário, cada usuário possui seu *login* e seu diretório HOME. Diretório HOME é a área privada de cada usuário, onde seus arquivos ficam armazenados.

É possível endereçar também esse diretório durante a navegação em *shell*. Se o usuário desejar se mover para seu diretório HOME, ele pode realizar qualquer um dos seguintes comandos:

```
$> cd $HOME

$> cd ~

$> cd ~usuario

$> cd
```

A primeira opção é uma variável em *shell* que guarda o diretório HOME de cada usuário. Essa variável é definida cada vez que um usuário faz *login* no sistema e cada usuário visualiza essa variável com um valor diferente, pois se trata da sua área particular. É possível imprimir o valor dessa variável, a título de curiosidade:

```
$> echo $HOME
/home/bcc/usuario
```

A segunda opção é um alias, um apelido, um *link* para a primeira opção. Se for impresso o valor de "∼" (til), será idêntico ao valor de "$HOME".

A terceira opção é mais genérica que a segunda, pois ela usa o nome do usuário para ir ao seu diretório HOME. Da mesma forma que é possível ir a seu próprio diretório HOME, se houver permissões, o usuário pode ir a

[1]Parecido com o significado de diretório corrente, arquivos ocultos (normalmente arquivos de configuração) são identificados por um '.' (ponto) no início do nome do arquivo. Veja mais detalhes consultando a *man page* do comando "ls", lendo sobre o parâmetro "-a"

qualquer outro diretório HOME existente; basta digitar o nome do outro usuário que se deseja visualizar a HOME em vez do seu próprio.

A quarta opção é um atalho do comando "cd" para voltar ao diretório HOME do usuário. É possível estar em qualquer localização no sistema, se o comando "cd" for dado sem mais argumentos, o usuário se moverá para seu diretório HOME.

Como uma curiosidade, há também o diretório "-" (menos), que nada mais é do que a última localização visitada pelo usuário fora a atual. Caso o usuário estivesse na raiz do sistema, mudasse para seu diretório HOME e desejasse voltar para a raiz do sistema; ele poderia se mover para a raiz através desse diretório.

1.4 Permissões de acesso

O sistema de arquivos no Linux trabalha com conjunto de permissões. Para cada arquivo ou diretório existente, está associado um conjunto de permissões de acesso. Elas garantem a posse ou propriedade sobre arquivos ao seu devido possuidor.

A posse de arquivos pode ser vista em três níveis: usuário dono do arquivo (*user*), grupo que tem privilégios sobre o arquivo (*group*) e outros usuários (*others*) que não é nem o dono nem mesmo são do grupo de possui privilégios. A seguir, está o resultado que o programa "ls" mostra após sua execução:

```
$> ls -l
-rw-r--r-- 1 josineys ppginf 0 2012-03-06 14:59 meu_codigo.pas
drwxr-xr-x 2 josineys ppginf 6 2012-03-06 15:18 exercicios
$>
```

Os dez primeiros caracteres de cada linha acima estão representando as permissões de acesso, na primeira linha temos um arquivo denominado "meu_codigo.pas" e na segunda linha um diretório chamado "exercicios". O primeiro caractere está reservado para denotar se o arquivo em questão é ou não um diretório 'd' para diretório e '-' para arquivo.

Os nove caracteres que se seguem podem ser divididos em três grupos de três elementos cada um. Cada um desses grupos podem ser povoados com as letras 'r', 'w' ou 'x'; cada uma dessas ocupando as posições um, dois ou três dentro do grupo, respectivamente.

O primeiro grupo está reservado para mostrar as permissões do usuário dono do arquivo; o segundo, para mostrar as permissões do grupo que possui privilégios sobre o arquivo; o terceiro, para os demais usuários (*others*).

Nos exemplos acima, tanto o arquivo quanto o diretório pertencem ao usuário "josineys". Este usuário pertence ao grupo de usuários "ppginf", mesmo grupo que tem privilégios sobre esse arquivo. Já a informação de

outros usuários não está representada na saída desse comando; mas, por exclusão, os outros usuários que podem trabalhar com esse arquivo são aqueles que não são nem o dono ("josineys") nem os que estão no grupo "ppginf".

Essas permissões mostradas podem ser do tipo leitura (*read*), escrita (*write*) ou de execução (*execute*). Representam a forma como cada um desses usuários podem operar sobre os arquivos ou diretórios existentes. O Linux considera diretórios como sendo arquivos especiais, por isso há uma pequena distinção de significado dessas permissões entre arquivos comuns e diretórios.

Para os arquivos comuns, as permissões 'r', 'w' e 'x' garantem que o usuário que o acessa pode ler o conteúdo do arquivo, fazer modificações sobre o arquivo ou executar o arquivo (se for um programa executável - semelhante ao .exe do Windows) respectivamente; de acordo com o grupo de permissões que o usuário pertença.

Já para diretórios, 'r', 'w' e 'x' permitem que usuários possam ler (listar, ver) o conteúdo do diretório, fazer modificações (apagar ou criar novos arquivos ou subdiretórios) ou acessar (entrar em) um certo diretório; respectivamente.

Caso alguma dessas permissões esteja com um '-', significa que aquela determinada permissão que deveria estar ali presente não foi atribuída, ou seja, o usuário não tem a capacidade de se valer do que aquela permissão lhe concederia.

