



Linux
Essentials

Sumário

1	Editores de texto GNU/Linux	3
1.1	Nano	4
1.2	vim	5
1.2.1	Modos de comandos e de edição	6
1.2.2	Utilizando o vim	7
1.2.3	Alternando entre o modo de comandos e o modo de edição	7
1.2.4	Gerenciamento de arquivos	8
1.2.5	Automatização de tarefas	9
1.2.6	Navegar por entre os arquivos	9
1.2.7	Realizar buscas	10
1.2.8	Outros comandos úteis	10
1.2.9	O arquivo vimrc	11
1.3	Dicas LPI	11
1.4	Tópicos para revisão do capítulo	12
2	O que é um pacote?	13
2.1	Mas o que é um gerenciador de pacotes?	13
3	Repositórios	15

3.1	Adicionar o caminho de um CD ou DVD no repositório	17
3.2	Atualização da lista dos repositórios	17
4	Instalação de pacotes	18
4.1	Consultas com o apt-cache	19
4.2	Consultar informações com o aptitude	20
4.2.1	Verificar e corrigir falhas do comando apt	20
4.2.2	Atualização de pacotes	21
4.2.3	Excluir pacotes	21
5	yum	25
5.1	Configurar repositório no CentOS (/etc/yum.repos.d/)	26
5.1.1	Instalação de pacotes YUM	28
5.1.2	Remover pacotes YUM	28
5.1.3	Atualizar pacotes YUM	28
6	dnf	29
7	ZYpp	30
8	Tópicos para revisão do capítulo	32
9	Empacotadores e Compactadores de Arquivos	33
9.1	Sobre o material	33
9.2	Introdução	34
9.3	Tar	35
9.4	CPIO	37
9.5	GZIP	39

9.5.1	Família de comandos do gzip	41
9.6	BZIP2	41
9.7	Família de comandos do bzip2	43
9.7.1	xz	43
9.8	GUNZIP	46
9.9	BUNZIP2	47
9.9.1	unxz	47
9.10	Considerações sobre backup de dados	48
9.11	Tópicos para revisão do capítulos	49
10	Raio-X do seu Hardware e Gerenciamento de Processos	50
10.1	Raio-X do seu Hardware	50
10.1.1	Introdução	50
10.1.2	O subdiretório /dev	50
10.1.3	Arquivos de dispositivo	51
10.1.4	Dispositivos de armazenamento	52
10.1.5	udev	53
10.1.6	Sistema de arquivos	53
10.1.7	Estrutura básica	54
10.1.8	Estrutura básica	55
10.2	Processos no Linux	57
10.2.1	Principais propriedades de um processo	57
10.2.2	Estado	58
10.2.3	Recursos	58
10.2.4	Proprietário	59

10.2.5	Prioridade	59
10.2.6	Número de identificação	59
10.2.7	Listando processos	60
10.2.8	ps	60
10.2.9	pidof	65
10.2.10	pstree	65
10.2.11	top	66
10.2.12	htop	68
10.2.13	Controlando nível de execução dos processos	69
10.2.14	Alterando o comportamento dos processos	72
10.3	Dicas LPI	77
10.4	Tópicos para revisão do capítulo	77

1

Editores de texto GNU/Linux

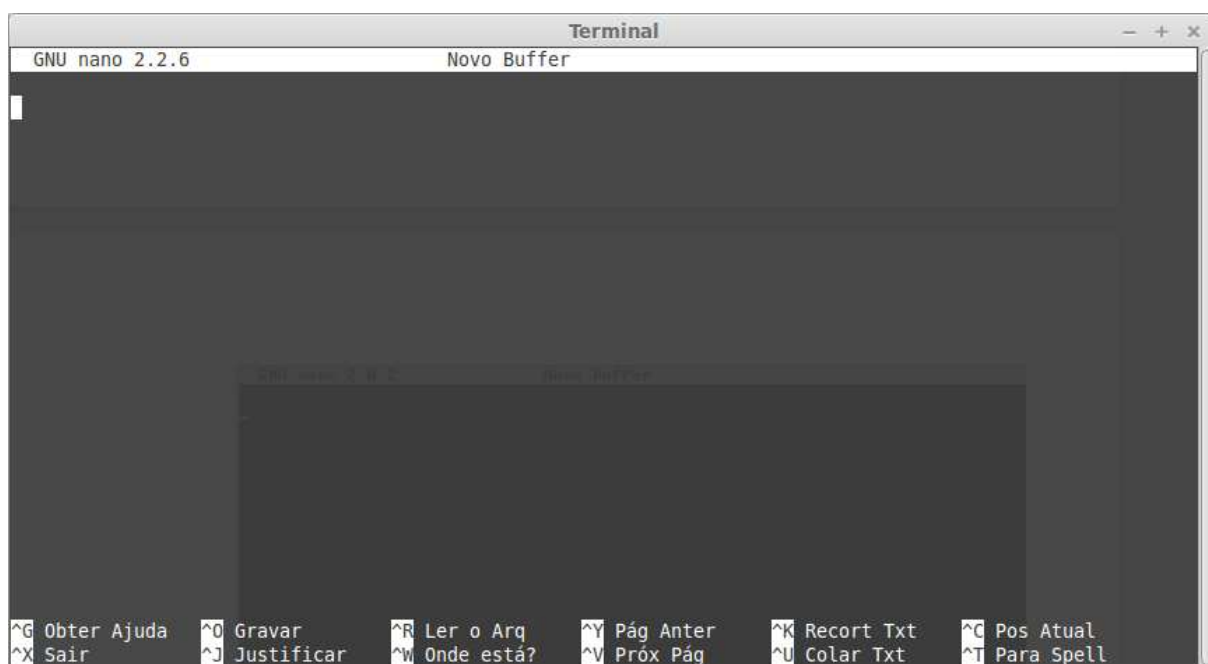
Vejamos uma breve descrição de alguns dos principais editores de texto existentes para sistemas GNU/Linux e suas principais características:

- **vi**: editor baseado em terminal disponível na maioria das distribuições GNU/Linux. É um dos mais conhecidos.
- **vim**: versão aprimorada do vi (Vi IMproved), trata-se de um dos editores mais utilizados atualmente.
- **nano**: o Debian e as distribuições baseadas nele, trazem o nano como editor de texto padrão. Muito prático, possui apenas o modo de edição.
- **joe**: é um editor de textos baseado em terminal muito simples e prático. Oferece uma grande quantidade de teclas de atalho para execução dos comandos.
- **pico**: parecido com o nano, é um editor simples que possui cinco ferramentas básicas: justificação de parágrafo, busca por caracteres, cópia e colagem de blocos, verificação ortográfica e um navegador de arquivos. Os comandos são enviados através de combinações de teclas de atalho.
- **mcedit**: o editor mcedit possui como diferencial a possibilidade de uso do mouse no ambiente de texto.
- **ed**: considerado um dos mais simples. Não permite a abertura de uma janela de edição. Usado para criar, exibir, modificar e manipular textos.
- **emacs**: originalmente criado por Richard Stallman, possui um interpretador do Emacs Lisp, que é um dialeto da linguagem de programação Lisp, permitindo a edição de textos.

Nano

O nano é um editor de textos simples e prático que segue a mesma linha de funcionamento do joe. Ele possui apenas o modo de edição e usa atalhos para gerenciar as funcionalidades. O nano vem instalado como padrão a partir do Debian etch e é um editor de textos que tem sido muito utilizado pelos usuários, pela praticidade e facilidade apresentada.

Basta entrar com o comando nano para abri-lo. Ao ser iniciado, seremos apresentados a uma tela como a seguinte:



As principais ações executadas no editor nano possuem atalhos que ficarão, por padrão, sempre exibidos no rodapé da página, como mostra a figura acima. Porém existem outros atalhos para facilitar o uso e gerenciamento dos arquivos por parte do nano.

O nano é carregado automaticamente em modo de edição, no qual, se nenhum arquivo for indicado para abertura com o mesmo, um novo arquivo poderá ser criado, e definido, posteriormente, seu nome e caminho.

O editor nano possui uma ajuda completa sobre atalhos que poderão ser utilizados no mesmo. Tal ajuda poderá ser acionada com a sequência de teclas CTRL + G.

Para abrir um arquivo de texto específico, utilizamos a sintaxe apresentada a seguir:

```
1 | nano <arquivo>
```

Para gerenciar arquivos com o nano, temos os seguintes atalhos que podem ser utilizados:

- CTRL + R: abre um arquivo especificado para edição ou insere o conteúdo de outro arquivo no arquivo atual.
- CTRL + X: fecha o arquivo atual; sai do nano.
- CTRL + O: salva o arquivo editado.

Para automatizar tarefas com o nano, os seguintes atalhos podem ser utilizados:

- CTRL + K: recorta a linha atual.
- ALT + ^: copia a linha atual.
- CTRL + U: cola a linha copiada ou recortada.

Para se movimentar dentro do arquivo, os seguintes atalhos podem ser utilizados:

- CTRL + Y: sobe uma página.
- CTRL + V: desce uma página.
- ALT + : avança até a primeira linha do arquivo.
- ALT + /: avança até a última linha do arquivo.
- CTRL + A: move o cursor para o início da linha.
- CTRL + E: move o cursor para o fim da linha.

Podemos utilizar o atalho CTRL + W para localizar uma palavra no arquivo.

Vejamos a seguir outros atalhos importantes utilizados no nano:

CTRL + C: apresenta informações referentes à posição do cursor. CTRL + G: abre a ajuda dos comandos e uma descrição do editor. CTRL + J: deixa o texto do arquivo inteiro justificado. CTRL + T: se o comando spell estiver instalado, aciona o corretor ortográfico.

vim

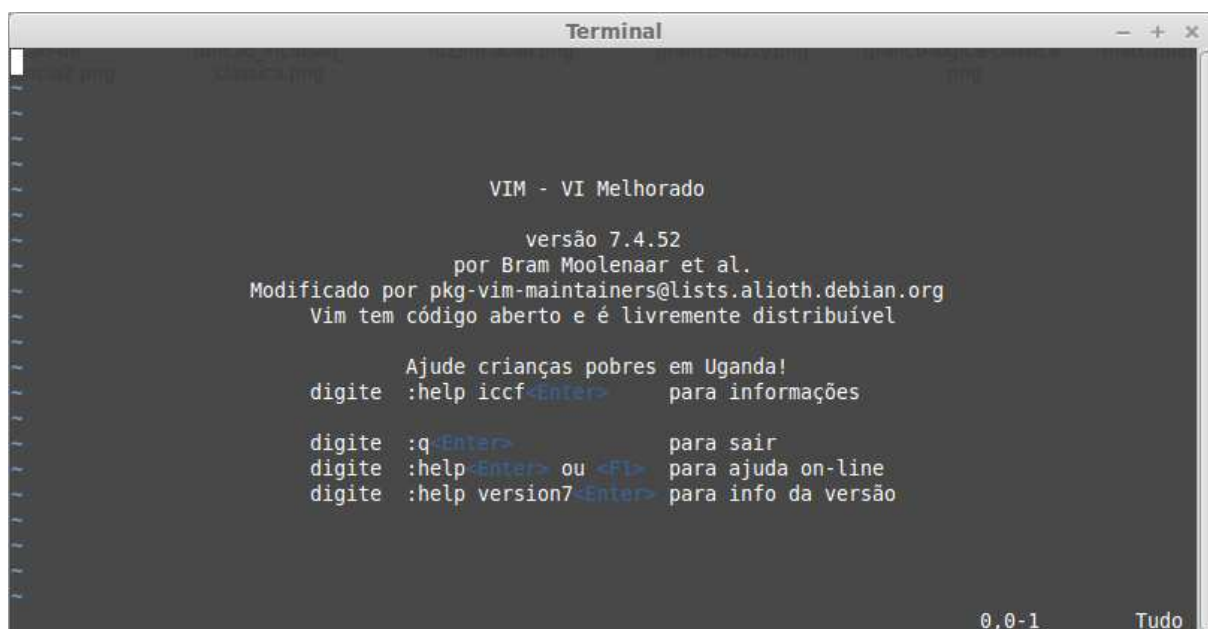
O editor de textos vim (Vi IMproved) é um dos editores de textos mais utilizados atualmente. É o sucessor do antigo vi, porém muitos sistemas criam alias (apelidos) para o vim, fazendo com que o mesmo seja referenciado pelo nome vi.

Sendo assim, o editor pode ser facilmente identificado simplesmente por vi, mas é importante lembrar que sua utilização é extremamente diferente, comparando as versões atuais do vim e o vi na versão antiga. Essa convenção de nomenclatura facilita seu acesso e utilização, uma vez que o antigo vi caiu em desuso.

Sua sintaxe é a seguinte:


```
1 | vim [opcoes] [arquivo]
```

A seguir, temos a tela inicial mostrada na figura abaixo do editor vim:



Um dos motivos que tornam o vim um editor de textos útil e prático é o seu formato e interface de comunicação final, que, para o principiante, é um pouco confusa, mas, após o seu entendimento, é facilitadora. O vim é um editor de textos que poderá ser utilizado diretamente na linha de comandos, porém, também existem versões do vim para quem deseja trabalhar com ele no ambiente gráfico e até versões para sistemas operacionais diferentes do GNU/Linux.

Outro recurso interessante do vim é o **Syntax Highlight**, que é uma coloração especial da sintaxe nos arquivos. Tais arquivos, como os de configuração ou código-fonte de várias linguagens, poderão receber uma coloração especial para sua melhor identificação.

Modos de comandos e de edição

Uma particularidade sobre o editor de textos vim, que é essencial para o nosso aprendizado, é entender internamente seus modos de trabalho, que são basicamente dois: **modo de comandos** e **modo de edição**.

Essa divisão é extremamente importante, pois define o funcionamento básico do vim e sua manipulação eficaz se dá pela boa alternância entre esses modos. Sobre os modos de utilização do vim temos:

- **Modo de comandos:** a maioria das teclas terá uma funcionalidade específica, como se cada uma delas fosse um determinado comando que executará uma tarefa. Ou seja, no modo de comandos, o vim é utilizado para manipular o arquivo, podendo executar

tarefas de automação, como copiar ou apagar uma sequência de linhas, ou tarefas de gerenciamento do próprio arquivo, como salvar o arquivo simplesmente ou salvá-lo com nome e caminho diferentes etc.

- **Modo de edição:** é exatamente neste modo que o vim trabalhará como um editor de textos comum, no qual as teclas terão suas funcionalidades reais e o texto poderá ser completamente redigido ou alterado para o formato desejado.

Saber alternar entre esses dois modos é essencial para executar qualquer tarefa utilizando o editor de textos vim.

Utilizando o vim

O vim poderá ser inicializado diretamente na linha de comandos, a partir da digitação da palavra vim simplesmente. Nesse modo de carregamento, o editor irá inicializar com um arquivo novo e ainda sem nome. Mas também é possível inicializá-lo com um arquivo já existente, nomeado e armazenado em um caminho completo, caso seja necessário, como no exemplo a seguir:

```
1 | vim /etc/apt/sources.list
```

Independentemente da forma como o vim foi inicializado, com um arquivo novo ou um arquivo existente, ele sempre será inicializado em modo de comandos, no qual o arquivo ainda não poderá ser definitivamente redigido ou alterado. Para tal função, é preciso alternar para o modo de edição, sobre o que falaremos a seguir.

A tabela a seguir descreve opções que podemos utilizar para iniciarmos o vim:

Comando	Descrição
+	Inicia o vim com o cursor no final do arquivo.
+ <linha>	Inicia o vim com o cursor na linha de número determinado.

Alternando entre o modo de comandos e o modo de edição

O vim é sempre iniciado, por padrão, em modo de comandos, porém é possível alternar tranquilamente para o modo de edição e, posteriormente, voltar para o modo de comandos, e vice-versa.

