CENTRO PAULA SOUZA GOVERNO DO ESTADO DE SÃO PAULO

FACULDADE DE TECNOLOGIA DE LINS

Análise e Desenvolvimento de Sistemas

Disciplina: Sistemas Operacionais II

Parte 1

Fatec Lins

Professor: Alexandre Ponce de Oliveira

Criação: 2008 – Revisão: 2015/2022

Sumário

1. Estudo de Caso - Linux

O Kernel do Linux foi, originalmente, escrito por Linus Torvalds do Departamento de Ciência da Computação da Universidade de Helsinki, Finlândia, com a ajuda de vários programadores voluntários através da Internet.

Linus Torvalds iniciou cortando o kernel como um projeto particular, inspirado em seu interesse no Minix, um pequeno sistema UNIX desenvolvido por Andrey Tannenbaum. Ele se limitou a criar, em suas próprias palavras, "um Minix melhor que o Minix" ("a better Minix than Minix"). E depois de algum tempo de trabalho em seu projeto, sozinho, ele enviou a seguinte mensagem para comp.os.minix:

Você suspira por melhores dias do Minix-1.1, quando homens serão homens e escreverão seus próprios "device drivers"? Você está sem um bom projeto e esta morrendo por colocar as mãos em um S.O. no qual você possa modificar de acordo com suas necessidades? Você está achando frustrante quando tudo trabalha em Minix? Chega de atravessar noites para obter programas que trabalhem corretos? Então esta mensagem pode ser exatamente para você.

Como eu mencionei há um mês, estou trabalhando em uma versão independente de um S.O. similar ao Minix para computadores AT-386. Ele está, finalmente, próximo do estágio em que poderá ser utilizado (embora possa não ser o que você esteja esperando), e eu estou disposto a colocar os fontes para ampla distribuição. Ele está na versão 0.02... contudo eu tive sucesso rodando bash, gcc, gnu-make, gnu-sed, compressão, etc. nele.

No dia 5 de outubro de 1991 Linus Torvalds anunciou a primeira versão "oficial" do Linux, versão 0.02. Desde então muitos programadores têm respondido ao seu chamado, e têm ajudado a fazer do Linux o Sistema Operacional que é hoje.

O Linux é um dos mais proeminentes exemplos de desenvolvimento com código aberto e de software livre. O seu código fonte está disponível sob licença GPL para qualquer pessoa utilizar, estudar, modificar e distribuir livremente.

Softwares Livres são programas que possuem o código fonte incluído (o código fonte é o que o programador digitou para fazer o programa) e você pode modificar ou distribuí−los livremente. Existem algumas licenças que permitem isso, a mais comum é a *General Public Licence* (GPL - Licença Pública Geral).

Os softwares livres muitas vezes são chamados de programas de código aberto. Muito se acredita no compartilhamento do conhecimento e tendo liberdade de cooperar uns com outros, isto é importante para o aprendizado de como as coisas funcionam e novas técnicas de construção. Existe uma longa teoria desde 1950 valorizando isto, muitas vezes pessoas assim são chamadas de “Hackers Éticos”.

Outros procuram aprender mais sobre o funcionamento do computador e seus dispositivos (periféricos) e muitas pessoas estão procurando por meios de de evitar o preço absurdo de softwares comerciais através de programas livres que possuem qualidade igual ou superior, devido a cooperação em seu desenvolvimento.

Você pode modificar o código fonte de um software livre a fim de melhora−lo ou acrescentar mais recursos e o autor do programa pode ser contactado sobre a alteração e os benefícios que sua modificação fez no programa, e esta poderá ser incluída no programa principal. Deste modo, milhares de pessoas que usam o programa se beneficiarão de sua contribuição.

Em computação, o projeto GNU é um projeto iniciado por Richard Stallman em 1984, com o objetivo de criar um sistema operacional totalmente livre, aonde qualquer pessoa teria direito de usar, modificar e redistribuir o programa juntamente com seu código fonte, desde que garanta para todos esses mesmos direitos.

Este sistema operacional GNU deveria ser compatível com o sistema operacional UNIX, porém não deveria utilizar-se do código fonte do UNIX. Stallman escolheu o nome GNU porque este nome, além do significado original do mamífero Gnu, é um acrônimo recursivo de: **G**NU is **N**ot **U**nix (em português: GNU não é Unix).

Características do UNIX e do LINUX

Multitarefa – Um único usuário pode requisitar que sejam efetuados vários comandos ao mesmo tempo em seu terminal. É responsabilidade do sistema UNIX controlar estas execuções paralelas.

Quando um usuário executa mais de um comando ao mesmo tempo, geralmente é somente um que necessita a interação com o usuário. Os demais comandos executados são na sua maioria comandos que não exigem a atenção do usuário, sendo tarefas demoradas. Quando isto ocorre, dizemos que os programas que o usuário está executando sem a interação ficam em *Background (*um segundo plano).

Multiusuário – O UNIX pode controlar o acesso ao sistema através de vários terminais, virtuais ou reais, cada um pertencendo a um usuário. O UNIX aceita as requisições de comandos de cada um dos usuários e gera as filas de controle e prioridades para que haja uma distribuição correta dos recursos de *hardware* necessários a cada usuário. Devido a característica de ser um sistema multiusuário, o UNIX implementa um sistema de segurança visando impedir o acesso aos arquivos e diretórios de um usuário por outro. No módulo sobre permissões veremos como se pode liberar ou restringir o acesso entre usuários.

Kernel – Como é chamado o núcleo do sistema Unix e Linux. Este núcleo faz o gerenciamento direto dos dispositivos de E/S (*device drivers*), gerenciamento de memória e controle do uso da CPU pelos vários processos do sistema.

Processo – É um conceito básico do sistema. Toda vez que se executa um programa/comando é gerado um processo no sistema. Todo gerenciamento é feito sobre este processo. Os processos são, portanto, comandos/programas em execução. Todo processo é identificado por um número chamado *process id* (PID). Este *process ID* é único no sistema durante a execução do processo, portanto pode e deve ser usado para identificação do processo em caso de necessidade.