Resumindo:

	arquivos	diretórios
'r' (*read*)	ler, abrir, copiar	listar, ver o conteúdo
'w' (*write*)	alterar o conteúdo	criar, apagar ou mover arquivos
'x' (*execute*)	executar	acessar, entrar
'-'	nega a permissão	nega a permissão
'd' (*directory*)		denota um diretório

Há outros estados que essas permissões podem assumir, como 'x', 'X', 's', 't'. Para ler sobre eles, acesse a *man page* (ver seção 1.2) do comando "chmod".

1.4.1 chmod

O comando "chmod" modifica as permissões de um arquivo ou diretório baseado na seguinte lista de opções:

- **u**, significa usuário;
- **g**, significa grupo;
- **o**, significa outros;
- **a**, significa todos os usuários;

- **r**, significa leitura;
- **w**, significa escrita;
- **x**, significa execução;
- +, significa adicionar permissão;
- **-**, significa remover permissão.

o comando é executado como no exemplo:

```
$> chmod go-rx
```

Remove a permissão de leitura e execução aos usuários do grupo e outros.

```
$> chmod a+rwx
```

Dá permissão de leitura, execução e escrita para todos os usuários.

1.5 Executando programas

Para se executar programas, é necessário que:

- Tenha sido compilado previamente;
- Tenha permissão de execução.

Programas como "ls" e "fpc" foram compilados anteriormente e estão disponíveis no sistema para uso. A forma de executá-los é através de um *shell* (ver seção 1.1), digitando o nome do arquivo executável no *prompt* de comandos.

Outros comandos precisam ainda ser compilados antes de seu uso. Quando se escreve um código-fonte, somente depois de se executar um conversor de linguagem de alto nível (instruções em uma linguagem de programação de alto nível - por exemplo, Pascal) para linguagem de máquina é que se terá um arquivo executável que realiza o mesmo trabalho do escrito no código-fonte.

Também é preciso que um programa executável possua permissão de execução (ver seção 1.4). Confirmado esses requisitos, ou se executa os programas compilados anteriormente simplesmente digitando seu nome no *shell* ou digitando o nome do arquivo compilado adicionado de './' (ponto-barra), indicando que se trata de um arquivo executável localizado no diretório local (ver seção 1.3.1).

1.6 Redirecionando entradas e saídas

Entradas e saídas (do inglês *input* e *output*) são as formas com que o sistema operacional faz a comunicação entre programas, comandos, processos e usuário. A entrada mais conhecida é o teclado, enquanto a saída mais conhecida é a tela do computador. Por isso, são as entrada e saída padrões, respectivamente.

Contudo, essas entradas e saídas podem ser redirecionadas para outros locais. Um programa executado na linha de comandos provavelmente produzirá resultados que será visíveis na tela, porém pode-se querer que a saída desse programa seja enviada diretamente para um arquivo do disco rígido.

De forma semelhante, um arquivo do disco pode ser usado como entrada de comandos. Em vez do usuário digitar todos os valores para cada uma das variáveis do programa, ele pode criar um arquivo com os valores desejados e usar esse arquivo no lugar de sempre digitar toda a sequência de valores.

Em Linux, a entrada padrão é definida para o descritor de arquivo[2] 0 (zero); a saída padrão, para o descritor de arquivos 1 (um) e a saída de erros, para o descritor de arquivos 2. São esses os valores utilizados no momento de se redirecionar entradas e saídas.

1.6.1 Redirecionamento de saída

Começando pela saída padrão, quando se quer redirecioná-la, usa-se o descritor de arquivos 1. No *shell* (ver seção 1.1), o descritor de arquivos 1 é usado junto com o '>' (maior). Dessa forma, para se transferir a saída para um arquivo, faz-se:

```
$> ls 1> saida_ls.txt
```

É equivalente a:

```
$> ls > saida_ls.txt
```

Nos casos acima, a saída do comando "ls" não será apresentada na tela, mas sim será gravada no arquivo "saida_ls.txt". Vale mencionar que esse arquivo não precisa existir previamente, quando o redirecionamento for feito, ele será criado (se não existir) ou será sobrescrito (caso já exista). Isso é chamado de redirecionamento destrutivo.

Se deseja-se um redirecionamento não destrutivo, ou melhor, um redirecionamento que não apague o conteúdo já existente de um arquivo, nesse caso é preciso usar um segundo '>' (maior). Para os exemplos acima, o redirecionamento não destrutivo ficaria:

```
$> ls 1>> saida_ls.txt
```

[2] `http://pt.wikipedia.org/wiki/Descritor_de_arquivo`

Ou:

```
$> ls >> saida_ls.txt
```

O mesmo pode ser feito para a saída de erros, cujo descritor de arquivos é o 2. Essa saída cuida de quaisquer erros que possam acontecer na execução de um comando. Por exemplo, tentar listar um arquivo que não exista, produz um erro que será enviado para a tela:

```
$> ls arquivo-inexistente.txt
ls: impossível acessar arquivo-inexistente.txt: Arquivo ou diretório não encontrado
$>
```

Esse não é o comportamento esperado para o comando ls, trata-se de um erro que foi informado pelo sistema. Caso o usuário queira, ele pode redirecionar essa saída de erro para um arquivo:

```
$> ls arquivo-inexistente.txt 2> saida_erro_ls.txt
$>
```

Dessa vez, o erro não foi mostrado na tela, mas foi guardado no arquivo "saida_erro_ls.txt". Esse redirecionamento com "2>" é destrutivo, assim como o "1>". Caso deseje fazê-lo não destrutivo, deve-se usar o "2>>".