Entrando no modo de edição

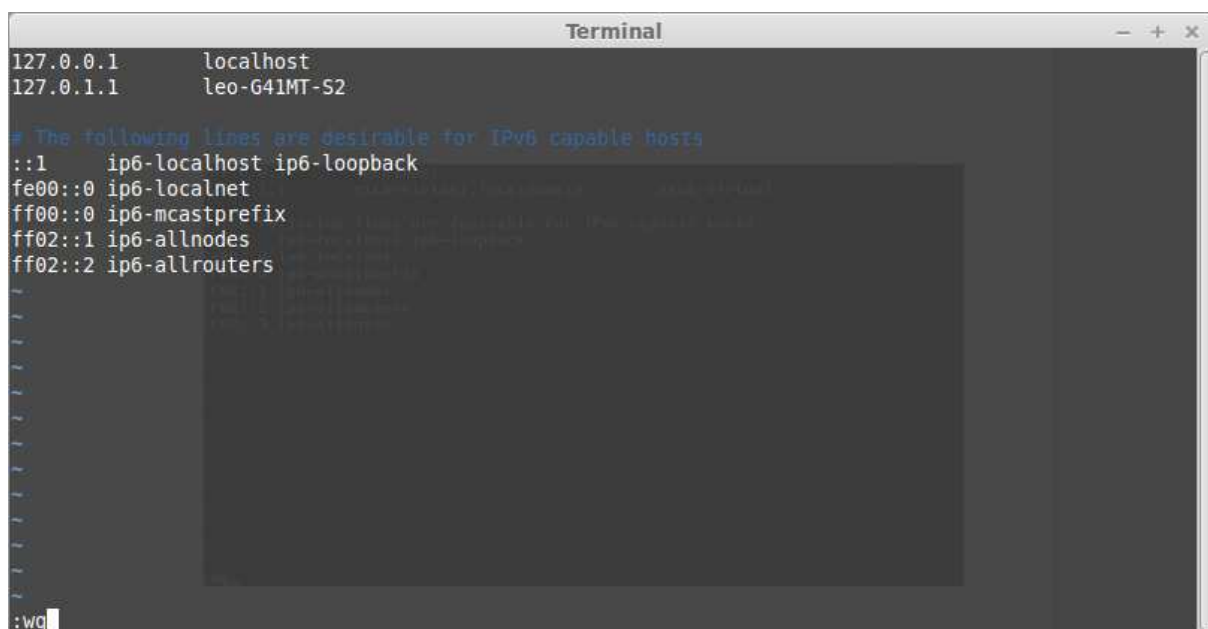
- **i / <INSERT>:** entra no modo de edição, mantendo o cursor na posição original.
- **a:** entra no modo de edição, movendo o cursor um caractere à direita de sua posição original.

- o: entra no modo de edição, inserindo uma nova linha abaixo da posição original do cursor.

Basta pressionar ESC para retornar ao modo original de comandos.

Gerenciamento de arquivos

Depois que um determinado arquivo foi criado ou editado com o vim, também é preciso conhecer alguns comandos para gerenciar o arquivo em si, por exemplo, salvando alterações feitas ou simplesmente descartando-as. No modo de comandos, temos a possibilidade de utilizar outros tipos de comandos, que serão sempre precedidos pelo caractere : (dois pontos). O comando passado junto com os dois pontos poderão ser visualizados no rodapé da página do vim, como no exemplo da figura abaixo:



A terminal window titled "Terminal" showing a vim editor session. The top part of the screen displays the contents of a file, which appears to be a hosts file with IPv6 entries. The bottom part of the screen shows the vim command line with the command `:wq` entered, indicating a save and quit operation. The command line is highlighted in blue.

```
127.0.0.1    localhost
127.0.1.1    leo-G41MT-S2

# The following lines are desirable for IPv6 capable hosts
::1          ip6-localhost ip6-loopback
fe00::0      ip6-localnet
ff00::0      ip6-mcastprefix
ff02::1      ip6-allnodes
ff02::2      ip6-allrouters

:wq
```

Comandos:

- :q: sai do vim. Alterações feitas no arquivo e que não foram salvas, gerarão um alerta com uma pergunta sobre o que deseja fazer com o arquivo.
- :w: salva as últimas alterações feitas.
- :w!: tem a mesma função do :w com a diferença de forçar a sobrescrição do arquivo.
- :wq: salva as últimas alterações feitas e sai do vim.
- :q!: sai sem salvar qualquer alteração feita no arquivo.
- :w <nome-arq>: salva o arquivo com o nome <nome-arq>. Como não foi especificado o caminho, o arquivo será salvo no diretório atual, no qual o vim foi carregado.
- :r <outro-arq>: insere o conteúdo do arquivo de nome <outro-arq> no arquivo atual, a partir da posição original do cursor.
- :e <arq-teste>: alterna a edição do arquivo atual para o arquivo de nome <arq-teste>.

Automatização de tarefas

Com o vim, também é possível automatizar tarefas de gerenciamento do arquivo, como copiar, apagar, recortar múltiplas linhas, inserir linhas etc. Vejamos:

- yy: copia a linha inteira na qual se encontra o cursor.
- <N>yy: copia as <N> próximas linhas, a partir da posição atual do cursor.
- Dd: apaga a linha inteira na qual se encontra o cursor.
- <N>dd: apaga as <N> próximas linhas, a partir da posição atual do cursor.
- Dgg: apaga a partir da linha em que se encontra o cursor, até o início do arquivo (primeira linha).
- dG: apaga a partir da linha em que se encontra o cursor, até o final do arquivo (última linha).
- X: apaga um único caractere, no qual o cursor se encontra.
- <N>x: apaga <N> caracteres, a partir da posição atual do cursor.
- cc: recorta a linha em que se encontra o cursor e armazena-a no buffer.
- <N>cc: recorta <N> linhas, a partir da posição atual do cursor, e as armazena no buffer.
- p: cola o trecho de linhas que foi copiado, apagado ou recortado.
- <N>p: cola <N> vezes o que estiver na área de transferência.
- V: aciona a seleção visual de linhas.
- J: junta a linha atual á seguinte.
- u: desfaz a modificação mais recente.
- U: desfaz todas as modificações feitas na linha em que se encontra o cursor.
- D: exclui todo o conteúdo da linha a partir da posição atual do cursor.
- r: substitui o conteúdo do início da linha em que o cursor estiver.
- r<caractere>: substitui o caractere sob o cursor pelo que for colocado no lugar de <caractere>.
- A: adiciona texto no fim da linha onde se encontra o cursor.
- O: insere linha acima da linha atual.
- CTRL + H: exclui o último caractere.
- :%s<string1><string2>g*: substitui <string1> por <string2>.

Navegar por entre os arquivos

Também existem comandos específicos para uma movimentação automatizada dentro do arquivo.

- gg: move o cursor para a primeira linha.
- G: avança para a última linha.
- <N>G: avança para a linha <N> especificada.
- \$: avança para o fim da linha atual.
- ^: leva o cursor para o primeiro caractere que não esteja em branco da linha atual.
- :<N>: avança para a linha <N>.
- CTRL + F: avança para a tela seguinte.

- CTRL + B: leva o cursor para a tela anterior.
- H: leva o cursor para a primeira linha da tela.
- h: leva o cursor um caractere à esquerda.
- M: leva o cursor para o meio da tela.
- L: avança para a última linha da tela.
- j: avança para a linha seguinte.
- k: leva o cursor para a linha anterior.
- l: avança um caractere à direita.
- w: avança para o começo da próxima palavra.
- W: avança para o começo da próxima palavra, separada por espaço.
- b: leva o cursor para o começo da palavra anterior.
- B: leva o cursor para o início da palavra anterior, separada por espaço.
- 0 (zero): leva o cursor para o começo da linha atual.

Realizar buscas

Para realizar buscas por palavras no vim, temos os seguintes comandos:

- <palavra>: faz uma busca, pelo texto inteiro, por aquilo que for colocado no lugar de <palavra>.
- ? <palavra>: leva o cursor até a ocorrência anterior à <palavra>.
- n: repete o comando / ou ? mais recente.
- N: repete, na direção inversa, o comando / ou ? mais recente.
- CTRL + G: exibe o nome do arquivo, o número da linha atual e o total de linhas do texto.

Outros comandos úteis

O vim também traz diversos comandos úteis à utilização, ao gerenciamento e à automatização do arquivo. A tabela a seguir descreve os principais deles:

- CTRL + R: refaz as últimas ações dos comandos desfeitos.
- .: repete o comando anterior.
- N: vai para o próximo resultado da pesquisa.
- :![comando]: executa um comando de nome [comando] e exibe o resultado no arquivo simplesmente, não inserindo tal resultado como conteúdo do mesmo.
- :r![comando]: executa um comando de nome [comando] e insere o resultado do mesmo no arquivo atual.
- :set nu: insere uma numeração de identificação das linhas no arquivo. Esta numeração é apenas visual, não sendo inserida no arquivo original.
- :set nonu: desabilita a numeração no arquivo.
- :help: inicializa a ajuda do vim.

O arquivo vimrc

O vim possui um arquivo de configuração, no qual poderão ser ativadas e desativadas diversas funcionalidades, como o recurso de coloração de sintaxe (*Syntax Highlight*). O arquivo de configuração do vim é o `/etc/vim/vimrc`.

Os comentários desse arquivo, ou seja, as linhas que serão ignoradas pelo arquivo de configuração, são linhas iniciadas com `"` (aspas). A linha que ativa o recurso de coloração é a linha com o conteúdo `syntax on`, que vem geralmente comentada, bastando simplesmente descomentá-la.

Ao descomentar as funcionalidades desejadas, o usuário já está configurando o vim, bastando copiar, com o nome de `.vimrc`, o arquivo modificado para seu diretório home.

```
1 | cp /etc/vim/vimrc ~/ .vimrc
```

Dicas LPI

O editor `od`, pode ser importante durante a realização da prova. Sua principal funcionalidade é mostrar a saída de um arquivo em formatos como octal, ASCII, entre outros.

As variáveis relacionadas com os editores podem ser valiosas na prova. Uma delas é a própria variável `EDITOR`.

Algumas dicas de `vi` são importantes para a prova LPI:

- `:set number` ou `set nu`
- `:syntax on`
- `:set ic`
- `:set hlsearch`
- `:set background=dark`

[illegible]

Tópicos para revisão do capítulo

Atente-se para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes deste capítulo.

- Os principais editores de texto existentes para sistemas GNU/Linux são o vim, o nano, o joe, o pico, o mcedit, o ed e o emacs;
- O nano é um editor de textos simples e prático que segue a linha de funcionamento do joe. Ele possui apenas o modo de edição e usa atalhos para gerenciar as funcionalidades;
- O vim (Vi IMproved) é um dos editores de texto mais utilizados atualmente, e que, além de útil e prático, possui um recurso interessante, o *Syntax Highlight*, que é uma coloração especial da sintaxe nos arquivos;
- O editor de textos joe é uma outra alternativa de edição de textos tradicional. O joe sempre trabalha no modo de edição e usa uma infinidade de teclas de atalho para executar comandos com funcionalidades específicas;
- Podemos alterar o editor de texto padrão por meio do comando `update-alternatives --config editor`.

2

O que é um pacote?

Os diversos programas para **GNU/Linux** são distribuídos em forma de pacotes específicos para cada distribuição.

Neste capítulo aprenderemos um pouco sobre esses pacotes e como instalá-los e removê-los do sistema. Pacotes são conjuntos de binários pré-compilados, bibliotecas, arquivos de controle e arquivos de configuração, que são instalados facilmente no sistema operacional.

Eles podem, eventualmente, conter sistemas de listagem/checagem de dependências e **scripts** para configurações pós instalação.

Os pacotes nos sistemas baseados em **Debian** têm uma extensão característica: `.deb`.

Já nas distribuições baseadas em **RedHat**, temos pacotes com a extensão característica: `.rpm`.

Mas o que é um gerenciador de pacotes?

Um gerenciador de pacotes é um sistema para a instalação, atualização e remoção de programas em uma distribuição **GNU/Linux**. Parece muito simples falar em instalação de pacotes, mas temos que lembrar que é o gerenciador de pacotes quem faz toda a parte “**suja**” para nós.

Um pacote nem sempre depende apenas dele mesmo, ou seja, quando instalamos um programa,

ele pode depender de bibliotecas de áudio, vídeo, imagens, funções e vários outros programas que precisam estar instalados antes do pacote. É esse elo entre programas que chamamos de **dependências**.

O trabalho feito pelo gerenciador de pacotes é interpretar a necessidade de cada um dos pacotes, para que eles possam ser instalados e/ou removidos. Para os sistemas baseados em **Debian**, a ferramenta a ser utilizada é o aptitude ou o apt-get.

Já para sistemas baseados em **RedHat** temos uma ferramenta análoga chamada yum.

3

Repositórios

Os gerenciadores `apt-get` e `aptitude`, que acabamos de mencionar, só podem automatizar a administração das dependências dos pacotes, caso a origem desses pacotes esteja especificada de forma correta nos arquivos chamados repositórios. Os repositórios poderão ser indicados por meio do arquivo `/etc/apt/sources.list`. Arquivos adicionais podem ir para o diretório `/etc/apt/sources.list.d/`.

A indicação de um repositório funcional e acessível é fundamental para o processo de download e instalação do pacote desejado. Um repositório nada mais é do que uma centralização de pacotes `.deb` disponíveis para download por meio do `apt` para instalação em um cliente. Repositórios podem ser representados por diversos meios e tipos, como:

- Repositórios de Internet acessíveis por meio de protocolos como `http` e `ftp`;
- Repositório local, disponibilizado por um servidor ou cliente da rede e acessível também pelos protocolos `http` e `ftp`;
- Repositórios representados por mídias específicas, podendo ser um CD, DVD, ou até mesmo uma determinada partição de um HD.

É interessante saber também que as distribuições possuem repositórios próprios, oficiais e não oficiais. Independentemente do tipo de repositório escolhido para obter pacotes, é preciso fazer suas respectivas indicações de uso, além de ser permitido utilizar vários tipos de repositórios ao mesmo tempo.

É por meio do arquivo `/etc/apt/sources.list` que os repositórios poderão ser configurados e indicados para uso. Cada linha desse arquivo referenciará um repositório e o tipo de pacote que poderá ser instalado no sistema. Dentro dos repositórios, existe uma divisão por sessões, indicando os tipos de pacotes acessíveis para instalação.

Um arquivo `sources.list` pode ser acessado por meio de um arquivo modelo em `/usr/share/doc/apt/examples/sources.list`, inclusive com caminhos para repositórios oficiais da Internet, que poderão ser utilizados para obter pacotes.

O arquivo `sources.list` terá a seguinte sintaxe para adicionar um repositório, indicando um protocolo de acesso em um respectivo endereço:

```
1 | <tipo_de_pacote> <protocolo_de_acesso://endereco> <versao_da_distro> <secao_de_pacotes>
```

Exemplo:

```
1 | deb http://debian.usp.br stable main contrib non-free
```

Sobre os campos do arquivo `sources.list` temos:

- **Tipos de pacote:** além dos pacotes com a extensão `.deb` trabalhados até o momento, também é possível obter o código-fonte desses pacotes para que possam ser instalados pelo processo de compilação (visto no início do capítulo). O código-fonte dos pacotes poderá ser acessado, caso o campo tipo de pacote seja identificado como `deb-src`. Se forem utilizados apenas os pacotes normais com a extensão `.deb`, basta identificar o campo tipo de pacote como `deb`;
- **Protocolo de acesso:** no campo endereço é indicado o protocolo de acesso a um servidor de Internet ou local, podendo ser `http` ou `ftp`, com seu respectivo endereço de acesso. Também é possível usar a indicação `file` para especificar que o repositório é um diretório do sistema com imagens `.iso` dos pacotes Debian;
- **Versão da distribuição:** é necessário indicar a versão dos pacotes que serão instalados, que está diretamente associada à versão de distribuição, atualmente em uso. Se a distribuição for uma versão `stable`, os pacotes `.deb` que serão instalados também deverão ser desta mesma versão, e assim por diante. Nesse campo, podemos encontrar as versões `stable`, `testing` e `unstable`; O codinome da distribuição também pode ser utilizado, ao invés da versão.
- **Sessão de pacotes:** na sessão de uso dos pacotes que serão instalados, é criada uma divisão para diferenciar os pacotes em relação às suas características de licença. São elas:
 - `main`: contém apenas pacotes de software livre;
 - `contrib`: contém pacotes livres, porém, que dependem de algum software proprietário;
 - `non-free`: contém pacotes com código-fonte fechado.

Com os campos devidamente informados, é possível configurar um repositório, a cada linha, no arquivo `sources.list`.

Adicionar o caminho de um CD ou DVD no repositório

Para adicionar um CD ou DVD como mídia de repositório, basta utilizar o comando `apt-cdrom` e, por meio da ação `add`, será possível ler todo o conteúdo de mídia inserido, criando uma lista de todos os pacotes nela presentes.

```
1 | apt-cdrom [opcoes] <acao>
```

A opção `add` adiciona um CD ou DVD como mídia de repositório.

Assim, é possível adicionar várias mídias diferentes como meio de repositório. Quando um pacote precisar ser instalado, o próprio `apt` solicitará a inserção da respectiva mídia para a cópia do pacote.

Atualização da lista dos repositórios

Os comandos `apt-get update` ou `aptitude update` são utilizados para atualizar a base de informações locais sobre os repositórios remotos. Sem essa atualização, novas versões de programas ou pacotes não serão encontradas, causando a perda de atualizações que podem ser importantes. Isso pode ser facilmente evitado, já que é possível até mesmo programar a execução periódica de um desses dois comandos e manter o sistema atualizado automaticamente.

4

Instalação de pacotes

O sistema de empacotamento apt-get usa um banco de dados próprio para saber quais pacotes estão instalados, quais não estão e quais estão disponíveis para instalação. O gerenciador usa esse banco de dados para instalar os pacotes solicitados pelo usuário e para saber quais pacotes são necessários para que o pacote selecionado seja executado perfeitamente. Já o aptitude é um front-end para o **Advanced Packaging Tool (APT)**. Ele exibe uma lista de pacotes de software e permite ao usuário escolher interativamente quais pacotes deseja instalar ou remover. Possui um sistema de busca especialmente poderoso utilizando padrões de pesquisa flexível.