1.1 Interpretador de Comandos

Também conhecido como *shell*. É o programa responsável em interpretar as instruções enviadas pelo usuário e seus programas ao sistema operacional (o *kernel*). Ele que executa comandos lidos do dispositivo de entrada padrão (teclado) ou de um arquivo executável. É a principal ligação entre o usuário, os programas e o *kernel*. O *Unix/Linux* possui diversos tipos de interpretadores de comandos, entre eles podemos destacar o *bash (Bourne Again Shell), ksh (Korn Shell), csh (C Shell), tcsh (*versão padronizada do *csh*), entre outros. Entre eles o mais usado é o *bash* (no Linux). O interpretador de comandos do DOS, por exemplo, é o *command.com*.

Os comandos podem ser enviados de duas maneiras para o interpretador: interativa e não-interativa:

- Interativa – Os comandos são digitados no interpretador de comandos um a um. Neste modo, o computador depende do usuário para executar uma tarefa, ou próximo comando.

- Não-interativa – São usados arquivos de comandos criados pelo usuário (scripts) para o computador executar os comandos na ordem encontrada no arquivo. Neste modo, o computador executa os comandos do arquivo um por um e dependendo do término do comando, o script pode checar qual será o próximo comando que será executado e dar continuidade ao processamento. Este sistema é útil quando temos que digitar por várias vezes seguidas um mesmo comando ou para compilar algum programa complexo.

O *shell bash* possui ainda outra característica interessante: A complementação dos nomes. Isto é feito pressionando-se a tecla TAB. Por exemplo, se digitar "ls tes" e pressionar <tab>, o *bash* localizará todos os arquivos que iniciam com "tes" e completará o restante do nome. Caso a complementação de nomes encontre mais do que uma expressão que satisfaça a pesquisa, ou nenhuma, é emitido um *beep*. A complementação de nomes funciona sem problemas para comandos internos.

1.2 Usuários

Dentro do sistema existem dois tipos de usuários: normal e super-usuário. O usuário comum é aquele que tem acesso limitado somente a seus dados e arquivos. Se tentar acessar dados de outro usuário, o sistema, dependendo das permissões configuradas, não deixará, emitindo uma mensagem de erro.

O super-usuário ou conta *root* é uma conta com poderes supremos sobre toda a máquina. Com ela pode-se acessar qualquer arquivo que se encontra na máquina, removê-lo, mudá-lo de lugar, entre outras ações. O esquema de permissão e segurança do Unix/Linux não se aplica ao super-usuário.

Existem também certos comandos que só podem ser executados quando o usuário tem permissão de super-usuário. Estes comandos geralmente servem para a manutenção do sistema e não devem ser deixados à disposição de usuários comuns devido à complexidade e perigo no uso destes comandos.

1.3 Principais diretórios

Os diretórios de um sistema de arquivos têm uma estrutura pré-definida, com poucas variações. A seguir ilustramos os principais:

- **/home**: raiz dos diretórios home dos usuários;
- **/boot**: arquivos de boot (kernel do sistema);
- **/var**: arquivos variáveis, áreas de spool (impressão, e-mail, news), arquivos de log;
- **/etc**: arquivos de configuração dos serviços;
- **/usr**: aplicações voltadas aos usuários;
- **/tmp**: arquivos temporários;
- **/mnt**: montagem de diretórios compartilhados temporários;
- **/bin**: aplicações de base para o sistema;
- **/dev**: arquivos de acesso aos dispositivos físicos e conexões de rede;
- **/lib**: bibliotecas básicas do sistema.

1.4 Diretório HOME

Cada usuário possui um diretório especial, chamado "**diretório home**" (casa), onde são armazenados:

- arquivos e diretórios pessoais de trabalho;
- e-mails já lidos (pastas pessoais);
- arquivos de configuração individuais;
- configuração das aplicações usadas.

O diretório ***home*** do usuário é o seu local de início de sessão de trabalho (via *shell* ou gráfica). O usuário possui plenos poderes de acesso ao seu diretório *home* (e seus sub-diretórios), e normalmente não pode criar arquivos fora dele. O diretório *home* de cada usuário é normalmente inacessível aos outros usuários, mas isso pode ser controlado pelo administrador do sistema (*root*).

1.5 Comandos

Formato Geral de um Comando: **comando [opções] [argumentos]**

- Comando – Comando ou programa a ser executado
- Opções – Modificadores do comando (opcional)
- Argumentos – Define o objeto a ser afetado pelo comando (opcional)

A maioria dos comandos Unix/Linux possui a sintaxe compatível ao formato acima. Temos o nome do comando, seguido de opções e argumentos. As opções, quando colocadas, devem sempre preceder os argumentos.

Observar que os caracteres separadores dos campos da linha de comando são o espaço em branco e o <Tab>. Um outro detalhe, muito importante, é o fato de que o Unix/Linux faz distinção entre os caracteres maiúsculos e minúsculos. Portanto, para o Unix/Linux, **Ls** é diferente de **ls** por exemplo.

Quase sempre as opções dos comandos são precedidas pelo caractere "-" (menos) ou "+" (mais) e podem entrar em qualquer ordem e posição na linha de comando, mas sempre antes dos argumentos (há poucas exceções). Na maioria das vezes as opções são representadas por letras, podendo-se agrupar uma série de letras em uma única opção. Por exemplo, as opções "-w -l -c" do comando **wc** podem ser escritas como "-wlc". Existem também opções que são mutuamente exclusivas, não podendo aparecer ao mesmo tempo em um comando.

O terceiro tipo de opção que pode existir em um comando, é a opção que exige logo após, um argumento específico. Neste caso, quase sempre esta opção é colocada separada, precedida por "-" ou "+" e seguida de seu argumento Caso ela seja colocada juntamente com as demais opções, ela deve ser a última da lista.

Um detalhe que gera muita confusão para o iniciante do sistema Unix/Linux é o fato de que as opções variam de comando para comando, tornando-se difícil uma memorização das mesmas. Se isto acontecer com você, não se preocupe, pois poucas pessoas sabem todas as opções de todos os comandos.

Os argumentos definem os objetos sobre os quais o comando será aplicado. Temos como exemplos de argumentos: arquivos, periféricos, entre outros.

1.6 Comandos Básicos

1.6.1 Iniciando uma Sessão

O acesso ao sistema é feito através de uma conta previamente cadastrada no sistema pelo administrador do sistema. Sem esta conta não é possível o acesso ao sistema. Esta conta é chamada *user name* ou *login name*. A conta deve ser única em cada máquina.