1.6.2 Redirecionamento de entrada

Finalmente, para se redirecionar a entrada padrão (teclado), usa-se o caminho contrário do redirecionamento. Agora, o símbolo usado é "<". Seja o seguinte programa em Pascal:

```
program le_e_imprime;

var vetor3nums : array [1..3] of integer;
    i: integer;

begin
    (* Le tres numeros e os imprime em ordem inversa *)
    for i := 1 to 3 do
        read(vetor3nums[i]);

    for i := 3 downto 1 do
        write(vetor3nums[i], ' ');
    writeln;
end.
```

Após compilado e posto para executar, o programa esperará que três números sejam lidos da entrada padrão, só depois disso os três números são impressos na saída padrão em ordem inversa.

```
$> ./le_e_imprime
1 2 3
3 2 1
$>
```

Enquanto trata-se da leitura de apenas três números, não é muito esforço sempre digitá-los enquanto se testa o programa desenvolvido. Contudo, se o programa precisasse ler 50 números e então imprimi-los em ordem inversa a digitada, testar esse programa sempre digitando os números torna-se uma tarefa cansativa.

Para isso, é possível automatizar o processo, escrevendo os números em um arquivo texto e usando-o como entrada para o programa:

```
$> cat entrada_numerica.txt
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50
$>
```

Seja o novo programa em Pascal, alterado para ler 50 números e imprimi-los em ordem inversa:

```
program le_e_imprime;

var vetor50nums : array [1..50] of integer;
    i: integer;

begin
    (* Le 50 numeros e os imprime em ordem inversa *)
    for i := 1 to 50 do
        read(vetor50nums[i]);

    for i := 50 downto 1 do
        write(vetor50nums[i], ' ');
    writeln;
end.
```

Após compilado, usa-se então o arquivo texto com todos os números a serem testados:

```
$> ./le_e_imprime < entrada_numerica.txt
50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
$>
```

1.6.3 Redirecionamento de ambos

É possível combinar ambos os redirecionamentos em um mesmo comando:

```
$> ./le_e_imprime < entrada_numerica.txt > saida_numerica.txt
$>
```

Neste exemplo, o conteúdo do arquivo chamado "entrada_numerica.txt" (tal qual descrito anteriormente) é redirecionado para a entrada do programa le_e_imprime feito em Pascal. Por sua vez, a saída do programa compilado que antes era mostrada na tela, agora é redirecionada para o arquivo chamado "saida_numerica.txt".

1.6.4 *Pipe*

É possível encadear comandos Linux utilizando o sinal da barra vertical, também chamado de *pipe*, |. Por exemplo:

```
$> ls | sort
$>
```

Esse comando irá redirecionar a saída do programa "ls" para a entrada do programa "sort". É equivalente a fazer em sequência:

```
$> ls > lista.txt
$> sort < lista.txt
$>
```

1.7 Metacaracteres

Os metacaracteres são caracteres reservados dotados de alguma função especial; eles aumentam muito a versatilidade dos comandos Linux.

1.7.1 Asterisco

O '*' (asterisco) simboliza qualquer número variado de caracteres. Por exemplo:

```
$>ls lista*
$>
```

O comando acima executa o programa “ls” em todos os arquivos e pastas do diretório corrente cujo nome inicie por “lista” incluindo um arquivo chamado apenas “lista”, caso ele exista. “*lista” e “l*a” são outros exemplos válidos de uso do asterisco.

1.7.2 Interrogação

O ‘?’ (ponto de interrogação) simboliza qualquer um (e exatamente um) carácter. por exemplo:

```
$>ls ?ala
$>
```

Este comando irá executar o programa em “ls” em arquivos com nomes como “cala”, “bala”, “fala”, “mala”. Mas não em um cujo nome seja “ala” ou “coala”.

1.7.3 Colchetes

Os símbolos ‘[’ ‘]’ (abre e fecha colchetes) em conjunto simbolizam qualquer um (e exatamente um) carácter que esteja dentro do conjunto interno ao colchetes, por exemplo:

```
$>ls lista[123].txt
$>
```

Este comando irá executar o comando “ls” em um arquivo que se chame “lista1.txt” ou “lista2.txt” ou “lista3.txt”.

```
$>rm notas_da_turma_[a-f].txt
$>
```

Este comando executa o comando “rm” em todos os arquivos chamados “notas_da_turma_” seguido de qualquer carácter que esteja entre ‘a’ e ‘f’ (a,b,c,d,e,f) seguido de “.txt”.

1.7.4 Chaves

Os símbolos ’’ ’’(abre e fecha chaves) em conjunto simbolizam que exatamente a sequência de caracteres de um elemento que esteja dentro do conjunto interno às chaves deve coincidir. Aqui, cada um dos elementos do conjunto é separado por ‘,’ (vírgula). Por exemplo:

```
$> ls lista{1,2,3}.txt
lista1.txt   lista2.txt   lista3.txt
```

O exemplo mostra que o comando "ls" vai procurar por exatamente cada um dos elementos '1', '2' e '3'; adicionados do prefixo "lista" e do sufixo ".txt".

```
$> ls lista{1,2,3,4-6}.txt
lista1.txt   lista2.txt   lista3.txt   lista4-6.txt
```

Dessa vez, o símbolo '-' (menos) não denota uma sequência de valores, não se trata de um alcance de 4 a 6. Neste caso, é procurado pela existência do arquivo "lista4-6.txt"; e não pelos arquivos "arquivo4.txt", "arquivo5.txt" e "arquivo6.txt" como seria se em vez de chaves fosse usado colchetes.

```
$> ls lista{,1,2,3,4-6}.txt
lista.txt   lista1.txt   lista2.txt   lista3.txt   lista4-6.txt
```

Aqui, o primeiro elemento do conjunto interno às chaves é nada. Dessa forma, o *prompt* vai interpretar que, para o primeiro elemento, nenhum carácter deve ser usado para compor o nome do arquivo desejado. Ou seja, o *prompt* vai procurar pelo arquivo "lista.txt".