Antes de instalar um programa, devemos procurar por ele. Para isso, utilizamos o apt da seguinte forma:

```
1 | apt-cache search <programa>
```

O comando apt nos traz uma saída mais amigável para o usuário como o progresso de status da instalação do pacote quando utilizado com o parâmetro install e ainda possui funções de outros comandos como o apt-cache search, sendo possível realizar a mesma busca de pacotes através do comando apt search.

Também, podemos utilizar o aptitude para verificar a existência do programa a ser instalado:

```
1 | aptitude search <programa>
```

No tópico anterior, aprendemos sobre a utilização dos comandos apt-get update e aptitude

update. Caso nenhum resultado apareça na busca pelos programas desejados, isso pode ter sido causado pelo fato de que os bancos de dados de informações não estão atualizadas, logo, o uso de um desses comandos é recomendado. É possível também que o programa simplesmente não exista nos repositórios especificados em `/etc/apt/sources.list`.

Verificada a existência do programa, podemos instalá-lo por meio da seguinte sintaxe:

```
1 | apt-get install <programa1> <programa2> ...
```

Vamos instalar o front-end aptitude:

```
1 | apt-get install aptitude
```

O processo será totalmente automatizado, além de o apt realizar o download automático do pacote e suas respectivas dependências a partir dos repositórios configurados, completando o processo com suas respectivas instalações. Algumas opções poderão ser utilizadas em conjunto com a ação `install`.

É possível, também, instalar vários pacotes ao mesmo tempo, pois o apt gerenciará a instalação de todos eles de forma tranquila. Porém, não é possível executar o comando `apt-get` com qualquer ação em paralelo.

Opções:

- `-d` ou `--download-only`: apenas faz o download, não instala os pacotes e as possíveis dependências.
- `-f` ou `--fix-broken`: tenta corrigir erros gerados na instalação de pacotes.
- `-s` ou `--simulate`: apenas simula o processo de instalação do pacote com suas respectivas dependências, não realizando sua instalação.
- `--reinstall`: reinstala um pacote, caso já esteja instalado.
- `-u` ou `--show-upgraded`: exibe uma lista dos pacotes que serão atualizados.
- `-y` ou `-yes`: assume a resposta Yes para todos os questionamentos.

Consultas com o apt-cache

Outro modo de fazer consultas na lista de repositório por determinados pacotes e suas respectivas descrições é utilizar o comando `apt-cache`.

Com a ação `search`, o pacote especificado será procurado na lista de repositórios e caso seja encontrada alguma palavra relacionada ao pacote desejado, será exibido o nome do pacote com uma simples descrição sobre ele. Para isso, utilizamos a seguinte sintaxe:

```
1 | apt-cache [opcoes] search <pacote>
```

O exemplo a seguir demonstra o uso do apt-cache com a ação search:

```
1 | apt-cache search apache2-mod-php5
2 | libapache2-mod-php5 - linguagem de scripts imersa em HTML, roda...
3 | php5-cgi - server-side, HTML-embedded scripting language (CGI binary)
4 | libapache2-mod-php5filter - server-side, HTML-embedded scripting language...
5 | php5-fpm - server-side, HTML-embedded scripting language (FPM-CGI binary)
```

Obtendo detalhes de um pacote no repositório:

```
1 | apt-cache [opcoes] show [pacote]
```

Após localizar um pacote de nome específico, é possível obter informações sobre ele, de forma detalhada. A ação show tem a finalidade de exibir uma descrição completa sobre um pacote da lista de repositórios.

Consultar informações com o aptitude

Também podemos utilizar o aptitude search, abordado anteriormente, para consultar pacotes. A seguir, temos um exemplo do seu uso:

```
1 | aptitude search apache2-mod-php5
2 | p  libapache2-mod-php5 - linguagem de scripts imersa em HTML
3 | p  libapache2-mod-php5:i386 - linguagem de scripts imersa em HTML
4 | p  libapache2-mod-php5filter - server-side, HTML-embedded...
5 | p  libapache2-mod-php5filter:i386 - server-side, HTML-embedded...
```

Verificar e corrigir falhas do comando apt

Por um motivo qualquer, certas vezes, podem ocorrer falhas na execução de um comando da família apt, principalmente na instalação ou atualização de pacotes, no momento de seu download. Para resolver problemas relacionados, é possível executar a ação check, que verifica arquivos corrompidos, fazendo uma correção automática dos mesmos e atualizando a lista de repositório.

```
1 | apt-get [opcoes] check
```

Segue um exemplo da execução do comando `apt-get` com a ação `check`:

```
1 | apt-get check
2 | Lendo listas de pacotes... Pronto
3 | Construindo árvore de dependências
4 | Lendo informação de estado... Pronto
```

Atualização de pacotes

Para atualizar um programa que já esteja instalado, usa-se o comando `apt-get upgrade`. Sua sintaxe é a seguinte:

```
1 | apt-get [opcoes] upgrade <pacote>
```

Veja as principais opções que podemos utilizar com `apt-get upgrade`:

- `-f` ou `--fix-broken`: tenta corrigir erros gerados na atualização de pacotes.
- `-s` ou `--simulate`: apenas simula o processo de atualização.
- `-y` ou `-yes`: assume a resposta `Yes` para todos os questionamentos.

Também, é possível atualizar de uma só vez todos os pacotes que possuam alguma versão mais recente nos repositórios especificados como origem. Para isso, utilizamos o comando `apt-get upgrade` sem a especificação de um pacote. Existe também a opção de utilizar, analogamente, o comando `aptitude upgrade` para atualização de todos os pacotes, porém, nas versões mais recentes do programa, é recomendável que se use o seguinte comando:

```
1 | aptitude safe-upgrade
```

Como foi visto anteriormente, sem a utilização constante dos comandos `apt-get update` ou `aptitude update`, os repositórios não são atualizados, de forma que a atualização de um programa não poderá ser feita com os comandos que acabamos de ver.

Excluir pacotes

O `apt` também pode ser utilizado para remover pacotes instalados e suas respectivas dependências, caso existam, utilizando a ação `remove`. A remoção também é automatizada. A ação `remove` também aceita certas opções para ajustar a execução de desinstalação, como podemos perceber na sintaxe adiante:


```
1 | apt-get [opcoes] remove <pacote1> <pacote2> ...
```

Também, é possível remover vários pacotes ao mesmo tempo com o comando `apt-get` e a ação `remove`.

Veja as opções que podem ser utilizadas com o `apt-get` e a ação `remove`:

- `--purge`: remove os pacotes de forma completa, removendo, também, seus respectivos arquivos de configuração.
- `-s` ou `--simulate`: apenas simula o processo de remoção do pacote, com suas respectivas dependências.
- `-y` ou `-yes`: assume a resposta `Yes` para todos os questionamentos.

O exemplo seguinte demonstra o uso de `apt-get` com a opção `-y` e a ação `remove`:

```
1 | apt-get -y remove lynx
2 | Lendo listas de pacotes... Pronto
3 | Construindo árvore de dependências
4 | Lendo informação de estado... Pronto
5 | Os pacotes a seguir serão REMOVIDOS:
6 |   lynx
7 | 0 pacotes atualizados, 0 pacotes novos instalados, 1 a serem removidos e 51 não
   |   atualizados.
8 | Depois desta operação, 43,0 kB de espaço em disco serão liberados.
9 | (Lendo banco de dados ... 292835 ficheiros e directórios actualmente instalados
   |   .)
10 | Removing lynx (2.8.8pre4-1) ...
```

Também podemos remover um pacote por meio do `aptitude`, utilizando a seguinte sintaxe:

```
1 | aptitude remove <pacote>
```

O exemplo a seguir demonstra o uso do comando `aptitude remove`:

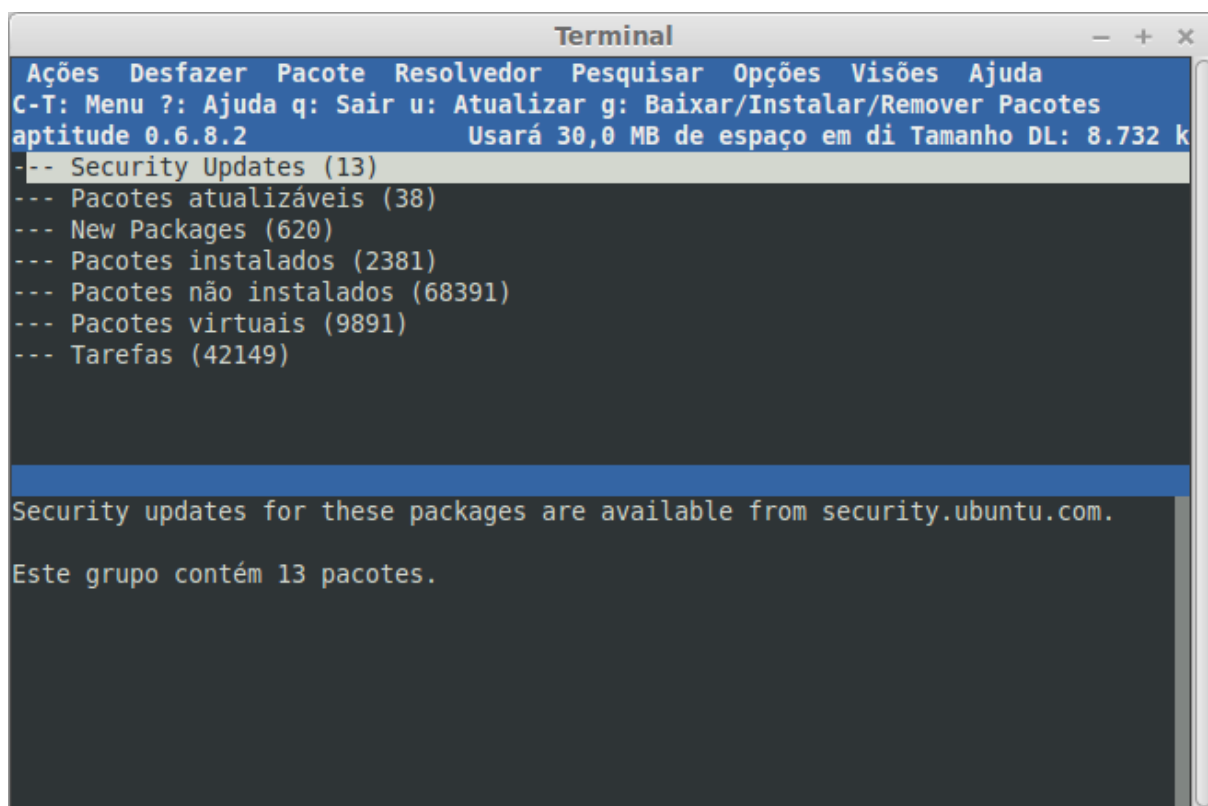
```
1 | aptitude remove lynx
2 | Os pacotes a seguir serão REMOVIDOS:
3 |   lynx
4 | 0 pacotes atualizados, 0 novos instalados, 1 a serem removidos e 51 não
   |   atualizados.
5 | É preciso obter 0 B de arquivos. Depois do desempacotamento, 43,0 kB serão
   |   liberados.
6 | (Lendo banco de dados ... 292835 ficheiros e directórios actualmente instalados
```

```
7 | .)
7 | Removing lynx (2.8.8pre4-1) ...
```

O comando `apt-get -f install` também é muito utilizado para correção de problemas de pacotes quebrados.

O gerenciador de pacotes `aptitude` possui um modo visual, veja a tela na figura abaixo:

```
1 | aptitude
```



Os pacotes baixados pelo `aptitude` ou pelo `apt-get` ficam armazenados no diretório: `/var/cache/apt/archives`. Com o tempo este diretório pode ocupar muito espaço em disco. Para esvaziá-lo basta executar o seguinte comando:

```
1 | aptitude clean
```

Ou:

```
1 | apt-get clean
```

Agora, liste o conteúdo do diretório:

```
1 | ls /var/cache/apt/archives
```

5

yum

Bastante parecido com os gerenciadores apt-get e aptitude, o yum é usado nas distribuições Red Hat. Traz como vantagem bem conveniente a possibilidade de instalação de um pacote a partir da Internet e, como os outros gerenciadores, promove a resolução automática das dependências desse programa. Seu arquivo de configuração é o `/var/cache/yum`.

Os repositórios verificados pelo yum possuem a extensão `.repo` e devem constar na lista de diretórios encontradas na opção `reposdir` do arquivo de configuração.

Veja os principais comandos utilizados com o yum:

- `search [pacote]`: faz uma busca por um determinado pacote.
- `install [pacote]`: instala determinado pacote.
- `remove [pacote]`: exclui determinado pacote.
- `provides [recurso]`: exibe qual pacote fornece o recurso ou arquivo determinado, independentemente desse pacote estar instalado ou não. É uma alternativa para o comando `yum whatprovides [recurso]`.
- `update`: se usado com argumento, atualiza o pacote determinado. Sem argumentos, atualiza todos os pacotes disponíveis. Com o subcomando `--obsoletes`, faz ainda a verificação dos pacotes obsoletos durante a atualização.
- `upgrade`: executa a função do yum `update`, mas também traz a opção de atualizar a versão da distribuição.

O comando `yumdownloader` simplesmente baixa um pacote sem instalá-lo. Com a opção `--source`, ele ainda copia o código-fonte do pacote no lugar do programa compilado.

Configurar repositório no CentOS (/etc/yum.repos.d/)

```

1 [base]
2 name=CentOS-$releasever - Base
3 mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo
  =os
4 gpgcheck=1
5 gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-6
6 [updates] / [extras] / [centosplus] / [contrib]
7 mirrorlist
8 [centosplus] / [contrib]
9 enabled

```

Os repositórios ficam em: /etc/yum.repos.d, vamos adicionar o repositório do dag:

```

1 vim /etc/yum.repos.d/dag.repo
2 [dag]
3 name=Dag RPM Repository for Red Hat Enterprise Linux
4 baseurl=http://apt.sw.be/redhat/el$releasever/en/$basearch/dag
5 gpgcheck=1
6 gpgkey=http://dag.wieers.com/rpm/packages/RPM-GPG-KEY.dag.txt
7 enabled=1

```

Procurando um programa:

```
1 yum search <pacote>
```

Para buscar pelo software vim, digite:

```
1 yum list vim
```

ou

```

1 yum search vim
2 Plugins carregados: fastestmirror
3 Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
4 Determining fastest mirrors
5   base: mirror.nbtelecom.com.br
6   extras: mirror.nbtelecom.com.br
7   updates: mirror.globo.com
8 ===== N/S matched: vim =====

```

```
9 | golang-vim.noarch : Vim plugins for Go
10 | vim-X11.x86_64 : The VIM version of the vi editor for the X Window System
11 | vim-common.x86_64 : The common files needed by any version of the VIM editor
12 | vim-enhanced.x86_64 : A version of the VIM editor which includes recent
13 |                       : enhancements
14 | vim-filesystem.x86_64 : VIM filesystem layout
15 | vim-minimal.x86_64 : A minimal version of the VIM editor
```

Nome e sumário correspondem, use search all para tudo.

A diferença entre as opções list e search é que a primeira opção list irá trazer os resultados que contenham o argumento passado na busca no nome do programa, enquanto que a opção search trará como resultado tanto os comandos, quanto os resumos dos comandos que contenham o argumento passado na busca.

Obtendo informações sobre o pacote:

Usando o yum para mostrar informações de pacotes:

```
1 | yum info <pacote>
```

Para obter informações sobre o pacote do samba:

```
1 | yum info samba
2 | Plugins carregados: fastestmirror
3 | Repodata is over 2 weeks old. Install yum-cron? Or run: yum makecache fast
4 | Loading mirror speeds from cached hostfile
5 |   base: mirror.nbtelecom.com.br
6 |   extras: mirror.nbtelecom.com.br
7 |   updates: mirror.globo.com
8 | Pacotes disponíveis
9 | Nome           : samba
10 | Arquitetura    : x86_64
11 | Versão         : 4.1.1
12 | Lançamento     : 37.el7_0
13 | Tamanho        : 536 k
14 | Repo           : updates/7/x86_64
15 | Sumário        : Server and Client software to interoperate with Windows machines
16 | URL            : http://www.samba.org/
17 | Licença        : GPLv3+ and LGPLv3+
18 | Descrição      : Samba is the standard Windows interoperability suite of programs for
19 |                 : Linux and Unix.
```

Instalação de pacotes YUM

Para instalar um pacote diretamente do repositório:

```
1 | yum install <pacote>
```

Para instalar o samba:

```
1 | yum install samba
```

Remover pacotes YUM

Para remover um pacote do sistema:

```
1 | yum remove <pacote>
```

ou:

```
1 | yum erase <pacote>
```

Remova o samba:

```
1 | yum remove samba
```

ou

```
1 | yum erase samba
```

Atualizar pacotes YUM

Para atualizar os pacotes instalados, digite:

```
1 | # yum update
```

6

dnf

O dnf, a ferramenta de gerenciamento de pacotes usada no Fedora, é um fork do yum. No CentOS 8 por exemplo, foi adotado como opção primária, tendo o yum como opção secundária e isso tem uma explicação: o dnf utiliza a versão 3 do Python que é mais recente, já o yum utiliza a versão 2. Podemos observar que muitos dos comandos e parâmetros são semelhantes.