Junto com a conta, é atribuída uma senha de acesso. Esta senha é que garante que não se faça nenhum acesso indevido aos dados de determinado usuário. A manutenção desta senha e de sua validade é de responsabilidade única do usuário.

Uma vez acertada a conta e a senha, o sistema apresentará um *prompt* indicando que ele está apto a receber comandos do usuário. Neste ponto dizemos que estamos com uma sessão aberta.

O esquema de segurança, baseado em senhas, perde seu sentido se o usuário não seguir certas normas de segurança. Estas normas são simples e óbvias e o cumprimento delas pode evitar transtornos futuros. As normas são as seguintes:

- Não forneça sua senha para ninguém;
- Não use palavras óbvias ligadas a você (Nome próprio ou de parentes, iniciais, nome do cachorro, números do telefone, nome do setor, palavras ligadas ao trabalho, etc.);
- Mude regularmente a sua senha por medida de segurança ou toda vez que achar que sua senha foi descoberta;
- Não anote a senha em nenhum lugar. Memorize-a.

1.6.2 Comando exit

O comando **exit** deve ser utilizado para encerrar uma sessão de trabalho, terminando a conexão com o usuário. O sistema volta a pedir uma nova conta.

1.6.3 Obtendo HELP no Sistema – Páginas de Manual

As páginas de manual acompanham todos os sistemas Unix/Linux. Elas trazem uma descrição básica do comando/programa e detalhes sobre o funcionamento de opção. Uma página de manual é visualizada na forma de texto único com rolagem vertical. Também documenta parâmetros usados em alguns arquivos de configuração.

A utilização da página de manual é simples, digite:

man [*seção*] [*comando/arquivo*]

- ***seção:*** É a seção de manual que será aberta, se omitido, mostra a *primeira* seção sobre o comando encontrada (em ordem crescente).
- ***comando/arquivo:*** Comando/arquivo que deseja pesquisar.

A navegação dentro das páginas de manual é feita usando-se as teclas:

- q - Sai da página de manual;
- PageDown ou f - Rola 25 linhas abaixo;
- PageUP ou w - Rola 25 linhas acima;
- SetaAcima ou k - Rola 1 linha acima;
- SetaAbaixo ou e - Rola 1 linha abaixo;
- p ou g - Inicio da página;
- h - Ajuda sobre as opções da página de manual;
- s - Salva a página de manual em formato texto no arquivo especificado (por exemplo: */tmp/ls*).

Cada seção da página de manual contém explicações sobre uma determinada parte do sistema. As seções são organizadas em diretórios separados e localizadas no diretório */usr/man*.

1.6.4 Desligando o sistema

Para evitar danos ao sistema de arquivos, é necessário que o super usuário pare o sistena antes de desligar o computador. Um dos comandos que podem ser utilizados é o comando **shutdown**. Este comando permite tanto desligar quanto reiniciar o computador.

```
[root@teste /root]# shutdown –h now
```

O comando acima permite desligar o computador imediatamente, enviando uma mensagem a todos os usuários que estão utilizando o sistema.

O comando **halt** diz ao sistema que ele deverá desligar imediatamente.

```
[root@teste /root]# halt
```

Para reinicializar o sistema, pode-se utilizar, além do comando **shutdown**, o comando **reboot:**

```
[root@teste /root]# shutdown –r now
```

Esta opção finaliza todos os processos e reinicia o computador.

```
[root@teste /root]# reboot
```

O comando **reboot** chama o comando **shutdown** e ao final deste reinicia o sistema.

Após executar os comandos deve-se aguardar até que o sistema esteja parado (com a mensagem “O sistema está parado”) para então poder desligar seu computador ou esperar que ele reinicie. Existe ainda uma outra alternativa que dispensa a senha de acesso do super usuário: basta você pressionar a seguinte combinação de teclas:**Ctrl+Alt+Del** e o computador reiniciará.

1.6.5 Listando Arquivos

O comando **ls** mostra o conteúdo de um diretório. O formato do comando é o seguinte:

ls [opções] [dir]

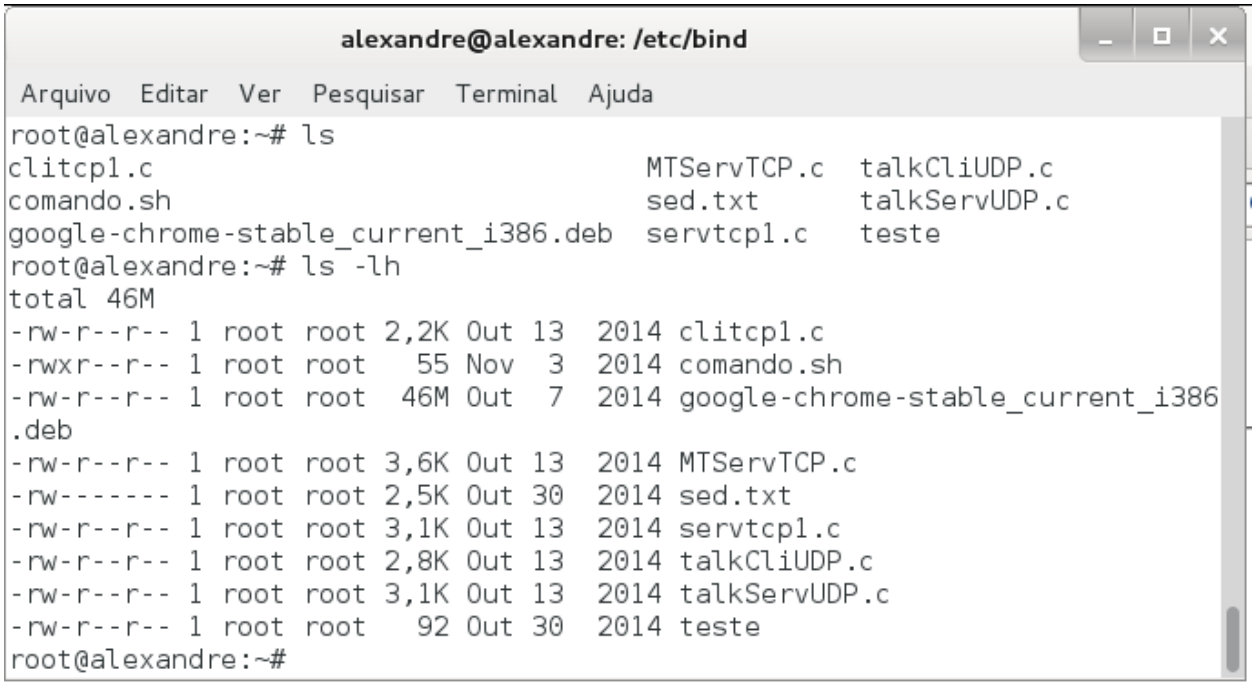
Existem várias opções, na qual, pode ser visualda com o comando:

```
[root@teste /root]# ls --help
```