Diferentemente do colchetes, para as chaves todos os elementos do conjunto interno devem existir.

1.8 Processos e *jobs*

Em Linux todos os programas em execução podem ser chamados de processos e são identificados por um número identificador próprio chamado PID, *Process Identification*.

O comando "ps" pode ser utilizado para verificar informações referentes aos processos em execução.

Os processos podem estar em 3 estados diferentes: em *foreground* (primeiro plano), em *background* (segundo plano) ou suspensos.

Processos em *foreground* costumam segurar o controle do terminal até encerrarem, um modo de executar o processo sem que este mantenha o controle do terminal é deixá-lo em *background*. Para deixar um processo em *background* ao executá-lo coloque o símbolo '&' ao final do comando separado por um espaço. Se o processo já estiver em execução é possível pressionar **CTRL + Z** para suspender o processo e depois digitar o comando "bg" para colocar o processo suspenso em *background*.

Ao digitar o comando "jobs" você verá a lista dos processos que estão rodando em *background*. Para colocar um processo em *foreground* é possível digitar o "fg %numero_do_*job*", "fg" sem parâmetros fará o último processo posto em *background* ficar em *foreground*.

Exemplo:

```
$> xclock
^Z
[1]+  Stopped                 xclock
$> ps
  PID TTY          TIME CMD
 2068 pts/0    00:00:00 bash
 2825 pts/0    00:00:00 xclock
 2834 pts/0    00:00:00 ps
$> jobs
[1]+  Stopped                 xclock
$> fg 1
xclock
```

No exemplo acima o comando "xclock" foi executado, este programa exibe uma pequena janela contendo um relógio. O "^Z" indica que **CTRL + Z** foi pressionado o que suspendeu o "xclock" e devolveu o controle do terminal ao usuário. Ao executar o comando "ps" o PID de cada programa em execução pode ser verificado, enquanto "jobs" exibe o número job de cada programa em colchetes, no caso "[1]" significa que o "xclock" é o job de número 1. Por fim "fg 1" colocou novamente o programa xclock em execução e com o controle do terminal.

1.8.1 Matando processos

Para matar um processo que esteja suspenso ou em *background* basta digitar o comando "kill %numero_do_*job*" também é possível matar o processos pelo PID, bastando digitar "kill numero_do_PID" o parâmetro "-9" força o processo a fechar mesmo quando ele se recusa a encerrar.

1.9 Boas práticas em nomenclaturas Linux

Seguem algumas sugestões de boas práticas de nomenclatura de nomes de arquivos que evitam principalmente problemas com a codificação de caracteres e manipulação de arquivos.

- Evite utilizar acentos e símbolos em nomes de arquivo, ao invés de espaço prefira *underscore* como separador. Por exemplo, meu_codigo.pas;
- Utilize extensões usuais para seus arquivos. Por exemplo, ".c" para códigos escritos em linguagem C, ".pas" para códigos em Pascal, ".txt" para arquivos de texto comuns;
- Adote letras minúsculas para iniciar nome de arquivos.

2 Comandos Linux

Nesta seção, é apresentada uma coletânia com os comandos Linux mais comuns. Cada um desses comandos está listado de acordo com um determinado número de campos explicativos: sintaxe (representando a forma de uso do comando em um terminal), descrição (explicação do que o comando faz), opções (as opções, argumentos, parâmetros mais utilizados para o programa) e exemplo (exemplificação de uso do programa).

Para tanto, são usados arquivos com o seguinte conteúdo:

```
$> cat lista.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I

$> cat lista2.txt
Pascal
C
Java

$> cat lista3.txt
leg04
jos04
rums04

$> cat numeros.txt
1236
4498
14366
18951
6958
28317
4305
13709
30617
27806
```

Na ordem: "lista.txt" contém um conjunto de nomes de disciplinas; "lista2.txt", um conjunto de linguagens de programação; "lista3.txt", um conjunto de logins; "números.txt", um conjunto de números obtidos aleatoriamente.

Além destes, também aparecem nos textos a seguir os arquivos:

- "verbos.txt" que possui como conteúdo um conjunto de verbos;

- “substantivos.txt”, um conjunto de substantivos;
- “foto1.jpg” e “foto2.jpg”, dois arquvos binários que representam fotos;
- “linklista”, um *link* para arquivo;
- “.arquivo-oculto”, um arquivo oculto;
- “diretorio-listas” e “dir-arquivos/documentos”, diretórios.

OBS.: as opções descritas nas subseções a seguir podem não ser todas as opções disponíveis para o programa. Verificar todas as opções acessando a *man page* do comando desejado.

2.1 cat

Sintaxe: cat [OPÇÃO]... [ARQUIVO]...