Comando	Descrição
search	Pesquisa nomes e descrições de pacotes de repositórios para palavras-chave específicas
info	Exibe informações sobre o pacote especificado
install	Instala o pacote especificado
remove	Remove um pacote do sistema
upgrade	Atualiza o(s) pacote(s) especificado(s), mas remove os pacotes obsoletos
provides	Mostra a qual pacote um arquivo pertence
list -- installed	Mostra informações sobre os pacotes instalados
repoquery -l	Listar o conteúdo de um pacote
upgrade	Atualiza o(s) pacote(s) especificado(s), mas remove os pacotes obsoletos

7

ZYpp

A distribuição openSUSE Linux usa o sistema de gerenciamento de pacotes RPM e distribui software em arquivos `.rpm`, mas não usa a ferramenta `yum` ou `dnf`. Em vez disso, o openSUSE criou sua própria ferramenta de gerenciamento de pacotes chamada ZYpp (também chamada `libzypp`). Seu comando, o `zypper` permite você possa consultar, instalar e remover pacotes de software em seu sistema diretamente de um repositório SUSE. A tabela abaixo lista os comandos do utilitário `zypper` mais comumente usados.

Comando	Descrição
<code>help</code>	Exibe informações gerais de ajuda geral ou ajuda sobre um comando especificado
<code>install</code>	Instala o pacote especificado
<code>info</code>	Exibe informações sobre o pacote especificado
<code>list-updates</code>	Exibe todas as atualizações de pacote disponíveis para pacotes instalados do repositório
<code>lr</code>	Exibe informações do repositório
<code>packages</code>	Lista todos os pacotes disponíveis ou lista os pacotes disponíveis de um repositório especificado
<code>what-provides</code>	Mostra a qual pacote um arquivo pertence
<code>refresh</code>	Atualiza as informações de um repositório
<code>remove</code>	Remove um pacote do sistema
<code>search</code>	Pesquisa o(s) pacote(s) especificado(s)

Comando	Descrição
update	Atualiza o(s) pacote(s) especificado(s) ou, se nenhum pacote for especificado, atualiza todos os pacotes atualmente instalados para as versões mais recentes no repositório
verify	Verifica se os pacotes instalados têm suas dependências necessárias satisfeitas

Instalação do editor de textos emacs em uma distribuição openSUSE:

```
1 | # zypper install emacs
```

8

Tópicos para revisão do capítulo

Atente-se para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes deste capítulo.

- Pacotes no Linux são compostos por diversos arquivos, incluindo bibliotecas, traduções, documentação etc.
- O `apt-get` é uma ferramenta poderosa desenvolvida especialmente para o gerenciamento de pacotes no Linux e adotada pelas mais diversas distribuições: Debian, Ubuntu, Linux Mint etc. É capaz de gerenciar pacotes .`deb` pode instalar, remover e atualizar pacotes, além de realizar um upgrade da própria distribuição Linux.
- Um **repositório** de software é um local de armazenamento de onde pacotes de software podem ser recuperados e instalados, exemplo de uma entrada do arquivo `sources.list`:
`deb http://deb.debian.org/debian buster main`
- Em sistemas baseados no Debian, repositórios poderão ser indicados por meio do arquivo `/etc/apt/sources.list`. Arquivos adicionais podem ir para o diretório `/etc/apt/sources.list.d/`.
- Em sistemas baseados em Red Hat, repositórios poderão ser indicados por meio de arquivos com a extensão `.repo` existentes no diretório `/etc/yum.repos.d/`.

9

Empacotadores e Compactadores de Arquivos

Sobre o material

Existem basicamente dois tipos de materiais de estudo: os de referência e os tutoriais. O material de referência é orientado para a consulta por quem já conhece um assunto, e precisa apenas lembrar, ou necessita apenas de detalhes mais específicos sobre um tópico, como a sintaxe de um comando.

Já o material tutorial visa ensinar um conceito ou assunto. Ele tenta ensinar a quem não sabe como fazer algo, como usar os recursos e ferramentas disponíveis para se atingir a um resultado.

Ambos são muito importantes e úteis tanto para o estudante quanto para o profissional. Ambos servem para consulta futura, pois enquanto que o material de referência será acessado com mais frequência depois de acabada a etapa inicial de aprendizado, existem inúmeras maneiras diferentes de se apresentar um assunto, refletindo as variações existentes no mundo real. Então, mesmo quem “já sabe” irá se beneficiar em estudar um novo tutorial de vez em quando. Fora o fato que podemos “esquecer” coisas se ficarmos algum tempo sem fazer, então podemos precisar voltar aos tutoriais para lembrar.

As apostilas deste curso são tutoriais. Não iremos apresentar todas as variações de sintaxe de um comando, nem todas as diretivas de um arquivo de configuração. Isto é informação de referência, que é melhor atendida por recursos da Internet como Google, Wikipedia, e principalmente pela documentação oficial.

Acreditamos que o aluno não tem necessidade de “mais um” material de referência para Linux. Já existem materiais muito bons, disponíveis gratuitamente.

Mas acreditamos que podemos fazer diferença com uma forma diferente de fazer tutoriais. Tentaremos seguir uma abordagem de “aprender fazendo”, privilegiando exemplos de exercícios completos, que o aluno pode executar imediatamente.

Esperamos que o aluno aprenda pela leitura e análise destes exemplos. Ainda assim, é importante complementar o material nestas apostilas com as referências apresentadas no nosso ambiente virtual de ensino.

Também não temos a pretensão de escrever o “melhor tutorial do mundo”. Cada pessoa tem sua própria forma de ver o mundo e aprender sobre os assuntos. Por isso indicaremos, no material complementar, alguns outros tutoriais que serão úteis durante ou depois do curso para solidificar o seu entendimento do Linux.

Introdução

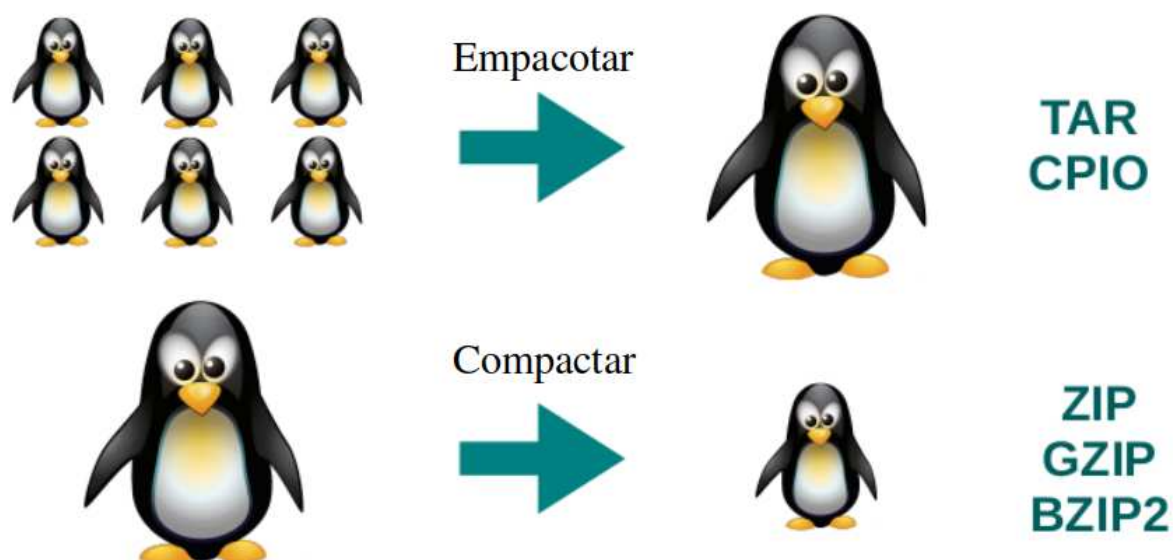
A compressão é usada para reduzir a quantidade de espaço que um conjunto específico de dados consome. Em geral, recorremos a ela para reduzir a quantidade de espaço necessária para armazenar um arquivo. Outro uso comum é reduzir a quantidade de dados enviados através de uma conexão de rede.

Usamos ferramentas de empacotamento para agrupar arquivos e diretórios em um único arquivo. Alguns usos comuns são backups, empacotamento de código-fonte de software e retenção de dados.

Em sistemas operacionais da Microsoft, as tarefas de compressão e empacotamento são geralmente realizadas por um programa **zip**, que executa as operações em conjunto, de maneira que os usuários normalmente sequer sabem a diferença entre elas.

Já em sistemas Unix-like, ambos são efetuados separadamente, e, considerando sua importância, é essencial que se entendam seus procedimentos, para otimizar a distribuição de software, utilização de espaço em disco, backup de sistema, entre outros.

O empacotamento é o ato de aglutinar arquivos e diretórios em um único arquivo. A compressão toma um conjunto de dados e codifica-o de uma maneira que permita seu armazenamento num espaço menor. Na figura abaixo, temos a ilustração da diferença entre Empacotar e Compactar:



O presente capítulo aborda a utilização dos empacotadores **tar** e **cpio**, dos compactadores **gzip** e **bzip2**, dos descompactadores **gunzip** e **bunzip2**, além de expor aspectos relacionados ao backup de dados.

Tar

O **tar** é usado como empacotador de arquivos, preservando as informações do sistema de arquivos aplicados a eles, ou seja, os metadados. É usado em conjunto com compactadores como **gzip** e **bzip2**.

A sintaxe do **tar** é a seguinte:

```
1 | tar [opcoes] <dispositivo/arquivo> <arquivo1> [arquivo2] ...
```

Assim como existe a ação de empacotar arquivos, também é possível executar o processo inverso, que seria o desempacotamento. Para isso, utilizamos **tar** com a opção **-x**.

Os arquivos empacotados pelo **tar** têm a extensão **.tar**, porém o destino de um arquivo empacotado pode ser diretamente um dispositivo de fita que foi originalmente iniciado com a utilização do comando **tar**.

Veja as principais opções que podemos utilizar com o empacotador **tar**:

- **-c** ou **--create**: Cria o arquivo empacotado conforme o destino especificado, podendo ser um arquivo no sistema local ou diretamente em um dispositivo de fita.

- **-f** ou **-file**: Indica se o arquivo empacotado será um arquivo no próprio sistema, com uma extensão `.tar` (recomendada para identificação), ou se o mesmo será um dispositivo de fita. É importante estar atento à utilização dessa opção, pois é obrigatório informar, logo após a mesma, a forma de arquivamento do pacote.
- **-j** ou **-bzip2**: Indica que o pacote será compactado no momento do empacotamento, utilizando o utilitário `bzip2`.
- **-J** ou **-xz**: Indica que o pacote será compactado no momento do empacotamento, utilizando o utilitário `xz`.
- **-v** ou **-verbose**: Exibe, na tela, o processo de empacotamento ou desempacotamento.
- **-t** ou **-list**: Utilizada para visualizar o conteúdo de um pacote `tar`.
- **-z** ou **-gzip**: Indica que o pacote será compactado no momento do empacotamento, utilizando o utilitário `gzip`.
- **-x** ou **-extract**: Desempacota ou extrai o conteúdo de um pacote `tar`.

No exemplo a seguir, o comando **tar** é utilizado para aglutinar (e não compactar, como veremos mais adiante) a pasta `/etc` dentro da pasta `/root/testetar`. O nome do arquivo gerado foi **teste_tar.tar**, com o tamanho de 20MB.

```

1 root@mycomputer:~# mkdir testetar
2 root@mycomputer:~# cd testetar
3 root@mycomputer:~/testetar# tar -cf ./teste_tar.tar /etc
4 tar: Removendo '/' inicial dos nomes dos membros
5 root@mycomputer:~/testetar# ls
6 teste_tar.tar
7 root@mycomputer:~/testetar# ls -l
8 total 20352
9 -rw-r--r-- 1 root root 20838400 Dez 16 17:52 teste_tar.tar

```

tar: Removendo '/' inicial dos nomes dos membros, inicialmente essa mensagem de saída do comando `tar` pode parecer um erro ou até algum tipo de remoção indesejada, mas é apenas um recurso de segurança. Isso evita que o diretório original seja substituído acidentalmente pelos arquivos que estão no backup. Posteriormente ao extrair o conteúdo desse backup ele irá criar um diretório `etc` no diretório corrente, do contrário seria criado um diretório `/etc`, assim sobrescrevendo o diretório original.

Já no próximo exemplo, utilizamos a opção **-x** para descompactar o arquivo. Neste caso, como não definimos o local de destino com a opção **-c**, o arquivo foi descompactado na pasta em que estamos atualmente localizados.

```

1 root@mycomputer:~/testetar# tar -xf teste_tar.tar
2 root@mycomputer:~/testetar# ls -l
3 total 20364

```

```

4 | drwxr-xr-x 151 root root    12288 Dez 16 10:26 etc
5 | -rw-r--r--   1 root root 20838400 Dez 16 17:52 teste_tar.tar
6 |
7 | root@mycomputer:~/testetar# ls etc/
8 | acpi                      fstab.d                      lvm
9 | adduser.conf             fuse.conf                    magic
10 | apg.conf                 ghostscript                  manpath.config
11 | apm                      gimp                        mdm
12 | apparmor                 gnome                       mime.types
13 | ...
14 | ...
15 | ...

```

CPIO

Como o **tar**, o comando **cpio** também exerce as funções de empacotador ou desempacotador, no caso do uso da opção **-i**, além de poder ser usado simplesmente para copiar arquivos. Embora seja capaz de ler e escrever no formato **.tar**, o **cpio** difere do **tar** por trabalhar apenas com redirecionamentos de entrada e saída padrão.

No exemplo a seguir, compactamos a pasta **/etc** com o **cpio**. Além deste último (executado com a opção **-ov**), utilizamos também o comando **ls**. O arquivo compactado foi redirecionado para a pasta **/root/testecpio**.

```

1 | root@mycomputer:~# mkdir testecpio
2 | root@mycomputer:~# cd /etc
3 | root@mycomputer:/etc# ls | cpio -ov > /root/testecpio/pasta_etc.cpio
4 | acpi
5 | adduser.conf
6 | adjtime
7 | alternatives
8 | apparmor
9 | apparmor.d
10 | apport
11 | python3.4
12 | rarfiles.lst
13 | rc0.d
14 | rc1.d
15 | rc2.d
16 | rc3.d
17 | rc4.d
18 | rc5.d
19 | rc6.d

```



```
20 | rc.local
21 | skel
22 | .
23 | .
24 | .
25 | zsh_command_not_found
```

Ao verificarmos a pasta de destino utilizando o comando **ls -l**, podemos notar que o arquivo foi criado com o tamanho de 428K:

```
1 | root@mycomputer:/etc# cd ~/testecpio
2 | root@mycomputer:~/testecpio# ls -hl
3 | total 428K
4 | -rw-r--r-- 1 root root 425K Dez 16 18:01 pasta_etc.cpio
```

Já para descompactar o arquivo, devemos estar na pasta de destino, ou seja, onde ocorrerá a descompactação. Então, executamos o **cpio** com as opções **-iv** e acionamos o arquivo compactado.

```
1 | root@mycomputer:~/testecpio# cpio -iv < pasta_etc.cpio
2 | acpi
3 | adduser.conf
4 | adjtime
5 | alternatives
6 | .
7 | .
8 | .
9 | apm
10 | apparmor
```

Após a descompactação, podemos visualizar o os arquivos na pasta:

```
1 | root@mycomputer:~/testecpio# ls -l
2 | acpi
3 | adduser.conf
4 | adjtime
5 | alternatives
6 | .
7 | .
8 | .
9 | apparmor
```

GZIP

O comando **gzip** é utilizado para compactar e também descompactar arquivos. Arquivos compactados com o utilitário **gzip** são identificados pela extensão **.gz**. Sua sintaxe é a seguinte:

```
1 | gzip [opcoes] <arquivo>
```

Um detalhe muito importante a ser considerado é que um diretório não pode ser compactado de forma direta pelo **gzip**. Ele tem que, obrigatoriamente, se tornar um arquivo empacotado, para depois ser compactado. Porém, é possível compactar todos os arquivos de um determinado diretório através da função recursiva. A compressão pode ser programada para seguir automaticamente o processo de empacotamento realizado pelo tar, com o uso da opção **-z**.