Tabela com algumas opções:

Opção	Descrição
-l	Usa o formato de lista longa (com mais detalhes)
-h	Mostra de uma forma mais compreensível. É possível ver os tamanhos em KB, MB, GB, e assim por diante.
-a	Inclui na listagem os arquivos ocultos

Exemplos:



Metacaracteres

Existem sinais chamados metacaracteres, usados para facilitar a utilização de comandos no Linux. Quando se trabalha com comandos de manipulação de arquivos, frequentemente é útil empregarmos metacaracteres. Estes símbolos como: *, ?, [], {}; são úteis para se referenciar arquivos que possuem características em comum.

Para os exemplos dados, será usada a seguinte lista de arquivos:

```
[root@teste /root]# ls
12arquivo
1arquivo
```

```
2arquivo
arquivo
arquivo3
arquivo34
arquivo4
arquivo5arquivo
```

O asterisco “*”: O asterisco é usado para representar qualquer quantidade de qualquer caractere. Poe exemplo, **arquivo*** retornaria todos os arquivos em que o nome iniciasse com “arquivo”. Veja o efeito da utilização prática deste metacaractere.

```
[root@teste /root]# ls arquivo*
Arquivo     arquivo3    arquivo34   arquivo4    arquivo5arquivo
```

O Ponto de Interrogação “?”: É utilizado para representar um único e qualquer caractere. Ao digitar **ls arquivo?**, o usuário estará pedindo a lista de arquivos cujos nomes são iniciados por “arquivo” e terminem com um único caractere qualquer. Como no exemplo que segue:

```
[root@teste /root]# ls arquivo?
arquivo3    arquivo4
```

1.6.6 Criando e Removendo Diretórios

O comando **mkdir** é usado para criar diretórios. A sintaxe do comando será mostrado a seguir:

```
mkdir [parâmetros] nome_dir
```

A linha de comando a seguir cria um diretório:

```
[root@teste /root]# mkdir meu_diretorio
```

O comando também pode ser usado para criar uma árvore de diretórios:

```
[root@teste /root]# mkdir –p a/b/c
```

Também é possível criar vários diretórios simultaneamente, simplesmente colocando vários nomes de diretórios junto com o comando, por exemplo:

```
[root@teste /root]# mkdir dir_1 dir_2 dir_3
```

O comando **rmdir** é usado para remover diretórios. Por exemplo: para remover o diretório `meu_dir` basta executar o seguinte comando:

```
[root@teste /root]# rmdir meu_dir
```

O comando também pode remover árvores de diretórios. Para tal, utiliza-se o parâmetro [-p], como por exemplo:

```
[root@teste /root]# rmdir –p temp/sub_dir/sub_dir_2
```

O comando remove diretórios e não arquivos, se existir algum arquivo dentro do diretório este não será removido. Para conseguir remover diretórios com arquivos deve-se utilizar o comando **rm**, que veremos em outra ocasião.

1.6.7 Copiando Arquivos

O comando **cp** é utilizado para efetuar a cópia de arquivos no Linux. Sua sintaxe é mostrado a seguir:

```
cp [parâmetros] arquivo_original [destino]
```

Algumas observações importantes sobre cópia de arquivos:

- Copiar um arquivo para outro diretório onde já existe outro arquivo com o mesmo nome, ele será sobrescrito;
- Não é permitido copiar um arquivo para outro diretório que possui um diretório com o mesmo nome do arquivo a ser copiado, pois no Linux um diretório também é um arquivo;
- O arquivo será renomeado quando copiar um arquivo especificando um outro nome para o arquivo destino.

Tabela de opções

Opção	Descrição
-a	Preserva as permissões do arquivo_original quando possível
-b	Faz backup de arquivos que serão sobrescritos
-i	Solicita confirmação antes de sobrescrever arquivos
-R	Copia diretórios recursivamente, ou seja, toda a árvore abaixo do diretório de origem. O destino sempre será um diretório

Iniciando pela forma mais simples do comando, vamos copiar um arquivo para um novo arquivo. O comando pode ser visto a seguir:

```
[root@teste /root]# cp doc.txt aluno.txt
```

Neste caso ocorre a criação do arquivo `aluno.txt` a partir do arquivo `doc.txt`. Também é possível copiar para outro local como será feito a seguir:

```
[root@teste /root]# cp doc.txt /tmp
```

Como não foi mencionado o nome do arquivo de destino, será criado com o mesmo nome do atual. È sempre bom ter cuidado no uso do comando **cp**; para tal será usado o parâmetro `-i`. A menos que se utilize essa opção, o comando **cp** irá sobrescrever os arquivos existentes, como teremos a seguir:

```
[root@teste /root]# cp -i doc.txt /tmp
cp: sobrescrever '/tmp/doc.txt'? y
```

Um método seguro também seria usar o parâmetro `-b` (backup), quando o **cp** encontra um arquivo com o mesmo nome cria uma cópia acrescentando um "~" ao nome do arquivo. Como pode-se observar a seguir:

```
[root@teste /root]# cp -b doc.txt /tmp

[root@teste /root]# ls /tmp/doc.txt*
doc.txt doc.txt~
```

Podemos copiar vários arquivos simultaneamente, basta colocar os nomes dos arquivos a copiar logo depois do comando:

```
[root@teste /root]# cp a1 a2 a3 a4 /tmp
```

Lembrando que o último nome é o destino.