Descrição: Concatena arquivos e imprime na saída padrão

Opções: -n (enumera todas as linhas de saída)

Exemplo:

```
$> cat lista.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I

$> cat -n lista2.txt
     1  Pascal
     2  C
     3  Java

$> cat lista.txt lista2.txt lista3.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I
Pascal
C
Java
leg04
jos04
rums04
```

2.2 cd (comando embutido do bash)

Sintaxe: cd [-L|-P] [dir]

Descrição: Muda o diretório corrente para "dir".

Opções: -L (segue *links* simbólicos)
-P (usa a estrutura física de diretórios em vez de seguir *links* simbólicos)

Exemplo:

```
$> cd /home/prof/professor/disciplina/pdfs
/home/prof/professor/disciplina/pdfs$>

/home/prof/professor$> cd disciplina/pdfs
/home/prof/professor/disciplina/pdfs$>

/home/prof/professor$> cd ..
/home/prof$>

/home/prof/professor$> cd ../..
/home$>
```

2.3 cp

Sintaxe: cp [opções] arquivo caminho
cp [opções] arquivo... diretório

Descrição: Copia arquivos e diretórios

Opções: -i (pergunta se será sobrescrito o arquivo regular de destino)
-f (remove os arquivos de destino existentes, e nunca pergunta antes de fazer isto)
-r (Copia diretórios de forma recursiva, copiando qualquer não- diretório e não-ligação simbólica (isto é, FIFOs e arquivos especiais) como se fossem arquivos regulares. Isto significa tentar uma leitura em cada arquivo origem e uma escrita para o destino. Assim, com esta opção, (Isto é uma falha. Ele pensar que você tem que evitar -r e usa -R se você não sabe o que esta na árvore que você está copiando. Abrindo um arquivo de dispositivo desconhecido, por exemplo um scanner, tem efeito desconhecido sobre o hardware.))

Exemplo:

```
$> cp lista.txt lista2.txt lista3.txt diretorio-listas

$> cp -r diretorio-listas dir-arquivos/documentos
```

```
$> cp -i lista.txt lista2.txt
cp: sobrescrever ‘lista2.txt’? y

$> cp -f lista.txt lista2.txt
$>
```

2.4 diff

Sintaxe: diff [opções] do-arquivo para-arquivo

Descrição: Acha diferenças entre dois arquivos

Opções:

Exemplo:

```
$> diff verbos.txt substantivos.txt
2,3c2,3
< Para
< Continua
---
> Paralisacao
> Continuo

$ diff lista.txt lista2.txt
1,3c1,3
< Circuitos Logicos
< Algoritmos e Estrutura de Dados I
< Calculo Diferencial e Integral I
---
> Pascal
> C
> Java

$> diff foto1.jpg foto2.jpg
Arquivos binários foto1.jpg e foto2.jpg diferem

$> diff lista.txt lista.txt
$>
```

2.5 du

Sintaxe: du [opções] [arquivo...]

Descrição: Estima o espaço utilizado

Opções: -h (anexa o rótulo de tamanho, como por exemplo M para binários de megabytes ('mebibytes'), para cada tamanho)
-s (exibe somente um total para cada argumento)

Exemplo:

```
$> du lista.txt
4       lista.txt

$> du -h lista.txt lista2.txt lista3.txt
4,0K    lista.txt
4,0K    lista2.txt
4,0K    lista3.txt

$> du -sh .
943M    .
```

2.6 echo

Sintaxe: echo [OPÇÃO-CURTA]... [STRING]...
echo OPÇÃO-LONGA

Descrição: Imprime uma linha de texto

Opções: -e (habilita interpretação de meta caracteres disparados pela '\' (contrabarra - *backslash escape*)
-n (não mostra nova-linha à direita - *trailing newline*)

Exemplo:

```
$> echo "Nova linha de texto"
Nova linha de texto
$>

$> echo -n "Nova linha de texto"
Nova linha de texto$>

$> echo -e "Nova\nlinha\nde\ntexto\n"
Nova
linha
de
texto

$>

$> echo -en "Nova\nlinha\nde\ntexto\n"
```

```
Nova
linha
de
texto
$>
```

2.7 file

Sintaxe: file [-bchikLnNprsvz] [–mime-type] [–mime-encoding] [-f namefile] [-F separator] [-m magicfiles] file
file -C [-m magicfile]
file [–help]

Descrição: Determina o tipo do arquivo

Opções: -i (mostra o tipo MIME do arquivo)

Exemplo:

```
$> file lista.txt
lista.txt: ASCII text

$> file -i lista.txt
lista.txt: text/plain; charset=us-ascii

$> file dir-arquivos
dir-arquivos: directory
```

2.8 find

Sintaxe: find [-H] [-L] [-P] [-D debugopts] [-Olevel] [path...] [expression]

Descrição: Procura por arquivos em uma hierarquia de diretórios

Opções: -name (procura por arquivos que contenham esse nome)
-newer (procura por arquivos que sejam mais novos que este)
-size (procura por arquivos que tenham o tamanho especificado aqui)
-exec (executa um comando sobre o resultado da busca - o conjunto de resultados é representado por {})

Exemplo:

```
$> find . -name "lista*"
./lista.txt
./lista2.txt
./lista3.txt
```

```
$> find . -name "lista?.txt"
./lista2.txt
./lista3.txt

$> find . -size +30M

$> find . -name "lista*txt" -exec ls -l {} \;
-rw-r--r-- 1 josineys ppginf 85 2012-04-20 17:24 ./lista.txt
-rw-r--r-- 1 josineys ppginf 14 2012-04-20 17:24 ./lista2.txt
-rw-r--r-- 1 josineys ppginf 19 2012-04-20 17:24 ./lista3.txt

$> find . -newer lista.txt
.
./lista2.txt
./lista3.txt
```

2.9 finger

Sintaxe: finger [-lmsp] [usuário ...] [usuário@máquina ...]