A seguir, veremos algumas opções que podem ser utilizadas com o **gzip**:

- **-c** ou **--stdout**: Não compacta o arquivo original, porém o resultado da compactação será enviado para a saída padrão (tela), e, geralmente, redirecionado para um novo arquivo.
- **-d** ou **--decompress**: Utilizada para descompactar arquivos no formato **.gz**.
- **-f** ou **--force**: Força a compactação ou descompactação de um arquivo, por exemplo, se existe um arquivo de mesmo nome já compactado ou descompactado.
- **-l** ou **--list**: Exibe informações detalhadas de um arquivo compactado, como o tamanho do arquivo compactado e o tamanho do arquivo pós-compactação.
- **-q** ou **--quiet**: Não exibe mensagens de alerta.
- **-r** ou **--recursive**: Compacta, recursivamente, todos os arquivos presentes em um diretório e também arquivos dentro de supostos subdiretórios.
- **-t** ou **--test**: Utilizada para testar a integridade do arquivo compactado.
- **-v** ou **--verbose**: Exibe o processo de compactação ou descompactação.
- **-** ou **--fast** / **--best**: Utilizada para gerenciar o grau de compactação, sendo um valor de 1 a 9. Quanto menor o valor, menor a compactação, e, consequentemente, menor será o tempo de execução do comando. Quanto maior o valor, maior a compactação, e, consequentemente, também será maior o tempo de execução do comando. O comando GNU **--best** equivale à opção Unix **-9**, e **--fast** equivale à opção Unix **-1**.

Vejamos:

```
1 | root@mycomputer:~# mkdir pasta
2 | root@mycomputer:~# cd pasta
3 | root@mycomputer:~/pasta# ls -lh
4 | total 13M
5 | -rw-r--r-- 1 root root 9,2M Dez 16 18:29 teste
6 | -rw-r--r-- 1 root root 3,0M Dez 16 18:31 teste2
```

```

7 |
8 | root@mycomputer:~/pasta# gzip teste
9 | root@mycomputer:~/pasta# ls -lh
10 | total 3,1M
11 | -rw-r--r-- 1 root root 3,0M Dez 16 18:31 teste2
12 | -rw-r--r-- 1 root root 51K Dez 16 18:29 teste.gz
13 |
14 | root@mycomputer:~/pasta# gzip -c teste2 > testinho.gz
15 | root@mycomputer:~/pasta# ls -lh
16 | total 3,1M
17 | -rw-r--r-- 1 root root 3,0M Dez 16 18:31 teste2
18 | -rw-r--r-- 1 root root 51K Dez 16 18:29 teste.gz
19 | -rw-r--r-- 1 root root 18K Dez 16 18:32 testinho.gz
20 |
21 | root@mycomputer:~/pasta# gzip -d teste.gz
22 | root@mycomputer:~/pasta# ls -lh
23 | total 13M
24 | -rw-r--r-- 1 root root 9,2M Dez 16 18:29 teste
25 | -rw-r--r-- 1 root root 3,0M Dez 16 18:31 teste2
26 | -rw-r--r-- 1 root root 18K Dez 16 18:32 testinho.gz

```

Na primeira execução de **gzip** da imagem anterior, temos uma simples compactação do arquivo teste; o arquivo original foi compactado. Na segunda execução, o arquivo teste2 foi compactado com a opção **-c**, utilizando o recurso do redirecionador **>**, sendo o resultado da compactação redirecionado para um novo arquivo de nome testinho.gz, mantendo o arquivo original intacto. Na última execução de **gzip**, o arquivo teste.gz foi descompactado por meio da opção **-d**.

Agora, vejamos este outro exemplo:

```

1 | root@mycomputer:~# mkdir teste_tar_gzip
2 | root@mycomputer:~# cd teste_tar_gzip
3 | root@mycomputer:~# teste_tar_gzip
4 | root@mycomputer~/teste_tar_gzip# tar -czf ./pasta_etc.tar.gz /etc/
5 | tar: Removendo '/' inicial dos nomes dos membros
6 | root@mycomputer~/teste_tar_gzip# ls -lh
7 | total 2,0M
8 | -rw-r--r-- 1 root root 2,0M Dez 16 18:36 pasta_etc.tar.gz
9 | root@mycomputer~/teste_tar_gzip#

```

No exemplo que acabamos de ver, diferentemente da aglutinação, utilizamos o **tar** com **gzip** (opção **-z**). O arquivo, então, é compactado. O tamanho do arquivo compactado é de 2 MB, bem menor do que os 20 MB do arquivo obtido com o exemplo de utilização do comando **tar**.

A seguir, temos um exemplo de descompactação de um arquivo com uso do **gzip -xz**:

```
1 root@mycomputer~/teste_tar_gzip# tar -xzf pasta_etc.tar.gz
2 root@mycomputer~/teste_tar_gzip# ls -l
3 total 1992
4 drwxr-xr-x 151 root root 12288 Dez 16 10:26 etc
5 -rw-r--r-- 1 root root 2023554 Dez 16 18:36 pasta_etc.tar.gz
6 root@mycomputer~/teste_tar_gzip# ls ./etc/
7 acpi                                fstab.d                                lvm
8 adduser.conf                       fuse.conf                              magic
9 adjtime                            gai.conf                              magic.mime
10 .
11 .
12 .
```

Família de comandos do gzip

O comando **gzip** tem uma família de comandos utilizados com a função de simplesmente visualizar arquivos já compactados, porém utilizando recursos adicionais de alguns comandos que vimos anteriormente. Essa família de comandos, descrita na tabela adiante, pode ser útil quando houver a necessidade de executar determinada tarefa em um arquivo compactado com o formato **gzip**.

- **zcat**: Visualiza, com o comando `cat`, arquivos compactados no formato `.gz`, sem necessariamente descompactar os arquivos.
- **zgrep**: Utiliza o recurso do comando `grep` para localizar palavras ou expressões em um arquivo compactado no formato `.gz`, sem descompactar o mesmo.
- **zless**: Utiliza o recurso do comando `less` para visualizar um arquivo compactado no formato `.gz`.
- **zmore**: Utiliza o recurso do comando `more` para visualizar um arquivo compactado no formato `.gz`.

BZIP2

O comando **bzip2** é utilizado para compactar e também descompactar arquivos. Arquivos compactados com o utilitário **bzip2** são identificados pela extensão **.bz2**. Sua sintaxe é a seguinte:

```
1 | bzip2 [opcoes] <arquivo>
```

A seguir as principais opções utilizadas com o **bzip2**:

- **-c** ou **-stdout**: Não compacta o arquivo original, porém o resultado da compactação será enviado para a saída padrão (tela), e, geralmente, redirecionado para um novo arquivo.
- **-d** ou **-decompress**: Utilizada para descompactar arquivos que foram compactados pelo comando **gzip**.
- **-f** ou **-force**: Força a compactação ou descompactação de um arquivo, por exemplo, se já existe um arquivo de mesmo nome, já compactado ou descompactado.
- **-q** ou **-quiet**: Não exibe mensagens de alerta.
- **-t** ou **-test**: Utilizada para testar a integridade do arquivo compactado.
- **-v** ou **-verbose**: Exibe o processo de compactação ou descompactação.
- **-** ou **-fast/-best**: Utilizada para gerenciar o grau de compactação, sendo um valor de 1 a 9. Quanto menor o valor, menor a compactação e, consequentemente, menor será o tempo de execução do comando. Quanto maior o valor, maior a compactação e, consequentemente, também será maior o tempo de execução do comando. O comando GNU **-best** equivale à opção Unix **-9**, e **-fast** equivale à opção Unix **-1**.

Um detalhe muito importante é que um diretório não pode ser compactado de forma direta pelo comando **gzip**, tendo, obrigatoriamente, que se tornar primeiro um arquivo empacotado, para depois ser compactado.

No exemplo a seguir, aproveitamos um arquivo aglutinado com **tar** e o copiamos para a pasta de teste do **bzip**. Podemos perceber que o arquivo possui 20 MB. Após utilizarmos o comando **bzip2** para compactá-lo, seu tamanho reduziu para 1,7 MB.

```

1 root@mycomputer:~# mkdir teste_bzip
2 root@mycomputer:~# cd teste_bzip
3 root@mycomputer:~/teste_bzip# tar cvf teste_tar.tar /etc/
4
5 root@mycomputer:~/teste_bzip# ls -lh
6 total 20M
7 -rw-r--r-- 1 root root 20M Dez 16 18:47 teste_tar.tar
8 root@mycomputer:~/teste_bzip#
9 root@mycomputer:~/teste_bzip# bzip2 teste_tar.tar
10 root@mycomputer:~/teste_bzip# ls -lh
11 total 1,7M
12 -rw-r--r-- 1 root root 1,7M Dez 16 18:47 teste_tar.tar.bz2
13 root@mycomputer:~/teste_bzip#
```

Para descompactarmos esse mesmo arquivo, utilizamos **bzip2** com a opção **-d**, como mostrado a seguir:

```

1 root@mycomputer:~/teste_bzip# ls
2 teste_tar.tar.bz2
3 root@mycomputer:~/teste_bzip# bzip2 -d teste_tar.tar.bz2
```

```
4 root@mycomputer:~/teste_bzip# ls -lh
5 total 20M
6 -rw-r--r-- 1 root root 20M Dez 16 18:47 teste_tar.tar
7 root@mycomputer:~/teste_bzip#
```

Opcionalmente, podemos utilizar o parâmetro **time** para obtermos o tempo gasto na compactação. No exemplo a seguir, adicionamos **time** ao **bzip2**. Assim, obtivemos o tempo gasto na compactação do arquivo especificado:

```
1 root@mycomputer:~/teste_bzip# ls
2 teste_tar.tar
3 root@mycomputer:~/teste_bzip# time bzip2 teste_tar.tar
4
5 real    0m1.750s
6 user    0m1.702s
7 sys     0m0.041s
8 root@mycomputer:~/teste_bzip# ls
9 teste_tar.tar.bz2
10 root@mycomputer:~/teste_bzip#
```

Família de comandos do bzip2

O comando **bzip2** também tem uma família de comandos utilizados com a função de apenas visualizar arquivos compactados, porém utilizando recursos de alguns comandos vistos anteriormente. Essa família de comandos, descrita na tabela adiante, pode ser útil quando houver a necessidade de executar determinada tarefa em um arquivo compactado com o formato **bzip2**.

- **bzcat**: Visualiza, com o comando **cat**, arquivos compactados no formato **.bz2**, sem necessariamente descompactar o arquivo.
- **bzgrep**: Utiliza o recurso do comando **grep** para localizar palavras ou expressões em um arquivo compactado no formato **.bz2**, sem descompactar o mesmo.
- **zless**: Utiliza o recurso do comando **less** para visualizar um arquivo compactado no formato **.bz2**.
- **zmore**: Utiliza o recurso do comando **more** para visualizar um arquivo compactado no formato **.bz2**.

xz

O comando **xz** também é utilizado para compactar e também descompactar arquivos. Arquivos compactados com o utilitário **xz** são identificados pela extensão **.xz**. Sua sintaxe é a seguinte:

```
1 | xz [opcoes] <arquivo>
```

A seguir, veremos algumas opções que podem ser utilizadas com o **xz**:

- **-c** ou **--stdout**: Não compacta o arquivo original, porém o resultado da compactação será enviado para a saída padrão (tela), e, geralmente, redirecionado para um novo arquivo.
- **-d** ou **--decompress**: Utilizada para descompactar arquivos no formato .xz.
- **-f** ou **--force**: Força a compactação ou descompactação de um arquivo, por exemplo, se existe um arquivo de mesmo nome já compactado ou descompactado.
- **-l** ou **--list**: Exibe informações detalhadas de um arquivo compactado, como o tamanho do arquivo compactado e o tamanho do arquivo pós-compactação.
- **-q** ou **--quiet**: Não exibe mensagens de alerta.
- **-t** ou **--test**: Utilizada para testar a integridade do arquivo compactado.
- **-v** ou **--verbose**: Exibe o processo de compactação ou descompactação.
- **-** ou **--fast** / **--best**: Utilizada para gerenciar o grau de compactação, sendo um valor de 1 a 9. Quanto menor o valor, menor a compactação, e, consequentemente, menor será o tempo de execução do comando. Quanto maior o valor, maior a compactação, e, consequentemente, também será maior o tempo de execução do comando. O comando GNU **--best** equivale à opção Unix **-9**, e **--fast** equivale à opção Unix **-1**.

Vejamos:

```
1 | root@mycomputer:~# mkdir teste_xz
2 | root@mycomputer:~# cd teste_xz
3 | root@mycomputer:~/teste_xz# touch teste teste2
4 | # ls -l
5 | total 0
6 | drwxrwxr-x 2 root root 4096 dez 17 14:49 teste
7 | drwxrwxr-x 2 root root 4096 dez 17 14:49 teste2
8 |
9 | root@mycomputer:~/teste_xz# xz teste
10 | root@mycomputer:~/teste_xz# ls -l
11 | total 4
12 | -rw-rw-r-- 1 root root 0 dez 17 14:49 teste2
13 | -rw-rw-r-- 1 root root 32 dez 17 14:49 teste.xz
14 |
15 | root@mycomputer:~/teste_xz# xz -c teste2 > testinho.xz
16 | root@mycomputer:~/teste_xz# ls -lh
17 | total 8,0K
18 | -rw-rw-r-- 1 root root 0 dez 17 14:49 teste2
19 | -rw-rw-r-- 1 root root 32 dez 17 14:49 teste.xz
20 | -rw-rw-r-- 1 root root 32 dez 17 14:50 testinho.xz
21 |
22 | root@mycomputer:~/teste_xz# xz -d teste.xz
```

```

23 root@mycomputer:~/teste_xz# ls -lh
24 total 4,0K
25 -rw-rw-r-- 1 root root  0 dez 17 14:49 teste
26 -rw-rw-r-- 1 root root  0 dez 17 14:49 teste2
27 -rw-rw-r-- 1 root root 32 dez 17 14:50 testinho.xz

```

Na primeira execução de **xz** da imagem anterior, temos uma simples compactação do arquivo teste, o arquivo original foi compactado. Na segunda execução, o arquivo teste2 foi compactado com a opção **-c**, utilizando o recurso do redirecionador **>**, sendo o resultado da compactação redirecionado para um novo arquivo de nome testinho.gz, mantendo o arquivo original intacto. Na última execução de **xz**, o arquivo teste.gz foi descompactado por meio da opção **-d**.

Agora, vejamos este outro exemplo:

```

1 root@mycomputer:~# cd ~/teste_xz
2 root@mycomputer:~/teste_xz# tar -cJf ./pasta_etc.tar.xz /etc/
3 tar: Removendo '/' inicial dos nomes dos membros
4 root@mycomputer:~/teste_xz# ls -lh
5 total 1,3M
6 -rw-r--r-- 1 root  root  1,2M dez 17 14:54 pasta_etc.tar.xz

```

No exemplo que acabamos de ver, diferentemente da aglutinação, utilizamos o **tar** com **xz** (opcao **-J**). O arquivo, então, é compactado. O tamanho do arquivo compactado é de 1,2M, bem menor do que os 8,6M do arquivo obtido caso o mesmo diretório tivesse sido agrupado apenas com o **tar**.

A seguir, temos um exemplo de descompactação de um arquivo com uso do **xz -xJ**:

```

1 root@mycomputer:~# cd ~/teste_xz
2 root@mycomputer:~/teste_xz# tar -xJf pasta_etc.tar.xz
3 root@mycomputer:~/teste_xz# ls -l
4 total 9956
5 drwxr-xr-x 147 root root 12288 dez 16 10:12 etc
6 -rw-r--r--  1 root  root  8919040 dez 17 14:56 pasta_etc.tar
7 -rw-r--r--  1 root  root  1253668 dez 17 14:54 pasta_etc.tar.xz
8 root@mycomputer:~/teste_xz# ls ./etc/
9 acpi                                hosts                                popularity-contest.conf
10 adduser.conf                       hosts.allow                         ppp
11 alsa                               hosts.deny                         profile
12 alternatives                       hp                                 profile.d
13 anacrontab                         ifplugd                            protocols
14 .
15 .

```

16 | .

GUNZIP

O comando `gunzip` é utilizado apenas para descompactar arquivos, tendo exatamente a mesma função do comando `gzip -d`. Ele é utilizado para descompactar apenas um arquivo ou todos os arquivos presentes em um diretório específico, devido à função recursiva.

A sintaxe do `gunzip` é a seguinte:

```
1 | gunzip [opcoes] <arquivo>
```

As principais opções do `gunzip` estão descritas a seguir:

- **-c** ou **–stdout**: Não descompacta o arquivo original, porém o resultado da descompactação será enviado para a saída padrão (tela), e, geralmente, redirecionado para um novo arquivo.
- **-f** ou **–force**: Força a descompactação de um arquivo, por exemplo, se existe um arquivo de mesmo nome já descompactado.
- **-q** ou **–quiet**: Não exibe mensagens de alerta.
- **-r** ou **–recursive**: Descompacta recursivamente todos os arquivos presentes em um diretório, e também arquivos dentro de supostos subdiretórios.
- **-v** ou **–verbose**: Exibe o processo de descompactação.