1.6.8 Movendo ou Renomeando Arquivos

Utilizamos essas funções para organizar informações dentro do sistema, para isso utilizamos o comando **mv**, sua sintaxe é:

```
mv arquivo [destino]
```

O comando **mv** é basicamente utilizado para mover um arquivo dentro do sistema de arquivos do Linux.

```
[root@teste /root]# mv aluno.txt /tmp
```

O comando acima move o arquivo `aluno.txt` para o diretório `/tmp`. É possível também usar o comando para renomear arquivos, como por exemplo:

```
[root@teste /root]# mv doc.txt documento.txt
```

Para não sobrescrever arquivos deve-se utilizar o parâmetro `-i`:

```
[root@teste /root]# mv -i doc.txt documento.txt
mv: sobrescrever 'documento.txt'? y
```

Para tornar este processo mais seguro ainda pode-se usar o parâmetro de backup `-b` para criar uma cópia do arquivo:

```
[root@teste /root]# mv -bi doc.txt documento.txt
mv: sobrescrever 'documento.txt'? y

[root@teste /root]# ls documento.*
documento.txt documento.txt~
```

Pode-se também renomear um arquivo durante a sua movimentação:

```
[root@teste /root]# mv documento.txt /tmp/documento2.txt
```

Agora temos um exemplo de um diretório:

```
[root@teste /root]# mv dir_1 dir_2
```

O commando acima move toda a estrutura do diretório `dir_1` para o diretório `dir_2`, mas isso acontece apenas quando o diretório `dir_2` não existe.

Por fim, pode-se movimentar vários arquivos ao mesmo tempo, conforme temos a seguir:

```
[root@teste /root]# mv a1 a2 a3 a4 dir_2
```

1.6.9 Removendo Arquivos

O comando **rm** é usado para remover arquivos e diretórios. É possível remover vários arquivos simultaneamente, basta colocar o nome dos arquivos a remover. A sintaxe do comando é:

```
rm [parâmetros] arquivo
```

Como primeiro exemplo tem-se o comando para apagar o arquivo `documento.txt`:

```
[root@teste /root]# rm documento.txt
```

Para excluir vários arquivos podemos fazer da seguinte maneira:

```
[root@teste /root]# rm documento.txt doc.txt documento2.txt
```

A maneira mais segura de utilizar o comando **rm** é com o parâmetro `-i`, para solicitar uma confirmação antes da exclusão:

```
[root@teste /root]# rm -ri /tmp
```

No comando acima, além de usar o parâmetro `-i` foi usado também o parâmetro `-r` que remove todos os arquivos do diretório `/tmp` de forma recursiva. Para cada arquivo excluído será pedido uma confirmação.

Podemos retirar a confirmação utilizando o parâmetro `-rf`, desta forma ele força a remoção de toda a árvore de diretórios:

```
[root@teste /root]# rm -rf /tmp
```

É possível utilizar os metacaracteres nos comandos, conforme exemplo a seguir:

```
[root@teste /root]# rm *.txt *.doc
```

1.6.10 Navegação entre Diretórios

Mudar de diretórios é fácil, é necessário saber em qual diretório você está, para isso utilize o comando **pwd** (o diretório mostrado será o seu diretório atual), isso se relaciona com o local aonde se pretende ir.

Para mudar de diretórios use o comando **cd**. Digitando apenas este comando você sempre retornará ao seu diretório home; para mover-se para outro diretório requer um caminho, você pode usar caminhos absolutos ou relativos.

Os caminhos absolutos começam na raiz do sistema de arquivos representada pela / seguida de seus subdiretórios; os caminhos relativos têm como partida o diretório atual só podendo ser abertos seus subdiretórios. A seguinte árvore de diretório ilustra como o comando cd trabalha:

```
/
/diretorio1
/diretorio1/diretorio2
/diretorio1/diretorio2/diretorio3
```

Se você estiver atualmente no diretório3 e quiser ir para o diretório1, e executar o seguinte comando:

```
cd diretório1
```

Será apresentada uma mensagem de erro que explica que não há nenhum subdiretório com esse nome. Isto ocorre porque não há nenhum diretório1 dentro do diretório3.

Para mover-se até diretório1, digite: **`cd /diretorio1`**

Este é um exemplo de um trajeto absoluto. Pois começamos na raiz da árvore do diretório (/) para mudar para o diretório1. Um trajeto é absoluto se o primeiro caracter for a /. Quando não utilizar a (/), será um caminho relativo.

Usar caminhos absolutos permite que você mude de um diretório para outro que esteja em qualquer caminho, para isto é necessário saber digitar o caminho completo.

O comando **`cd ..`**, diz ao seu sistema para ir até ao diretório imediatamente acima do diretório atual. Para ir à dois diretórios acima do atual, utilize o comando **`cd ../..`** .

Tabela com algumas opções

Comando	Função
`cd`	Retorna ao seu diretório home.
`cd ~`	Também retorna para seu diretório home.
`cd /`	Abre o diretório principal (raiz) tendo acesso aos diretórios de todo o sistema.
`cd ..`	Com esse comando você muda para o diretório acima do diretório atual.

2. Links

São referências, atalhos ou conexões lógicas entre arquivos ou diretórios. Estas referências podem ser de dois tipos: *Hard links* (diretas) ou *Symbolic links* (simbólicas).

2.1 Hard Links

A quantidade de *links* fazendo referência ao mesmo arquivo pode ser vista usando o comando **ls –l**. O valor que aparece após as permissões é o número de conexões lógicas:

```
[root@teste teste]# ls -l
total 8
-rw-r--r--    2 root 503            31 Out 18 16:59 a1
-rw-r--r--    1 root 503             0 Out 18 16:56 a4
-rw-r--r--    1 root 503             0 Out 18 16:34 doc.txt
-rw-r--r--    2 root 503            31 Out 18 16:59 teste
```

Neste caso o arquivo `a1` possui 2 links associados a ele. Quando qualquer um dos *links* é alterado, o original também é modificado. Não é possível criar *hard links* para diretórios. O formato do comando para criar um *hard link* é:

```
ln [arquivo] [link]
```

2.2 Symbolic Links

O *link* simbólico é como um atalho para um arquivo. O ato de se apagar um *link* simbólico não faz com que o arquivo original também apague, somente o vínculo será apagado.