Descrição: Programa de busca por informações de usuários

Opções:

Exemplo:

```
$> finger danilok
Login: danilok                          Name: Danilo Kiyoshi Simizu Yorinori
Directory: /home/c3sl/danilok           Shell: /bin/bash
Office Phone: 2009-03-10                Home Phone: resp. Castilho
Last login Thu Dec 29 02:44 (BRST) on pts/3 from 187.59.71.243
Mail forwarded to /home/c3sl/danilok/Mail/Mailbox
No mail.
No Plan.
```

2.10 grep

Sintaxe: grep [OPÇÕES] PADRÃO [ARQUIVO...]
grep [OPÇÕES] [-e PADRÃO — -f ARQUIVO] [ARQUIVO...]

Descrição: Imprime linhas que coincidam com um padrão

Opções: -i (ignora diferenciação de maiúsculas e minúsculas - *case sensitive*)

Exemplo:

```
$> grep "al" lista*txt
lista2.txt:Pascal
lista.txt:Calculo Diferencial e Integral I

$> grep "cal" lista*txt
lista2.txt:Pascal

$> grep -i "cal" lista*txt
lista2.txt:Pascal
lista.txt:Calculo Diferencial e Integral I
```

2.11 groups

Sintaxe: groups [OPÇÃO]... [NOME_DE_USUÁRIO]...

Descrição: Imprime os grupos de usuários em que um usuário está

Opções:

Exemplo:

```
$> groups
bcc

$> groups danilok
danilok : c3sl
```

2.12 head

Sintaxe: head [OPÇÃO]... [ARQUIVO]...

Descrição: Mostra a primeira parte de arquivos

Opções: -c (Mostra os primeiros N bytes de cada arquivo - se tiver '-' (menos) antes do número, mostra tudo menos os N últimos bytes)
-n (Mostra as primeiras N linhas em vez das 10 primeiras (10 primeiras é o comportamento padrão) - se tiver '-' (menos) antes do número, mostra tudo menos as N últimas linhas)

Exemplo:

```
$> head -c 20 lista.txt
Circuitos Logicos
Al$>

$> head -c 18 lista.txt
Circuitos Logicos
```

```
$>

$> head -c -18 lista.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferen$>

$> head -n 1 lista.txt
Circuitos Logicos

$> head -n -1 lista.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I

$> head -n 0 lista.txt
$>

$> head -n -0 lista.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I
```

2.13 id

Sintaxe: id [OPÇÃO]... [NOME_DE_USUÁRIO]...

Descrição: Imprime IDs reais e efetivos para usuário e grupo

Opções:

Exemplo:

```
$> id
uid=2847(josineys) gid=1011(ppginf) grupos=1005(c3sl),1011(ppginf)

$> id danilok
uid=1246(danilok) gid=1005(c3sl) grupos=1005(c3sl)
```

2.14 jobs (comando embutido do bash)

Sintaxe: jobs [-lnprs] [jobspec ...]
jobs -x command [args ...]

Descrição: Lista os jobs ativos para o shell atual

Opções:

Exemplo:

```
$> jobs
$>

$> jobs
josineys@macalan:~$ jobs
[1]-  Executando              xterm &
[2]+  Executando              xterm &
```

2.15 kill

Sintaxe: kill [-signal — -s signal] pid ...
kill [-L — -V, –version]
kill -l [signal]

Descrição: Manda um sinal para um processo

Opções: -9 (manda o sinal de matar - SIGKILL - aos processos identificados pelo número de PID)
-1 (após -9, informa que deve ser mortos todos os processos que o usuário pode matar)

Exemplo:

```
$> kill 123 543 2341 3453

$> kill -9 123 543 2341 3453

$> kill -9 -1

$> kill %1
```

2.16 less

Sintaxe: less -?
less –help
less -V
less –version
less [-[+]aBcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX] [-b space] [-h lines] [-j line] [-k keyfile] [-oO logfile] [-p pattern] [-P prompt] [-t tag] [-T tagsfile] [-x tab,...] [-y lines] [-[z] lines] [-# shift] [+[+]cmd] [–] [filename]...

Descrição: Exibe conteúdo de arquivos de forma mais sofisticada do que “more”

Opções: -N (mostra o número da linha visualizada, como no comando "cat" (ver Seção 2.1)

Exemplo:

```
$> less lista.txt

$> less -N lista.txt
```

2.17 ln

Sintaxe: ln [opções] origem [destino]
ln [opções] origem... diretório

Descrição: Cria uma ligação simbólica entre arquivos

Opções: -i (questiona se remove o arquivo de destino existente)
-f (remove o arquivo de destino existente) -s (cria uma ligação fraca - *soft*, simbólica - ao invés de ligações fortes)

Exemplo:

```
$> ln -s lista.txt linklista

$> ln -i lista.txt linklista
ln: substituir ‘linklista’? y

$> ln -f lista.txt linklista
$>
```

2.18 ls

Sintaxe: ls [opções] [arquivo...]