No exemplo a seguir, o comando `gunzip` foi utilizado para descompactar um arquivo que foi gerado com o `gzip`:

```
1 | root@mycomputer:~# mkdir teste_gunzip
2 | root@mycomputer:~# cd teste_gunzip
3 | root@mycomputer:~/teste_gunzip# ls -l
4 | total 1980
5 | -rw-r--r-- 1 root root 2023535 Dez 16 19:23 pasta_etc.tar.gz
6 | root@mycomputer:~/teste_gunzip# gunzip pasta_etc.tar.gz
7 | root@mycomputer:~/teste_gunzip# ls -l
8 | total 20352
9 | -rw-r--r-- 1 root root 20838400 Dez 16 19:23 pasta_etc.tar
```

BUNZIP2

O comando `bunzip2` é utilizado apenas para descompactar arquivos, e tem exatamente a mesma função do comando `bzip2 -d`. A sintaxe do `bunzip2` é a seguinte:

```
1 | bunzip2 [opcoes] [arquivo]
```

Veja as principais opções utilizadas com `bunzip2`:

- **-c** ou **–stdout**: Não descompacta o arquivo original, porém o resultado da descompactação será enviado para a saída padrão (tela), e geralmente redirecionado para um novo arquivo.
- **-f** ou **–force**: Força a descompactação de um arquivo, por exemplo, se existe um arquivo de mesmo nome já descompactado.
- **-q** ou **–quiet**: Não exibe mensagens de alerta.
- **-v** ou **–verbose**: Exibe o processo de descompactação.

No exemplo a seguir utilizamos o `bunzip2` para descompactarmos um arquivo criado pelo `bzip2`:

```
1 | root@mycomputer:~/teste_gunzip# ls -l
2 | total 1684
3 | -rw-r--r-- 1 root root 1721676 Dez 16 19:23 pasta_etc.tar.bz2
4 | root@mycomputer:~/teste_gunzip# bunzip2 pasta_etc.tar.bz2
5 | root@mycomputer:~/teste_gunzip# ls -l
6 | total 20352
7 | -rw-r--r-- 1 root root 20838400 Dez 16 19:23 pasta_etc.tar
```

UNXZ

O comando `unxz` é utilizado apenas para descompactar arquivos, tendo exatamente a mesma função do comando `xz -d`.

A sintaxe do `unxz` é a seguinte:

```
1 | unixz [opcoes] <arquivo>
```

As principais opções do `unxz` estão descritas a seguir:

- **-c** ou **–stdout**: Não descompacta o arquivo original, porém o resultado da descompactação será enviado para a saída padrão (tela), e, geralmente, redirecionado para um novo arquivo.

- **-k** ou **-keep**: Mantém o arquivo original e cria uma cópia compactada.
- **-f** ou **-force**: Força a descompactação de um arquivo, por exemplo, se existe um arquivo de mesmo nome já descompactado.
- **-q** ou **-quiet**: Não exibe mensagens de alerta.
- **-v** ou **-verbose**: Exibe o processo de descompactação.

No exemplo a seguir, o comando `unxz` foi utilizado para descompactar um arquivo que foi gerado com o `xz`:

```
1 root@mycomputer:~# cd teste_xz
2 root@mycomputer:~/teste_xz# ls -l
3 total 1212
4 -rw-rw-r-- 1 root root 1239704 dez 17 14:36 pasta_etc.tar.xz
5 root@mycomputer:~/teste_xz# unpz pasta_etc.tar.xz
6 root@mycomputer:~/teste_xz# ls -l
7 total 8636
8 -rw-rw-r-- 1 root root 8847360 dez 17 14:36 pasta_etc.tar
```

Considerações sobre backup de dados

O comando `dd` é eficiente nas operações de backup, especialmente para copiar discos ou partições inteiras e imagens de mídias como CDs e DVDs. O `dd` faz uma cópia sequencial de dados, byte a byte, de qualquer origem para qualquer destino. Sua utilização inadequada, porém, pode causar sérios danos ao sistema. Portanto, é necessário ter cautela ao utilizá-lo.

No exemplo a seguir, o comando `dd` faz uma cópia fiel do HD master primário (`hda`) para o HD slave primário (`hdb`):

```
1 dd if=/dev/hda of=/dev/hdb
```

Já neste outro exemplo, o comando `dd` copia o setor MBR (Master Boot Record) para um arquivo. O MBR localiza-se nos primeiros 512 setores do disco.

```
1 # dd if=/dev/sda of=/root/laboratorio/backup.mbr bs=512 count=1
2 1+0 registros de entrada
3 1+0 registros de saída
4 512 bytes (512 B) copiados, 0,000429038 s, 1,2 MB/s
```

Entre os tipos de backup que podemos realizar no sistema, diferem-se o backup completo, incremental e diferencial.

O backup completo salva todos os dados, independente de já terem sido salvos ou não em backups anteriores, portanto os objetos são armazenados completamente.

Os backups incremental e diferencial podem ser definidos pela sua relação com o último backup completo realizado. O backup diferencial salva as diferenças em relação ao backup completo, de forma que serão salvos, em cada backup diferencial, os dados adicionais ao último backup completo. Assim, caso sejam feitos três backups diferenciais, em momentos 1, 2 e 3, o primeiro salvará os dados do backup completo até o momento 1; o segundo salvará do backup completo até o momento 2; já o terceiro cobrirá os dados do backup completo até o momento 3.

O backup incremental gera um volume menor de dados, pois, usando o mesmo esquema que usamos para o backup diferencial, caso sejam realizados três backups incrementais, em momentos 1, 2 e 3, o primeiro deles salvará os dados do backup completo até o momento 1; o segundo salvará os dados do momento 1 ao momento 2; ao terceiro restarão os dados do momento 2 ao momento 3.

Tópicos para revisão do capítulos

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O empacotamento é o ato de aglutinar arquivos e diretórios em um único arquivo. A compressão toma um conjunto de dados e codifica-o de uma maneira que permita seu armazenamento num espaço menor;
- O comando tar é usado como empacotador de arquivos, preservando as informações do sistema de arquivos aplicados a eles, ou seja, os metadados;
- O comando cpio também exerce as funções de empacotador ou desempacotador, além de poder ser usado simplesmente para copiar arquivos. Embora seja capaz de ler e escrever no formato .tar, o cpio difere do tar por trabalhar apenas com redirecionamentos de entrada e saída padrão;
- O comando gzip é utilizado para compactar e também descompactar arquivos. Arquivos compactados com o utilitário gzip são identificados pela extensão .gz;
- O comando bzip2 é utilizado para compactar e também descompactar arquivos. Arquivos compactados com o utilitário bzip2 são identificados pela extensão .bz2;
- O comando gunzip é utilizado apenas para descompactar arquivos, tendo exatamente a mesma função do comando gzip -d;
- O comando bunzip2 é utilizado apenas para descompactar arquivos, e tem exatamente a mesma função do comando bzip2 -d;
- Os backups que podemos realizar no sistema podem ser completos, incrementais ou diferenciais. O comando dd é eficiente nas operações de backup, especialmente para copiar discos ou partições inteiras e imagens de mídias como CDs e DVDs.

10

Raio-X do seu Hardware e Gerenciamento de Processos

Raio-X do seu Hardware

Introdução

Com exceção das placas de rede, os dispositivos de hardware, em sistemas GNU/Linux, também são representados como arquivos, armazenados de forma organizada no diretório **/dev**, onde podem ser administrados pelo usuário.

Conhecer bem a estrutura e o funcionamento desses arquivos é necessário para um bom gerenciamento do sistema. Neste capítulo, começaremos com um estudo dos arquivos de dispositivos presentes no diretório **/dev** e, sequencialmente, contemplaremos conceitos e procedimentos de montagem de dispositivos, particionamento de discos, aplicação de sistemas de arquivos e gerenciamento desses sistemas.

O subdiretório **/dev**

O subdiretório **/dev** é um sistema de arquivos especial onde estão localizados todos os arquivos referentes aos dispositivos do computador, exceto a interface de rede, manipulada diretamente no nível do kernel. Fora ela, praticamente todos os dispositivos de hardware têm um arquivo correspondente no diretório **/dev**.

O **/dev** pode aparecer nos computadores de duas formas diferentes: **devfs** ou **udev**. Uma

diferença básica entre os dois é que, ao contrário do **devfs**, que é mais antigo e criava um arquivo de dispositivo para cada dispositivo de hardware suportado pelo sistema, independentemente de ele estar em uso ou mesmo fisicamente presente na máquina, o **udev** gera esses arquivos conforme a disponibilidade e presença dos dispositivos no sistema, de forma a otimizar a capacidade de armazenamento do **/dev**.

Arquivos de dispositivo

Em um sistema GNU/Linux, um dispositivo é acessado por meio de seu respectivo arquivo de dispositivo no diretório **/dev**. Esses arquivos, que acabamos de apresentar, são uma interface para o driver atual que acessa o dispositivo de hardware.

Os arquivos de dispositivo são divididos nos seguintes tipos:

- **Arquivos de dispositivos de caractere:** São orientados à comunicação e ao fluxo de informações em modo serial, ou seja, caractere a caractere. Geralmente, são utilizados em comunicações feitas por portas seriais ou paralelas, como, por exemplo, modems. O próprio terminal de comandos utiliza arquivos desse tipo para criar uma interface de comunicação com o kernel;
- **Arquivos de dispositivos de bloco:** São orientados, geralmente, ao armazenamento de informações temporárias em algum tipo de mídia, ou seja, tal comunicação feita em modo de blocos possibilitará o armazenamento e, posteriormente, a leitura dessas informações. Esse tipo de arquivo é utilizado, geralmente, para representar partições de discos ou pendrive, CD-ROM, DVD, etc;
- **Arquivos de dispositivos FIFO:** FIFO é uma sigla para o termo First In, First Out (primeiro a entrar, primeiro a sair), que caracteriza o processamento de dados em esquema de fila e viabiliza a comunicação entre dois processos independentes. É comum também chamar esses arquivos de pipe nomeado;
- **Arquivos de dispositivos socket:** Voltados para a criação de pontos de comunicação.
- Os tipos de arquivos de dispositivo mais manipulados são os de caractere e de bloco.

Vejamos alguns exemplos de arquivos de caractere e de bloco na tabela abaixo:

Tipo	Arquivo	Representação
Caractere	<code>/dev/ttyS1</code>	Primeira porta serial.
	<code>/dev/lp0</code>	Primeira porta paralela.
Bloco	<code>/dev/hda1</code>	Primeira partição de disco IDE.
	<code>/dev/fd0</code>	Primeiro dispositivo de disquete.

Dispositivos de armazenamento

Como o próprio nome já diz, dispositivo de armazenamento é qualquer hardware onde se possam gravar dados, como CDs, pendrives, HDs, etc. Existem tipos diferentes de dispositivos de armazenamento. Em computadores pessoais, os discos mais comuns, até pouco tempo atrás, eram os do tipo IDE. Com o avanço tecnológico, os mais comuns atualmente são os do tipo SATA. Discos SCSI e SAS são mais rápidos e sofisticados, sendo preferidos para servidores e máquinas de alto desempenho.

A nomeação dos diretórios e dispositivos de armazenamento em sistemas GNU/Linux é normalmente algo que confunde o usuário iniciante, já que difere muito de outras plataformas. O nome usado para identificar cada arquivo de dispositivo no diretório `/dev` vai depender do tipo de dispositivo, seja ele um disquete, IDE, SATA, SCSI ou qualquer outro. Dessa forma, é importante chamar atenção para os padrões a seguir:

- A primeira e a segunda unidade de disquete têm os nomes `/dev/fd0` e `/dev/fd1`, respectivamente;
- Dispositivos IDE são nomeados de acordo com o canal IDE utilizado e o modo como foram conectados, seja ele master (mestre) ou slave (escravo). Assim, `/dev/hda` e `/dev/hdb` serão respectivamente os disco master e slave conectados na controladora primária. Na secundária, os discos master e slave serão chamados `/dev/hdc` e `/dev/hdd`, nessa ordem;
- Dispositivos SATA e SCSI recebem os nomes `/dev/sda`, `/dev/sdb` e assim por diante, de acordo com a ordem em que são identificados pela BIOS. Controladoras SCSI podem ser de 8 ou 16 bits, sendo que o primeiro tipo comporta até 7 dispositivos, e o segundo aceita, no máximo, 15 (contando com a própria unidade, que utiliza 1 bit);
- Cartões e pendrives, que são dispositivos SDD, seguem o padrão SATA, SAS e SCSI de nomenclatura, e são numerados a partir da quantidade de discos SATA ou SCSI já existentes, ou seja, caso existam três discos SATA ou SCSI, um pendrive será representado por `/dev/sdd`, por exemplo;
- Drives de CD/DVD seguem o padrão de nomenclatura IDE ou SCSI, dependendo do tipo do dispositivo em questão. O último caractere não numérico do nome do arquivo será a letra do alfabeto seguinte à que foi utilizada para o último dispositivo reconhecido pela BIOS. Porém, é comum as distribuições do Linux também criarem automaticamente um link para o drive de CD/DVD com o nome `/dev/cdrom`.
- Se o usuário preferir e a distribuição do Linux que ele estiver utilizando permitir, os dispositivos IDE podem ser configurados para seguir o padrão SATA e SCSI de nomeação, de maneira que terão os nomes `/dev/sda` e `/dev/sdb` os discos master e slave da primeira controladora e assim sucessivamente.
- Para identificar cada partição num desses dispositivos, um numeral cardinal será adicionado ao seu nome, sendo que as partições primárias serão numeradas de 1 a 4 e as lógicas partirão do algarismo 5. Assim, se houver, por exemplo, duas partições no disco master da controladora primária, seus nomes serão `/dev/hda1` e `/dev/hda2`.

Nomes de unidades sempre terminam com 0. Exemplos: **fd0** e **lp0**. Já partições sempre terminam com 1. Exemplos: **hda1** e **sdb1**.

udev

O **udev** substituiu o gerenciador de dispositivos **devfs** a partir da versão 2.6.12 do kernel. Com o antigo **devfs**, todos os arquivos de dispositivo eram criados de uma vez no diretório **/dev** de maneira a gerar no diretório os dispositivos de todos os itens de hardwares que o sistema suportasse, independentemente da existência desses itens na máquina, ou seja, com o **devfs** acabavam-se criando arquivos de dispositivo desnecessários, super populando o diretório **/dev**.

No caso do **udev**, os arquivos de dispositivo são gerados dinamicamente, conforme os itens de hardware são instalados, o que significa que o que não estiver sendo usado no computador não ocupa espaço no disco. Nos sistemas mais recentes, o **udev** possui capacidade **coldplug** e **hotplug**, responsáveis por identificar e configurar os dispositivos presentes desde o ligamento do computador e aqueles conectados após a inicialização do sistema respectivamente.

O arquivo de configuração do **udev** é o **/etc/udev/udev.conf**, e os arquivos de regra ficam no diretório **/etc/udev/rules.d/**. Os comandos **udevmonitor** ou **udevadm monitor**, dependendo da versão do sistema, permitem monitorar os eventos armazenados no sistema de arquivos lógico chamado **SysFS**, onde são registradas as informações de identificação dos dispositivos.

Sistema de arquivos

Podemos definir um sistema de arquivos como um meio de deixar os dados de nosso sistema mais organizados. Todo sistema operacional gera um tipo de sistema de arquivos. A seguir, abordaremos as informações sobre os sistemas de arquivos e, também, sobre como podemos gerenciá-los por meio de ferramentas específicas. O Linux oferece suporte a vários sistemas de arquivos de outros sistemas operacionais. A seguir, citamos alguns:

- xiafs;
- minix;
- msdos;
- vfat;
- proc;
- smbfs;
- iso9660;
- hpfs;
- ufs.

Os sistemas de arquivos são alocados em partições lógicas; cada partição aceita um único sistema de arquivos.

Estrutura básica

De maneira geral, podemos dividir o sistema de arquivos em quatro partes: bloco de boot, superbloco, tabela de inodes e bloco de dados.

Bloco de boot

O bloco de boot fica localizado no início do sistema de arquivos. Para acessá-lo, podemos utilizar o código mínimo incorporado na ROM da BIOS do computador. O bloco de boot da partição bootável contém o código necessário para inicializar o sistema operacional posteriormente.

Superbloco

Este bloco é responsável por descrever o estado do sistema de arquivos no que diz respeito ao tamanho e à quantidade de arquivos que tal sistema pode armazenar. Além disso, ele descreve as informações sobre quais partes da área de armazenamento já estão sendo utilizadas e quais ainda estão disponíveis, bem como outras informações.