Um exemplo de um *link* simbólico é mostrado a seguir:

```
[root@teste etc]$ ls -l rc.local
lrwxrwxrwx    1 root     0          13 Set 20  2005 rc.local -> rc.d/rc.local
```

Acima temos o arquivo `rc.local que é um link simbólico do arquivo rc.local localizado em /rc.d`. Podemos notas que antes das permissões dos arquivos, temos a letra “l” que indica que o arquivo em questão é um *link* simbólico. A sintaxe do comando para criação de *links* simbólicos é:

```
ln -s [arquivo] [link]
```

A única diferença nos comandos de *links* diretos e simbólicos é o uso do parâmetro `-s`.

3. Permissões de Arquivos

O sistema de arquivos do Linuxpossibilita que sejam atribuídos direitos de acesso diferenciado para os usuários do sistema. A cada arquivo ou diretório do sistema é associado um proprietário, um grupo e suas respectivas érmissões. Para verificar estes atributos podemos utilizar o comando **ls –l,** como no exemplo abaixo:

```
drwxr-xr-x   2 root        root         1024 Dec 23 15:22 bin
drwxr-xr-x   2 root        root         1024 Dec 31 05:48 boot
drwxr-xr-x   2 root        root         1024 Dec  6 15:51 cdrom
-rw-rw-r--   1 tecnologia  tecnologia   2342 Mar 10 03:12 teste.txt
```

No exemplo acima todos os arquivos e diretórios têm como dono o usuário e grupo **root**, com exceção do arquivo **`teste.txt`** cujo dono e grupo é **`tecnologia`**. Como você pode ver do lado esquerdo de cada arquivo/diretório existe uma série de letras r, w, x ou d, vamos ver o que representa cada uma delas:

```
drwxrwxrwx
0111222333
```

No caso acima, a primeira coluna significa (numero 0) se o nome listado é um diretório ou não, caso não seja um diretório ele será exibido o caractere "**-**", conforme mostrado no arquivo **teste.txt**.

Também há casos em que no lugar do **'d'**, aparecem outras letras que indicam outros tipos de arquivos. A letra 'l' significa que é um link simbólico, as letras 'c' e 'b' correspondem àos dispositivos (/dev). Veja alguns exemplos abaixo:

```
crw-rw----   1 alex     audio     1408 Dec  2 15:48 audio
brw-rw----   2 alex     cdrom     2200 May  5 16:41 hdd
lrwxrwxrwx   1 root     root      1024 Oct 17 22:53 cdrom -> /dev/hdd
```

Os demais nove caracteres, divididos em três grupos de três caracteres cada, definem as permissões do dono do arquivo, dos membros do grupo a que está relacionado o arquivo e de outros usuários do sistema. As permissões de acesso aos arquivos são representadas pelas letras r, w e x, conforme detalhado na tabela abaixo:

Modo de acesso	Arquivo	Diretório
Leitura (r)	Examinar o conteúdo do arquivo	Listar arquivos do diretório
Escrita (w)	Modificar o conteúdo do arquivo	Alterar diretório
Execução (x)	Executar o arquivo	Pesquisar no diretório

Vamos analisar novamente os atributos do arquivo mostrado anteriormente:

```
-rw-rw-r--   1 tecnologia  tecnologia   2342 Mar 10 03:12 teste.txt
```

Para este arquivo, o usuário **tecnologia** possui permissões de **leitura e escrita**; assim como os membros do grupo **tecnologia**; os demais usuários do sistema (outros) possuem apenas permissão de **leitura** do arquivo.

3.1 Alterando permissões

O comando **chmod** permite alterar as permissões de um ou mais arquivos. Existem duas notações para se aplicar o comando: o modo simbólico e o octal. Somente o superusuário ou dono do arquivo podem executar esta operação. A sua sintaxe é:

```
chmod [opções] arquivos
```

No modo simbólico deve ser indicado quem será afetado (u, g, o, a) e qual, ou quais, permissões serão concedidas ou suprimidas conforme tabelas abaixo:

Símbolo	Descrição
u	Usuário ou dono do arquivo
g	Grupo do arquivo
o	Outros usuários
a	Todos os anteriores (u, g, o)

A tabela abaixo mostra as características dos operadores do chmod:

Operador	Descrição
+	Concede a permissão especificada
-	Remove a permissão especificada

Para as permissões simbólicas do chmod temos:

Permissão	Descrição
r	Permissão de leitura
w	Permissão de escrita
x	Permissão de execução

A segunda forma de alterar permissões consiste em definir uma seqüência de três algarismos octais. Ela é mais utilizada quando se deseja alterar todas as permissões. Cada algarismo se refere a um grupo de permissões. Para facilitar o entendimento, associa-se um valor decimal para cada permissão, conforme tabela a seguir:

Decimal Associado	Permissão
4	Leitura (read)
2	Escrita (write)
1	Execução (execute)

Para se obter o octal referente às permissões selecionadas, deve-se executar uma operação de soma estre elas. Abaixo um exemplo:

```
chmod 651 documento.txt
```

O comando acima concedeu permissão de leitura e gravação ao dono (rw = 4+2), leitura e execução para o grupo (rx = 4+1), e execução para qualquer outro usuário (x = 1).

Agora um exemplo utilizando permissão simbólica:

```
chmod u+rw,g+x documento.txt
```

No caso acima são concedidas permissões de leitura e gravação ao dono e execução ao grupo.

3.2 Padrão de permissões

O comando **umask** é o comando que define as permissões padrão dos arquivos quando são criados pelo usuário.

`umask [opções] modo`

O parâmetro **modo** informa as permissões que serão dadas ao usuário/grupo/outros. Ele pode ser informado de duas maneiras:

- Como um número octal (022)
- Como uma máscara semelhante à utilizada pelo comando **chmod** (u=rw, g=r, o=x).

Assim, consegue-se controlar automaticamente as permissões dos arquivos que são criados pelo usuário. Exemplo:

```
             rwx rwx rwx                          rwx rwx rwx
umask=022    000 010 010          umask=133       001 011 011
             ===========                          ===========
             rwx r-- r--                          rw- r-- r--
```

Você deve oberservar que o bit 0 liga uma permissão e que o bit 1 desliga uma permissão. Mas você deve ter notado que o bit "x" foi desligado, mesmo tendo a permissão 0. Isso é uma proteção do Linux, que não deixa criar nenhum arquivo com a permissão de executável. Se você quiser criar um arquivo executável deverá fazê-lo através do comando chmod. Isso é uma proteção muito boa, pois evita que vírus ataquem o sistema.