Descrição: Lista o conteúdo de um diretório

Opções: -a (inclui os arquivos com o nome iniciando com '.' na listagem)
-R (lista os diretórios encontrados, recursivamente)
-d (lista nome de diretórios como arquivo, preferencialmente no lugar de seus conteúdos)
-l (escreve várias informações sobre o arquivo ou diretório listado) -r (inverte a ordem do ordenação)
-1 (para saída em colunas simples)

Exemplo:

```
$> ls
dir-arquivos lista2.txt lista3.txt lista.txt

$> ls -l
drwxr-xr-x 1 josineys ppginf 85 2012-04-20 17:24 dir-arquivos
-rw-r--r-- 1 josineys ppginf 85 2012-04-20 17:24 lista.txt
-rw-r--r-- 1 josineys ppginf 14 2012-04-20 17:24 lista2.txt
-rw-r--r-- 1 josineys ppginf 19 2012-04-20 17:24 lista3.txt

$> ls dir-arquivos
foto1.jpg  foto2.jpg

$> ls -d dir-arquivos
dir-arquivos

$> ls -R
.:
dir-arquivos  lista2.txt  lista3.txt  lista.txt

./dir-arquivos:
foto1.jpg  foto2.jpg

$> ls -a
.  ..  .arquivo-oculto  dir-arquivos  lista2.txt  lista3.txt  lista.txt

$> ls -1
dir-arquivos
lista2.txt
lista3.txt
lista.txt
```

2.19 man

Sintaxe: man [-C file] [-d] [-D] [–warnings[=warnings]] [-R encoding] [-L locale] [-m system[,...]] [-M path] [-S list] [-e extension] [-i—-I] [–regex—-wildcard] [–names-only] [-a] [-u] [–no-subpages] [-P pager] [-r prompt] [-7] [-E encoding] [–no-hyphenation] [–no-justification] [-p string] [-t] [-T[device]] [-H[browser]] [-X[dpi]] [-Z] [[section] página ...] ...
man -k [apropos options] regexp ...
man -K [-w—-W] [-S list] [-i—-I] [–regex] [section] term ...
man -f [whatis options] page ...
man -l [-C file] [-d] [-D] [–warnings[=warnings]] [-R encoding] [-L locale] [-P pager] [-r prompt] [-7] [-E encoding] [-p string] [-t] [-T[device]] [-H[browser]] [-X[dpi]] [-Z] arquivo ...

man -w—-W [-C file] [-d] [-D] página ...
man -c [-C file] [-d] [-D] página ...
man [-hV]

Descrição: Uma interface para os manuais on-line de referência

Opções:

Exemplo:

```
$> man ls
```

2.20 mkdir

Sintaxe: mkdir [opções] diretório...

Descrição: Cria diretórios

Opções: -p (cria os diretórios-pai de um caminho, caso eles não existam ainda)
-m (indica o modo - permissões - de um diretório no momento de sua criação)

Exemplo:

```
$> mkdir -m 760 novo-dir
```

```
$> mkdir dir-arquivos/dir1/dir2/dir3/novo-dir
mkdir: é impossível criar o diretório ‘dir-arquivos/dir1/dir2/dir3/novo-dir’:
Arquivo ou diretório não encontrado
```

```
$> mkdir -p dir-arquivos/dir1/dir2/dir3/novo-dir
```

2.21 more

Sintaxe: more [-dlfpcsu] [-num] [+/padrão] [+linhanum] [arquivo ...]

Descrição: Exibe conteúdo de um arquivo

Opções:

Exemplo:

```
$> more lista.txt
```

2.22 mv

Sintaxe: mv [opção...] origem destino
mv [opção...] origem... destino

Descrição: Movimenta e/ou renomeia arquivos

Opções: -i (

Opções: -i (questiona ao usuário se deseja sobrescrever um arquivo de destino existente)
-f (apaga o arquivo de destino existente sem perguntar ao usuário)

Exemplo:

```
$> mv lista.txt lista_renomeada.txt

$> mv -i lista.txt lista2.txt
mv: sobrescrever ‘lista2.txt’? n

$> mv -f lista.txt lista2.txt
$>
```

2.23 pkill

Sintaxe: pkill [-sinal] [-fvx] [-n—-o] [-P ppid,...] [-g pgrp,...] [-s sid,...] [-u euid,...] [-U uid,...] [-G gid,...] [-t term,...] [padrão]

Descrição: Mata processos baseado no nome e outros atributos

Opções: -sinal (sinal a ser enviado ao processo - tanto sinal numérico quanto nome simbólico do sinal)

Exemplo:

```
$> pkill -9 firefox
```

2.24 ps

Sintaxe: ps [opções]

Descrição: Retorna uma fotografia dos processos correntes

Opções: a (todos os processos no sistema)
x (lista todos os processos pertencentes ao usuário)
u (mostra em formato de orientação ao usuário)

Exemplo:

```
$> ps
  PID TTY          TIME CMD
17906 pts/26   00:00:00 bash
18064 pts/26   00:00:00 ps

$> ps xu
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
josineys  9779  0.0  0.0  96920  2568 ?        SN   14:26   0:00 sshd: josineys@pts/
josineys  9780  0.0  0.0  20160  5800 pts/21   SNs  14:26   0:00 -bash
josineys 17407  0.0  0.0 172012  8876 pts/21   SNl+ 15:16   0:00 vi documento.txt
josineys 17905  0.0  0.0  96920  2568 ?        SN   15:19   0:00 sshd: josineys@pts/
josineys 17906  0.3  0.0  20160  5784 pts/26   SNs  15:19   0:00 -bash
josineys 18253  0.0  0.0  12232  1132 pts/26   RN+  15:20   0:00 ps xu
```

2.25 quota

Sintaxe: quota [-F format-name] [-guqvswi] [-l — [-QAm]]
quota [-F format-name] [-qvswi] [-l — [-QAm]] -u user...
quota [-F format-name] [-qvswi] [-l — [-QAm]] -g group...
quota [-F format-name] [-qvswugQm] -f filesystem...