A seguir, listamos as informações contidas no superbloco:

- Tamanho do sistema de arquivos, que faz referência à área de armazenamento do dispositivo ou da partição atual no dispositivo;
- Relação de blocos de armazenamento;
- Quantidade de blocos disponíveis;
- Localização de todos os blocos disponíveis;
- Tamanho da lista de inode: esta lista é inicializada com o propósito de rastrear a maior quantidade possível de arquivos;
- Quantidade de inodes disponíveis no sistema de arquivos;
- Índice do próximo inode disponível na lista de inodes disponíveis.

Tabela de inodes

O inode é definido como uma estrutura de dados que apresenta um sistema de arquivos que adota a semântica de sistemas Unix. Os inodes têm como característica conter dados de arquivos, tais como localização, tipo, proprietário, tempo de acesso, entre outros.

Podemos definir a lista de inodes como uma lista estática. O tamanho da lista não pode ser alterado depois que o sistema de arquivos for criado. O administrador e o tamanho do dispositivo de armazenamento são responsáveis por definir o tamanho desta lista.

A seguir, listamos os tipos de informações incluídas em um inode:

- **ID do proprietário do arquivo:** Refere-se ao UID especificado no arquivo de usuários para identificar o proprietário do arquivo;

- **ID do grupo:** Refere-se ao GID especificado no arquivo de grupos para identificar o grupo do arquivo;
- **Tipo de arquivo:** Responsável por armazenar a especificação do tipo de arquivo, isto é, especifica se o arquivo é um diretório, um link simbólico, um dispositivo de bloco ou caractere ou um pipe;
- **Permissões do arquivo:** Traz informações de permissões de acesso para as classes usuário (owner), grupo (group) e outros (other);
- **Tempo de acesso:** Por meio deste item, podemos obter informações sobre tempo de acesso e modificação dos arquivos, além do número de links, tamanho do arquivo e o local onde os dados estão armazenados no dispositivo.

Bloco de dados

Os blocos de dados estão localizados após os blocos que possuem a lista de inodes e têm a função de armazenar informações administrativas, bem como os dados dos arquivos.

É importante lembrar que um bloco de dados nunca pode pertencer a mais de um sistema de arquivos.

Estrutura básica

De maneira geral, podemos dividir o sistema de arquivos em quatro partes: bloco de boot, superbloco, tabela de inodes e bloco de dados.

Bloco de boot

O bloco de boot fica localizado no início do sistema de arquivos. Para acessá-lo, podemos utilizar o código mínimo incorporado na ROM da BIOS do computador. O bloco de boot da partição bootável contém o código necessário para inicializar o sistema operacional posteriormente.

Superbloco

Este bloco é responsável por descrever o estado do sistema de arquivos no que diz respeito ao tamanho e à quantidade de arquivos que tal sistema pode armazenar.

Além disso, ele descreve as informações sobre quais partes da área de armazenamento já estão sendo utilizadas e quais ainda estão disponíveis, bem como outras informações.

A seguir, listamos as informações contidas no superbloco:

- Tamanho do sistema de arquivos, que faz referência à área de armazenamento do dispositivo ou da partição atual no dispositivo;
- Relação de blocos de armazenamento;

- Quantidade de blocos disponíveis;
- Localização de todos os blocos disponíveis;
- Tamanho da lista de inode: esta lista é inicializada com o propósito de rastrear a maior quantidade possível de arquivos;
- Quantidade de inodes disponíveis no sistema de arquivos;
- Índice do próximo inode disponível na lista de inodes disponíveis.

Tabela de inodes

O inode é definido como uma estrutura de dados que apresenta um sistema de arquivos que adota a semântica de sistemas Unix. Os inodes têm como característica conter dados de arquivos, tais como localização, tipo, proprietário, tempo de acesso, entre outros.

Podemos definir a lista de inodes como uma lista estática. O tamanho da lista não pode ser alterado depois que o sistema de arquivos for criado. O administrador e o tamanho do dispositivo de armazenamento são responsáveis por definir o tamanho desta lista.

A seguir, listamos os tipos de informações incluídas em um inode:

- **ID do proprietário do arquivo:** Refere-se ao UID especificado no arquivo de usuários para identificar o proprietário do arquivo;
- **ID do grupo:** Refere-se ao GID especificado no arquivo de grupos para identificar o grupo do arquivo;
- **Tipo de arquivo:** Responsável por armazenar a especificação do tipo de arquivo, isto é, especifica se o arquivo é um diretório, um link simbólico, um dispositivo de bloco ou caractere ou um pipe;
- **Permissões do arquivo:** Traz informações de permissões de acesso para as classes usuário (owner), grupo (group) e outros (other);
- **Tempo de acesso:** Por meio deste item, podemos obter informações sobre tempo de acesso e modificação dos arquivos, além do número de links, tamanho do arquivo e o local onde os dados estão armazenados no dispositivo.

Bloco de dados

Os blocos de dados estão localizados após os blocos que possuem a lista de inodes e têm a função de armazenar informações administrativas, bem como os dados dos arquivos.

É importante lembrar que um bloco de dados nunca pode pertencer a mais de um sistema de arquivos.

Processos no Linux

Em sistemas Unix-like, o gerenciamento de processos seguirá um modelo padrão, que poderá ser utilizado em todos os sistemas operacionais que seguem o padrão POSIX.

Processos são a base do sistema operacional e são controlados de forma direta pelo próprio kernel, que faz um bom gerenciamento, além de proporcionar muita confiabilidade.

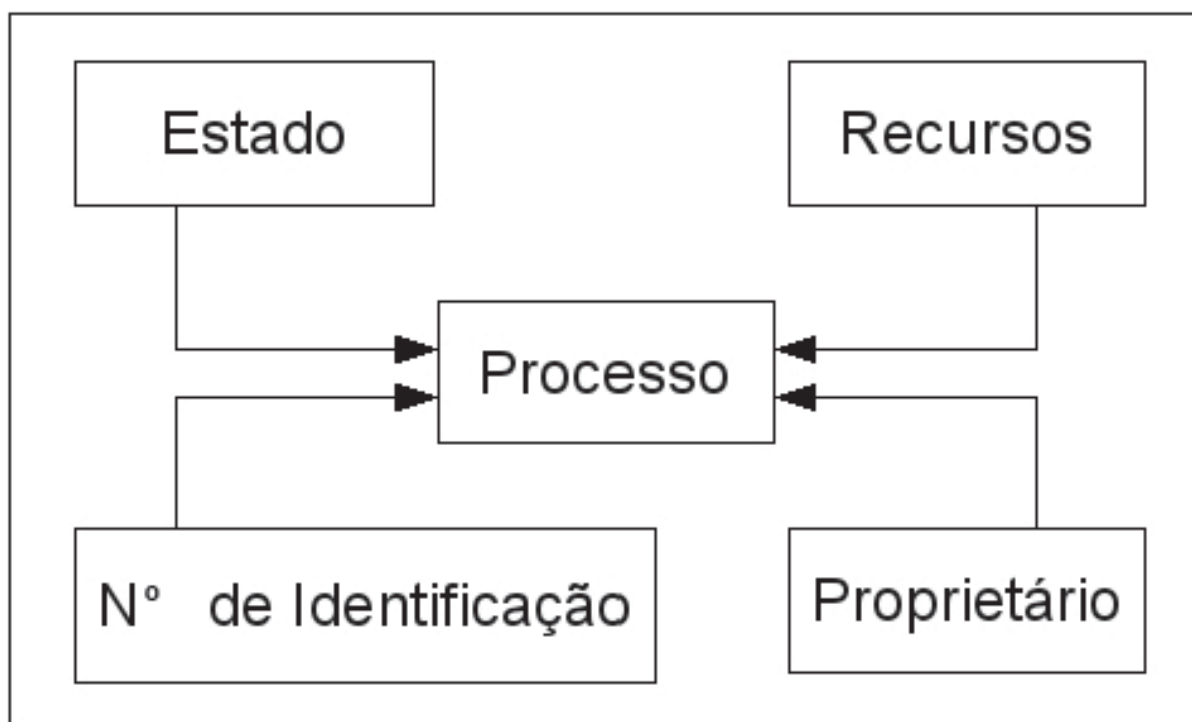
Algumas aplicações, entretanto, nem sempre são desenvolvidas de acordo com os padrões POSIX de gerenciamento de processos e, conseqüentemente, apresentam algum tipo de anomalia que está fora do controle do próprio kernel. Saber controlar de forma manual o funcionamento desses processos, portanto, é essencial para obter o controle real do sistema operacional e resolver problemas provenientes da má utilização ou desenvolvimento.

Primeiramente, é importante definir o que é um processo. Um processo pode ser um programa ou comando que, ao ser executado, passa a consumir determinados recursos do sistema e de hardware. Sempre que houver o consumo de algum tipo de recurso por parte de alguma aplicação, programa, comando, etc., teremos, então, um processo.

Controlar o funcionamento de um processo junto ao kernel é importante para o administrador do sistema. Contudo, antes de controlá-lo, é preciso obter a lista de todos os processos em execução no sistema, para posteriormente realizar algum tipo de gerenciamento específico, como finalizar sua execução, parar temporariamente, dedicar mais atenção aos recursos consumidos para tal processo ou minimizar tais recursos.

Principais propriedades de um processo

Para melhor definirmos um processo e iniciarmos seu gerenciamento, serão abordadas as seguintes propriedades presentes na figura abaixo:



Estado

Todo processo em execução tem um tipo de estado que o define. Por meio desse estado, é possível compreender em qual ciclo de execução tal processo se encontra, uma vez que os processos têm um ciclo de vida em que nascem, executam suas funções e morrem, caso não tiverem mais utilidade ao terminá-las.

Os principais estados são:

- **Running:** O processo se encontra em execução atualmente e consumindo recursos.
- **Stopped:** O processo se encontra parado, não consumindo nenhum tipo de recurso até que seja liberado para Running novamente.
- **Sleeping:** Enquanto não é solicitado, o processo fica hibernando e, portanto, não consome recursos.
- **Zombie:** O processo fica travado, não executa sua função, porém, continua consumindo algum tipo de recurso.

Recursos

A principal definição de um processo é o consumo de determinados recursos, dos quais o mesmo necessita. É importante ressaltar que os recursos nem sempre serão consumidos, pois temos uma variação em relação ao seu estado. Esses recursos podem ser de hardware (memória, processador, etc.), e até recursos do próprio sistema (bibliotecas ou funções do

próprio kernel).

Proprietário

Os processos terão donos, que poderão intervir na forma como estão sendo executados no sistema e também gerenciar seu funcionamento. É importante ressaltar que usuários comuns só podem gerenciar os seus próprios processos, não sendo permitido que eles modifiquem o funcionamento dos processos de outros usuários ou do sistema. O usuário root, que é o administrador, terá permissão para gerenciar qualquer tipo de processo em execução no sistema.

Prioridade

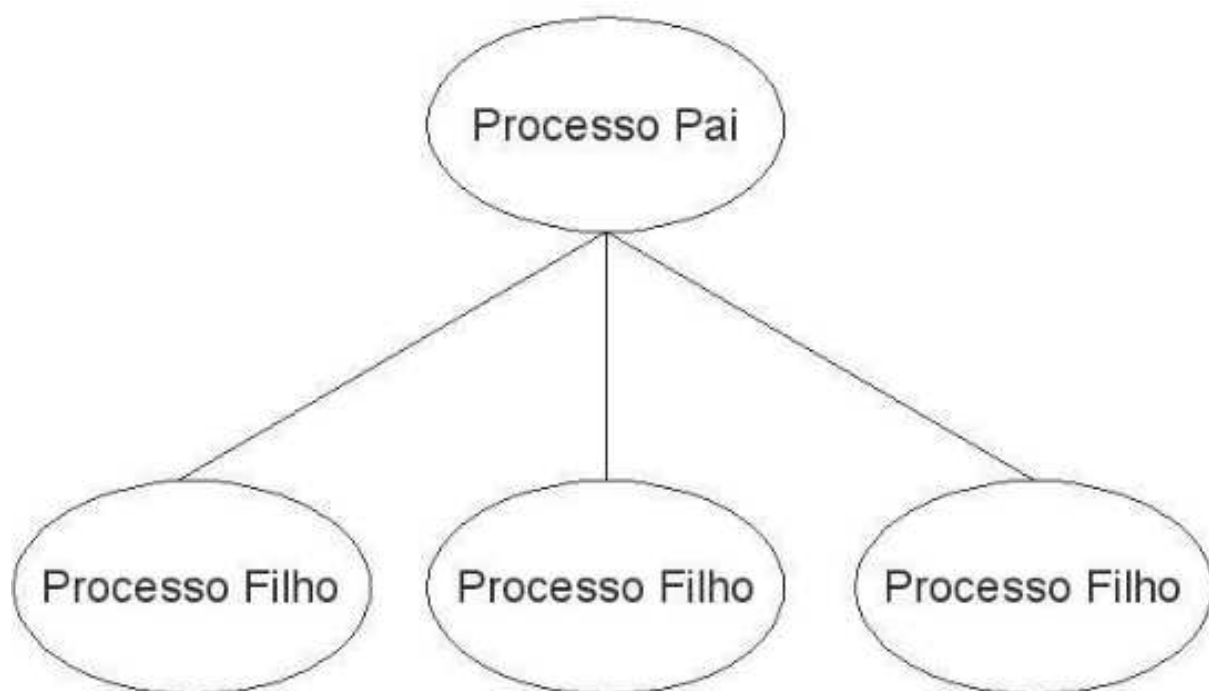
Os processos, quando em execução, terão certos tipos de prioridades, que serão definidos pela quantidade de recursos disponíveis para esses processos. Logo, quanto maior a prioridade, maior será a quantidade de recursos disponíveis.

Sendo assim, a prioridade é uma propriedade também gerenciável e poderá ser controlada, aumentando ou diminuindo recursos para os processos, por meio de um intervalo específico utilizado para definir essa prioridade.

Número de identificação

Para gerenciar os processos, como o estado em que se encontram ou até mesmo suas prioridades de execução, é preciso identificá-los por meio de um número, atribuído em seu início. Esse número de identificação é conhecido como PID (Process Identifier). Atribuído pelo próprio kernel, ele não pode ser utilizado por mais de um processo ao mesmo tempo, mas pode ser reaproveitado posteriormente.

Além do PID (Process Identifier), existe outra notação, chamada PPID (Parent Process Identifier), que corresponde ao número do processo pai do processo atual. Processos são executados em cadeia, desde o momento inicial do carregamento do sistema, no qual o primeiro processo, de nome init e com **PID 1**, é carregado. Em uma relação de processo pai e processo filho, um processo pode gerar outros processos, estabelecendo, assim, um vínculo direto entre eles. Essa relação pode ser observada na figura abaixo:



O processo filho depende, de forma direta, de seu processo pai; assim, se o processo pai parar por algum motivo, o processo filho também tem sua funcionalidade interrompida. Entretanto, se o processo filho parar, o processo pai continuará seu funcionamento normalmente. Essa relação é atribuída a todos os processos em funcionamento.

Listando processos

Para listar e obter informações sobre os processos em execução no sistema, dispomos de diversos comandos. Por meio deles, podemos obter qualquer tipo de informação relacionado aos processos em execução, como PID, estado, prioridade de execução, etc.

ps

O comando **ps** é utilizado apenas para obter informações sobre os processos que estão sendo executados no sistema. Há várias informações que poderão ser obtidas por meio desse comando, além dos simples nomes dos processos em execução, PID, donos, prioridades, tempo de execução, entre outros. É um comando completo, que trabalha com as opções Unix, BSD e GNU para manter uma padronização para administradores que trabalham com outros sistemas Unix-like.

Sua sintaxe é a seguinte:

```
1 | ps [opcoes]
```

A utilização do comando `ps` sem qualquer opção exibe apenas os processos que estão em execução a partir do terminal em que foram executados. Para se obter uma lista mais abrangente dos processos que estão sendo executados a partir de outros terminais (e até mesmo processos de sistema, como daemons), utilizamos as opções do comando `ps`.

O cuidado que se deve ter com o comando `ps` baseia-se na imensa quantidade de opções que ele possui, justamente por aceitar os sistemas Unix, BSD e GNU. A opção Unix `-a`, por exemplo, é bem diferente da opção BSD `a`. Outra característica importante do comando `ps`, que também está ligada, de forma direta, à grande quantidade de opções, é a quantidade de informações que podem ser obtidas em sua execução. As informações são divididas em colunas, e cada opção poderá exibir colunas contendo informações específicas.

Sobre as principais colunas de informações, temos as apresentadas nas tabelas abaixo:

Coluna	Descrição
USER	Nome do usuário dono do processo.
UID	Número de identificação (UID) do usuário dono do processo.
PID	Número de identificação do processo.
PPID	Número de identificação do processo pai.
%CPU	Porcentagem do processador em uso.
%MEM	Porcentagem da memória em uso.
VSZ (Virtual Size)	Tamanho virtual do processo.
RSZ (Resident Set Size)	Indica a quantidade de memória utilizada (em kilobytes).
TTY	Nome do terminal associado ao processo. Caso o valor seja interrogação (?), o processo é daemon, e sua execução é independente de um terminal.
START	Hora em que o processo foi iniciado.
TIME	Tempo de processamento consumido pelo processo.
NI	Valor de prioridade ajustado pelo comando <code>nice</code> ou <code>renice</code> .
PRI	Prioridade real do processo, controlado pelo próprio kernel.