3.3 Alterando Dono e Grupo dos Arquivos

O comando **chown** permite ao **root** a alteração do dono e do grupo relacionado ao arquivo ou arquivos. E possível também alterar o grupo do arquivo ou arquivos.

chown [novo_dono][:novo_grupo] arquivos

Exemplo1: chown :grupo2 documento.txt
Altera o grupo do arquivo documento.txt para **grupo2.**

Exemplo2: chown aluno:grupo2 documento.txt
Altera o dono do arquivo documento.txt para **aluno** e o grupo para **grupo2.**

4. Visualizando Arquivos

4.1 O comando cat

É utilizado para criar um arquivo qualquer. Inicialmente, o comando **cat** serve para mostrar o conteúdo de um arquivo no terminal, mas ele será utilizado aqui em conjunto com o redirecionador de saída **>**, direcionando a entrada do teclado para um arquivo. Para criar um arquivo com o comando **cat** digite o seguinte:

```
cat > teste.txt
```

Agora digite qualquer texto, quando terminar pressione **Ctrl + d** numa linha vazia para finalizar a entrada. Veja como fica:

```
cat > teste.txt
isto é um simples teste de criação de arquivo
```

Experimente digitar **cat teste.txt**; o comando **cat** irá lhe mostrar o conteúdo do arquivo.

```
cat teste.txt
isto é um simples teste de criação de arquivo
```

Além de monstrar o conteúdo de um arquivo o comando **cat** serve também para numerar as linhas de um arquivo. Para numerar as linhas de um arquivo ignorando linhas em branco utilize o parâmetro **–b;** para não ignorar linhas em branco, o parâmetro **–n**. Abaixo temos um exemplo:

```
[root@teste etc]$ cat -b teste.txt
     1  isto é um simples teste de criação de arquivo

     2  testando espaços entre linhas

[root@teste etc]$ cat -n teste.txt
     1  isto é um simples teste de criação de arquivo
     2
     3  testando espaços entre linhas
```

4.2 O comando more

No comando **cat** caso o tamanho do arquivo seja maior que o número de linhas da tela, ele irá mostrá-lo continuamente, sem parar, até que chegue ao final do arquivo.

Uma alternativa que resolve o problema de parada na tela é o comando **more**. Este comando interrompe a exibição do arquivo quando é preenchido uma tela completa, usualmente 25 linhas, pedindo a interferência do usuário para poder continuar. Será alterado o exemplo para que fique mais interessante a utilização do comando **more**, replicando a linha "isso é um teste ..." o suficiente para que preencha uma tela; na seqüência, é mostrada somente a parte da tela na qual é solicitada a interação do usuário.

```
...
isto é um simples teste de criação de arquivo
isto é um simples teste de criação de arquivo
```

```
isto é um simples teste de criação de arquivo
--Mais--(53%)
```

Repare que na última linha é exibida a porcentagem do arquivo que já foi percorida pelo **more**. Existem várias funcionalidades neste comando, dentre elas o avanço linha a linha (usando a tecla **Enter**) e o avanço página a página (usando a tecla **barra de espaços**).

4.3 O comando less

Uma alternativa ao uso do comando **more** seria o uso do comando **less**, que implementa a mesma funcionalidade que **more** e mais algumas, como a possibilidade de rolar a tela para cima e para o lado quando o texto ocupa mais de oitenta colunas.

A utilização dos comandos **less** e **more** se faz de maneira semelhante, a única diferença é que o comando **less** não finaliza automaticamente quando o fim do arquivo de texto é encontrado, sendo necessário pressionar a tecla **q (quit)**.

5. Pipe e Redirecionamento

Os comandos quando usados isoladamente têm uma função muito restrita, todavia se tornam extremamente poderosos quando usados em conjunto.

O conciteo de se trabalhar com pipes (“|”) é: a saída de um comando, em vez de ser enviada para a saída padrão, é redirecionada (através do pipe) para o comando subseqüente, tornando-se sua entrada.

Exemplos:

```
[root@teste etc]$ ls -l | less
```

O comando **ls –l** normalmente lista os arquivos e subdiretórios de um determinado diretório, contudo observa-se a utilização de um pipe (“|”), que redireciona esta saída para ser usada como entrada do comando **less**.

Como resultado do comando acima, será listado os arquivos e subdiretórios e caso ultrapasse 25 linhas, é dado uma pausa solicitando para o usuário pressionar a respectiva tecla para continuar.

```
[root@teste etc]$ cat sendlog.pl | wc -l
     99
```

Acima temos outro exemplo utilizando pipes, neste caso a saída do comando **cat** (visualizar o arquivo sendlog.pl) é enviada como entrada para o comando **wc** (que tem a função de contar o número de linhas do arquivo), o resultado é **99**, ou seja, o arquivo sendlog.pl tem 99 linhas.

```
[root@teste etc]$ ls a* | wc -l
      8
```

Neste caso, o comando **wc** conta a quantidade de arquivos ou subdiretórios que iniciem com **a**, o resultado é **8** arquivos.

No ambiente operacional do Linux, é convencionado que a entrada padrão é o teclado e a saída padrão, é o monitor de vídeo. Como já vimos na utilização dos pipes, para determinados fins é útil redirecionar a saída ou a entrada de dados.

Por exemplo, para redirecionar o resultado do comando **ls a*** para um arquivo em vez de apresentar seu conteúdo na tela, utiliza-se então o sinal “>”, como no exemplo a seguir:

```
[root@teste etc]$ ls a* > lista.txt
[root@teste etc]$ cat lista.txt
apt-0.5.5cnc6-fr1.i386.rpm
arquivo.old
arquivo.txt
aula.txt
```

Observe que nada apresentará no monitor porque foi criado um novo arquivo chamado **lista.txt** contendo o resultado do comando **ls a***, onde pode-se observar no comando **cat** logo em seguida.

```
[root@teste etc]$ cat arquivo.txt
teste com redirecionamento
[root@teste etc]$ cat arquivo.txt > exemplo.txt
[root@teste etc]$ cat exemplo.txt
teste com redirecionamento
```

Neste outro exemplo, a saída do comando **cat** é redirecionada para um arquivo novo chamado **exemplo.txt**. Observa-se que o comando **cp arquivo.txt exemplo.txt** teria o mesmo efeito.