Descrição: Mostra o uso de disco e limites

Opções: -s (tenta usar unidades legíveis aos humanos - KB, MB, ...)

Exemplo:

```
$> Cotas de disco para user josineys (uid 2847):
Sistema de arquivos  blocos   quota  limite   grace arquivos   quota  limite
grace
10.17.110.3:/home
                1787884  2000000 2200000           35381       0       0

$> quota -s
Cotas de disco para user josineys (uid 2847):
Sistema de arquivos  blocos   quota  limite   grace arquivos   quota  limite
grace
10.17.110.3:/home
                  1746M   1954M   2149M           35381       0       0
```

2.26 rm

Sintaxe: rm [opções] arquivo...

Descrição: Apaga arquivos e diretórios

Opções: -i (questiona se cada arquivo será apagado. Se a reposta for negativa, o arquivo é preservado) de destino existente)
-f (ignora arquivos não existentes a nunca questiona o usuário) -r (apaga o conteúdo dos diretórios de forma recursiva) -R (igual a -r)

Exemplo:

```
$> rm lista.txt

$> rm dir-arquivos
rm: não foi possível remover ‘dir-arquivos/’: É um diretório

$> rm -r dir-arquivos

$> rm -i lista.txt
rm: remover arquivo comum ‘lista.txt’? n
```

2.27 rmdir

Sintaxe: rmdir [opções] diretório...

Descrição: Remove diretórios vazios

Opções:

Exemplo:

```
$> rmdir dir-arquivos
rmdir: falhou em remover ‘dir-arquivos’: Diretório não vazio

$> rmdir dir-vazio
```

2.28 sort

Sintaxe: sort [OPÇÃO]... [ARQUIVO]...
sort [OPÇÃO]... –files0-from=F

Descrição: Ordena linhas de arquivos de texto ou da entrada padrão

Opções: -n (compara de acordo com o valor da string numérica)
-r (ordenação em ordem reversa)
-u (ordena de forma única - exclui repetições)

Exemplo:

```
$> sort numeros.txt
1236
13709
14366
18951
27806
28317
30617
4305
4498
6958

$> sort -n numeros.txt
1236
4305
4498
6958
13709
14366
18951
27806
28317
30617

$> cat lista.txt | sort
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I
Circuitos Logicos

$> sort -r lista.txt
Circuitos Logicos
Calculo Diferencial e Integral I
Algoritmos e Estrutura de Dados I
```

2.29 tail

Sintaxe: tail [OPÇÃO]... [ARQUIVO]...

Descrição: Mostra a última parte de arquivos

Opções: -c (Mostra os últimos N bytes - se tiver '+' (mais) antes do número, mostra tudo a partir dos N primeiros bytes)
-n (Mostra as últimas N linhas em vez das 10 primeiras (10 primeiras é o comportamento padrão) - se tiver '+' (mais) antes do número, mostra tudo a partir da N-ésima linhas)

Exemplo:

```
$> tail -n 1 lista.txt
Calculo Diferencial e Integral I

$> tail -n +1 lista.txt
Circuitos Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I

$> tail -c 10 lista.txt
ntegral I

$> tail -c +10 lista.txt
 Logicos
Algoritmos e Estrutura de Dados I
Calculo Diferencial e Integral I
```

2.30 tar

Sintaxe: tar [-] A –catenate –concatenate — c –create — d –diff –compare — –delete — r –append — t –list — –test-label — u –update — x –extract –get [options] [pathname ...]

Descrição: Gerencia compactação de arquivos no formato tar

Opções: -c (Cria um novo arquivo)
-x (extrai arquivos de um arquivo compactado)
-j (filtra o arquivo compactado através do bzip2)
-lzma (filtra o arquivo compactado através do lzma)
-z (filtra o arquivo compactado através do gzip)
-t (lista o conteúdo do arquivo compactado)
- (menos - opcional)

Exemplo:

```
$> tar xzf slides.tar.gz

$> tar cf listas.tar lista*

$> tar tf listas.tar
lista2.txt
lista3.txt
lista.txt
```

2.31 touch

Sintaxe: touch [-acm][-r arquivo_de_referência—-t tempo] arquivo...

Descrição: Altera o rótulo de tempo do arquivo

Opções: -c (não cria arquivos que não existam; por padrão, apenas o uso do "touch" sem argumentos faz com que arquivos inexistentes sejam criados com tamanho zero - arquivos vazios)

Exemplo:

```
$> touch arquivo_vazio
```

2.32 wc

Sintaxe: wc [OPÇÃO]... [ARQUIVO]...
wc [OPÇÃO]... –files0-from=F

Descrição: Imprime o número de linhas, palavras e bytes para cada arquivo

Opções: -c (imprime o número de bytes)
-m (imprime o número de caracteres)
-l (imprime o número de linhas)
-w (imprime o número de palavras)

Exemplo:

```
$> wc lista.txt
 3 13 85 lista.txt

$> wc -l lista2.txt
3 lista2.txt
```