6. Manipulando Arquivos

6.1 O comando tac

É um comando derivado do **cat**, o tac (propositalmente, **cat** de trás para frente) lista o arquivo de forma inversa. Veja a seguir a sintaxe do comando:

```
tac [opções] arquivos
```

```
[root@teste etc]$ tac cidades.txt
```

Exibe o arquivo **cidades.txt** de trás para frente, ou seja, a última linha será exibida primeiro, a penúltima linha em segundo, e assim, sucessivamente.

6.2 O comando sort

O comando **sort** classifica as linhas dos arquivos especificados segundo as opções selecionadas. Sua sintaxe é:

```
sort [opções] arquivos
```

Abaixo temos a tabela de opções:

Opção	Descrição
-b	Ignora os espaços e tabulações.
-d	Classifica em ordem de dicionário.
-f	Ignora sentenças entre maiúsculas e minúsculas.
-r	Ordem reversa de classificação.
-t c	Usa o caractere definido por c como separador de campos.
-u	Não repete linhas idênticas do texto.

Exemplos:

```
[root@teste etc]$ sort –bd cidades.txt
```

Ordena o arquivo **cidades.txt** ignorando os espaços e tabulações e classificando em forma de dicionário.

```
[root@teste etc]$ sort –ur cidades.txt
```

Ordena o arquivo **cidades.txt** sem repetir linhas idênticas e de forma reversa.

6.3 O comando wc

O comando **wc** faz a contagem de palavras e/ou linhas e/ou caracteres de determinado(s) arquivo(s). Se for especificado mais de um arquivo, é apresentado um total que é a somatória dos resultados de contagem de todos eles. Se não for colocada nenhuma opção, o padrão é apresentar todas as informações. Sua sintaxe é:

```
wc [opções] arquivos
```

Abaixo temos a tabela de opções:

Opção	Descrição
-l	Conta as linhas do arquivo especificado.
-c	Conta os caracteres do arquivo especificado.
-w	Conta as palavras do arquivo especificado.

Exemplo:

```
[root@teste etc]$ wc –lw cidades.txt
```

Apresenta a contagem das linhas e palavras do arquivo **cidades.txt**.

6.4 O comando head

O comando **head** envia para a saída padrão as primeiras linhas de um arquivo, conforme especificado nas opções. Sua sintaxe é:

```
head [opções] arquivos
```

Abaixo temos a tabela de opções:

Opção	Descrição
-n	Define em n o número de linhas a serem enviadas para a saída padrão a partir do início do texto.
-v	Imprime o cabeçalho com o nome do arquivo.

Exemplos:

```
[root@teste etc]$ head cidades.txt
```

Exibe as 10 primeiras linhas (padrão) do arquivo **cidades.txt**.

```
[root@teste etc]$ head -25 –v cidades.txt
```

Exibe as 25 primeiras linhas do arquivo **cidades.txt** identificando o início com cabeçalho.

6.5 O comando tail

O comando **tail** envia para a saída padrão as últimas linhas de um arquivo, conforme especificado nas opções. Sua sintaxe é:

```
tail [opções] arquivos
```

Abaixo temos a tabela de opções:

Opção	Descrição
-n	Define em n o número de linhas a serem enviadas para a

	saída padrão a partir do final do texto.
-c n	Ao invés de linhas, envia os n últimos caracteres do arquivo.
-f	Continua indefinidamente tentando ler caracteres ao final do arquivo, assumindo que o arquivo está crescendo.

Exemplos:

```
[root@teste etc]$ tail -25 cidades.txt
```

Exibe as 25 últimas linhas do arquivo **cidades.txt**.

```
[root@teste etc]$ echo $PATH | tail -c10
```

Exibe os 10 últimos caracteres da variável PATH.

```
[root@teste etc]$ tail -f cidades.txt
```

Exibe as linhas que vão sendo acrescentadas ao arquivo **cidades.txt** a partir do momento que foi executado o comando.

6.6 O comando grep

O comando **grep** procura strings ou expressões regulares, que contém caracteres normais misturados com metacaracteres, dentro de um ou mais arquivos. Sua sintaxe é:

```
grep [opções] exp [arquivos]
```

Abaixo temos a tabela de opções:

Opção	Descrição
-i	Ignora distinção entre letras maiúsculas e minúsculas.
-l	Lista os nomes dos arquivos em vez de linhas individualizadas.
-v	Lista as linhas não correspondidas.
-help	Exibe as informações de uso.
--version	Exibe as informações da versão.

Exemplos:

```
[root@teste etc]$ grep "01" lista.txt
```

Lista todas as linhas que contenham "01" do arquivo **lista.txt**.

```
[root@teste etc]$ grep -v "aa" lista.txt
```

Lista todas as linhas que não contenham "aa" do arquivo **lista.txt**.

6.7 O comando find

O comando **find** efetua uma busca na árvore de diretórios seguindo as condições e critérios especificados. Sua sintaxe é:

```
find [caminho] [expressão]
```

Abaixo temos a tabela de condições:

Condição	Descrição
-atime +n / -n / n	Procura arquivos que foram acessados há menos do que n(-n), mais do que n(+n), ou exatamente n (n) dias.

-ctime +n / -n / n	Procura arquivos que foram alterados há menos do que n(-n), mais do que n(+n), ou exatamente n (n) dias.
-exec comando {} \;	Executa comando para cada arquivo achado pelo find.
-name padrão	Procura arquivos que possuam um nome que satisfaça o padrão especificado.

Exemplos:

```
[root@teste etc]$ find /home –name documento.txt
```

Procura na árvore de diretórios, a partir de **/home**, arquivos com nome de **documento.txt**.

```
[root@teste etc]$ find /usr –atime 2 –exec ls –l {} \;
```

Procura arquivos, a partir do diretório **/usr**, que tenham sido acessados há exatamente dois dias, e à medida que forem encontrados, lista-os mostrando seus atributos.

7. Bibliografia

Comandos Linux: Prático e Didático / Marco Agisander Lunardi. 1ª Edição – Rio de Janeiro: Ciência Moderna , 2006.

Linux: Servidores de Rede / Craig Hunt. 1ª Edição - Rio de Janeiro: Ciência Moderna, 2004.

Linux: Redes e Servidores – Guia Prático / Carlos E. Morimoto, 2ª Edição – Porto Alegre: GDH Press e Sul Editores, 2006.