STAT - Define o estado do processo, em que temos os valores:

- **D**: Processo hibernando, mas com dificuldades para voltar a executar suas funções;
- **R**: Executando;
- **S**: Dormindo, suspenso;
- **T**: Parado;
- **X**: Processo finalizado;
- **Z**: Processo Zombie, travado.

STAT - Complementares:

- **>**: Processo sendo executado com prioridade maior que o normal;
- **N**: Processo sendo executado com prioridade menor que o normal;
- **L**: Processo com algum tipo de recurso bloqueado na memória;
- **s**: Líder de sessão;
- **+**: O processo está sendo executado em primeiro plano.

A tabela a seguir apresenta as opções utilizadas com o comando **ps**:

Unix	BSD	GNU	Descrição
-A / -e -a			Exibe uma lista completa dos processos em execução (daemons e processos ligados a algum terminal). Exibe apenas uma lista dos processos ligados a um terminal, exceto processos líderes de sessão (processos pai).
A			Exibe uma lista dos processos ligados a um terminal, incluindo processos líderes de sessão.
-d			Exibe uma lista completa dos processos em execução, exceto processos líderes de sessão.
R			Exibe apenas os processos que têm o estado Running .
-C <processo>			Exibe informações do processo que tenha o nome especificado.
- g<nome>		- group <nome>	Exibe informações dos processos do grupo de nome especificado.
- u<grupo>	U -user	-user	Exibe informações do processo que pertença ao usuário especificado.
- p<pid>	<pid> - <pid>		Exibe informações do processo que tenha o PID especificado.
- t<terminal>	T -tty	-tty	Exibe informações dos processos que estão sendo executados no terminal de nome especificado.
-F	j / l / u / v / x		Exibe mais formatos de colunas na listagem dos processos.
- o<coluna>		o<coluna> format	Define manualmente o formato de colunas desejado na visualização de informações dos processos. Sobre as possíveis colunas, podemos acompanhar a próxima tabela, ou executar o comando ps L .
-H L			Exibe a listagem de processos em formato hierárquico. Exibe o nome dos formatos de coluna visualizáveis com o comando ps .

A tabela adiante apresenta os formatos específicos de colunas, que podem ser utilizados com

a opção **-o**, **o** ou **-format**:

Coluna	Descrição
%cpu	Exibe a coluna de controle de porcentagem de processador em uso.
%mem	Exibe a coluna de controle de porcentagem de memória em uso.
command	Exibe a coluna com o nome do processo em execução.
group	Exibe a coluna, especificando o nome dos grupos que estão relacionados ao processo.
nice	Exibe a coluna de valor de prioridade ajustado pelos comandos nice e renice.
pri	Exibe a coluna com o valor de prioridade real do processo, controlado pelo próprio kernel.
rsz	Exibe a coluna com a quantidade de memória utilizada pelo processo.
start	Exibe a coluna com a hora em que o processo iniciou sua execução.
stat	Exibe a coluna com o estado de execução do processo.
time	Exibe a coluna com o tempo de consumo de recursos pelo processo.
tty	Exibe a coluna com o nome do terminal ao qual o processo está associado.
uid	Exibe a coluna com número de UID relacionado ao dono do processo.
user	Exibe a coluna com o nome do usuário dono do processo.
vsz	Exibe a coluna com o tamanho virtual do processo.

Vejamos o exemplo a seguir:

```

1 | $ ps | head -3
2 |   PID TTY          TIME CMD
3 |   3476 pts/2    00:00:00 bash
4 |   7253 pts/2    00:00:00 ps
5 |
6 |
7 | $ ps aux | head -3
8 | USER  PID   %CPU %MEM    VSZ   RSS TTY  STAT START   TIME COMMAND
9 | root    1      0.0  0.0   33916  3244 ?    Ss   07:35   0:01 /sbin/init
10 | root    2      0.0  0.0      0      0 ?      S    07:35   0:00 [kthreadd]
11 |
12 |
13 | $ ps ax -o user,pid,ppid,%cpu,%mem,stat,start,command | head -3
14 | USER      PID  PPID  %CPU  %MEM  STAT  STARTED      COMMAND
15 | root        1    0   0.0   0.0    Ss   07:35:28    /sbin/init
16 | root        2    0   0.0   0.0    S    07:35:28    [kthreadd]
```

No primeiro exemplo, o comando `ps` foi executado sem nenhuma opção, trazendo como resultado apenas os processos em execução no terminal atual.

Já no segundo exemplo, o comando `ps` foi executado com duas opções BSD, trazendo informações mais detalhadas sobre os processos em execução em todo o sistema.

No último exemplo, o comando `ps` foi executado com a opção de controle de colunas, selecionando somente as informações desejadas.

No exemplo a seguir, o comando `ps aux` traz informações de todos os processos em execução:

```

1 | $ ps aux
2 | USER  PID  %CPU  %MEM  VSZ   RSS  TTY  STAT  START  TIME  COMMAND
3 | root    1    0.0   0.0  33916  3244  ?    Ss    07:35   0:01  /sbin/init
4 | root    2    0.0   0.0    0      0  ?    S     07:35   0:00  [kthreadd]
5 | root    3    0.0   0.0    0      0  ?    S     07:35   0:00  [ksoftirqd/0]
6 | root    5    0.0   0.0    0      0  ?    S<    07:35   0:00  [kworker/0:0H]
7 | root    7    0.0   0.0    0      0  ?    S     07:35   0:14  [rcu_sched]

```

Em seguida, é utilizado o comando `ps faux` para trazer todos os processos, mas agora é possível visualizar quem são os processos pai e filho. Dessa forma fica mais fácil finalizar ou gerenciar toda uma árvore de processos.

```

1 | $ ps faux
2 | USER  PID  %CPU  %MEM  VSZ  RSS  TTY  STAT  START  TIME  COMMAND
3 | root    2    0.0   0.0    0    0  ?    S     07:35  0:00  [kthreadd]
4 | root    3    0.0   0.0    0    0  ?    S     07:35  0:00  _ [ksoftirqd/0]
5 | root    5    0.0   0.0    0    0  ?    S<    07:35  0:00  _ [kworker/0:0H]
6 | root    7    0.0   0.0    0    0  ?    S     07:35  0:15  _ [rcu_sched]
7 | root    8    0.0   0.0    0    0  ?    S     07:35  0:02  _ [rcuos/0]

```

Para visualizar a lista de opções disponíveis, basta executar o comando `ps --help`, como mostrado a seguir:

```

1 | $ ps --help

```

Já com o comando `ps -e f`, é possível visualizar os processos de uma forma mais organizada e com seus respectivos processos filho:

```

1 | $ ps -e f
2 |   PID TTY          STAT       TIME COMMAND
3 |     2 ?            S           0:00 [kthreadd]
4 |     3 ?            S           0:00 _ [ksoftirqd/0]
5 |     5 ?            S<          0:00 _ [kworker/0:0H]
6 |     7 ?            S           0:15 _ [rcu_sched]

```

```
7 | 8 ?      S      0:02  _ [rcuos/0]
```

pidof

O comando **pidof** é utilizado para verificar o PID de um processo em execução, caso o nome do processo seja especificado. Se nenhum processo for indicado, o comando informará o PID de todos os processos em execução. Sua sintaxe é a seguinte:

```
1 | pidof [opcoes] [programa]
```

Descrevemos opções da sintaxe acima na tabela a seguir:

Coluna	Descrição
-s	Leva o comando a retornar somente o primeiro PID encontrado.
-c	Retorna somente o PID de processos localizados num mesmo diretório raiz. Como usuários comuns não têm acesso a informações relacionadas a processos dos quais eles não são donos, essa opção só tem funcionalidade para o superusuário.
-x	Retorna o PID de shells que estejam executando os processos especificados.
-o	Os processos cujo PID sejam informados serão omitidos no retorno do comando.

No exemplo a seguir, o comando **pidof** é utilizado para descobrir o PID do processo **sshd**:

```
1 | # pidof sshd
2 | 8014
```

pstree

Com o comando **pstree**, os processos em andamento no sistema são mostrados no formato de árvore genealógica, com ligações entre processos pais e processos filhos.

Sem argumento nenhum, o comando mostrará todos os processos existentes. Porém é possível determinar quais processos serão visualizados, com a sintaxe seguinte:

```
1 | pstree [opcao] [PID ou usuario]
```

Em conjunto com o comando `less`, a saída do comando `pstree` pode ser melhor visualizada, já que os diagramas em forma de árvore são geralmente grandes demais para exibição em uma tela só. Com o comando `less`, a saída será exibida em telas diferentes e pode ser navegada com a Barra de espaço e a tecla B, que avançam e retornam a visualização respectivamente.

A seguir, veremos algumas opções possíveis na tabela abaixo com o comando **pstree**:

Coluna	Descrição
-p	Exibe os PIDs dos processos.
-h	Destaca em negrito o processo atual e seus ancestrais, caso esse destaque seja suportado pelo sistema.
-n	Leva a árvore a ser organizada por ordem de PID, em vez da ordem alfabética padrão.
-u	Exibe, entre parênteses, os proprietários dos processos, a menos que sejam filhos de processos de um mesmo proprietário.
-l	Impede o truncamento de informações em árvores cuja exibição não caiba na largura da tela.
-a	Exibe os argumentos da linha de comandos dos processos iniciados por um usuário, e não por outro processo.

top

O comando **top** também é utilizado para obter informações dos processos que estão sendo executados e ainda exibe as informações dos processos em tempo real, em nível de monitoramento, sendo possível obter praticamente todas as informações que o comando `ps` exibe. Sua sintaxe é a seguinte:

```
1 | top [opcoes]
```

Outra característica do comando `top` é que, por meio dele, é possível gerenciar os processos do estado de execução em que se encontram, e até mesmo suas prioridades. A imagem a seguir ilustra sua utilização:

```
top - 00:19:12 up 2 days, 8:46, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 52 total, 1 running, 51 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 256972k total, 252276k used, 4696k free, 66892k buffers
Swap: 240932k total, 0k used, 240932k free, 111816k cached
```

PID	USER	PR	NI	UIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
11684	aluno	15	0	2228	1104	856	R	0.3	0.4	0:00.09	top
1	root	15	0	1944	644	552	S	0.0	0.3	0:01.53	init
2	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.03	ksoftirqd/0
4	root	10	-5	0	0	0	S	0.0	0.0	0:03.01	events/0
5	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khelper
6	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	kthread
9	root	20	-5	0	0	0	S	0.0	0.0	0:00.15	kblockd/0
10	root	20	-5	0	0	0	S	0.0	0.0	0:00.00	kacpid
175	root	19	-5	0	0	0	S	0.0	0.0	0:00.00	kseriod
215	root	21	0	0	0	0	S	0.0	0.0	0:00.00	pdflush
216	root	15	0	0	0	0	S	0.0	0.0	0:00.57	pdflush
217	root	10	-5	0	0	0	S	0.0	0.0	0:00.07	kswapd0
218	root	16	-5	0	0	0	S	0.0	0.0	0:00.00	aio/0
782	root	10	-5	0	0	0	S	0.0	0.0	0:00.00	khubd
936	root	11	-5	0	0	0	S	0.0	0.0	0:00.00	scsi_eh_0
1229	root	10	-5	0	0	0	S	0.0	0.0	0:02.78	kjournald
1407	root	20	-4	2704	1152	352	S	0.0	0.4	0:00.45	udevd

O top possui um cabeçalho repleto de informações do sistema e de sua utilização, além de exibir uma lista de processos organizados por consumo de processamento, memória, etc. Como a exibição dos processos é dinâmica, por meio da organização anteriormente citada, os processos ficarão separados por páginas. O comando top trabalha com dois tipos de opções: opções externas, passadas no momento da execução do comando top, e opções internas, que terão efeito com o top já em execução. A seguir, abordaremos de forma detalhada esses tipos de opções:

Opções externas:

Opção externa	Descrição
-d	Define um tempo, em segundos, para a atualização das informações dos processos pelo top.
-i	Ignora processos em estado Zombie na exibição da lista de processos no top.
-u	Exibe apenas os processos que pertencem ao usuário de nome especificado.
-p ...	Inicia o top, monitorando apenas os processos de PID indicado.

Opções internas:

Opção interna	Descrição
enter ou espaço	Atualiza o tempo de monitoramento dos processos no momento desejado, sem que seja preciso aguardar a atualização automática.
h ou ?	Carrega o menu de ajuda do top.
B	Desabilita o recurso de destaque em negrito do top.
d ou s	Redefine o tempo, em segundos, para a atualização automática das informações dos processos monitorados pelo top.
u ou U	Determina que sejam monitorados apenas os processos que pertencem ao dono especificado.
q	Sai do top.
r	Define um novo valor nice de prioridade ao processo de PID especificado.
Z	Altera as cores da janela do top.
W	Salva, de forma permanente, as possíveis alterações feitas no top com os comandos anteriormente apresentados, deixando essas configurações salvas como padrão.
l	Desabilita ou habilita informações de tempo de execução do sistema e a quantidade de usuários autenticados que aparecem no cabeçalho do top.
m	Desabilita ou habilita informações de memória que aparecem no cabeçalho do top.
t	Desabilita ou habilita informações de processos e consumo de processador que aparecem no cabeçalho do top.
z	Desabilita ou habilita o top monocromático.
n #	Define uma quantidade de processos por página.
M	Comando para classificação dos processos no top por uso de memória
N	Comando para classificação dos processos no top por PID
P	Comando para classificação dos processos no top por uso de processador
T	Comando para classificação dos processos no top por tempo de uso de processamento.
k	Envia sinais a processos, pelo número de PID.

htop

O comando **htop** é semelhante ao comando top anteriormente apresentado, porém, apresenta mais interatividade e controle sobre os processos em execução.

Sua sintaxe é a seguinte:

```
1 | htop [opcoes]
```

A tabela a seguir descreve algumas opções utilizadas com htop:

Opção Unix	Descrição
-d	Especifica um tempo, em segundos, para a atualização automática do htop.
-u	Especifica um nome de usuário que terá todos os processos monitorados pelo htop.

No rodapé da página exibida com a execução de htop, há um menu de comandos com a descrição das suas principais funções, conforme ilustra a figura a seguir:

The screenshot shows the htop interface. At the top, system statistics are displayed: CPU usage at 0.7%, Memory at 170/250MB, and Swap at 0/235MB. System tasks are 35 total with 1 running. Load averages are 0.05, 0.01, and 0.00. Uptime is 2 days, 09:06:29.

PID	USER	PRI	NI	UIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
11998	aluno	15	0	2284	1180	936	R	0.0	0.4	0:00.08	htop
1	root	15	0	1944	644	552	S	0.0	0.2	0:01.52	init [31
1407	root	20	-4	2704	1152	352	S	0.0	0.4	0:00.45	udev --daemon
2420	daemon	16	0	1688	368	272	S	0.0	0.1	0:00.00	/sbin/portnap
2677	root	18	0	1628	640	512	S	0.0	0.2	0:00.52	/sbin/syslogd
2683	root	15	0	1576	380	308	S	0.0	0.1	0:00.18	/sbin/klogd -x
2778	root	15	0	1576	568	480	S	0.0	0.2	0:00.02	/usr/sbin/acpid -c
2819	root	18	0	4560	1764	1300	S	0.0	0.6	0:00.08	/usr/sbin/cupsd
2827	messageb	15	0	2248	864	676	S	0.0	0.3	0:00.51	/usr/bin/dbus-daem
2835	haldacmo	18	0	6276	4696	1660	S	0.0	1.6	0:04.29	/usr/sbin/hald
2836	root	17	0	2892	1016	856	S	0.0	0.4	0:00.13	hald-runner
2842	haldacmo	15	0	2020	852	728	S	0.0	0.3	0:00.02	hald-addon-acpi: 1
2847	haldacmo	15	0	2016	860	736	S	0.0	0.3	0:02.25	hald-addon-keyboar
2860	root	16	0	1808	632	544	S	0.0	0.2	2:18.06	hald-addon-storage
2872	root	15	0	2392	1388	680	S	0.0	0.5	0:07.04	/usr/sbin/dhcd -
2887	root	15	0	12180	1944	1696	S	0.0	0.7	0:01.76	/usr/sbin/NetworkM
2879	root	15	0	12180	1944	1696	S	0.0	0.7	0:02.46	/usr/sbin/NetworkM
2894	avahi	15	0	2652	1396	1156	S	0.0	0.5	0:03.15	avahi-daemon: runn

At the bottom, a function key menu is visible: F1Help, F2Setup, F3Search, F4Invert, F5Tree, F6SortBy, F7Nice, F8Nice, F9Kill, F10Quit.

Controlando nível de execução dos processos

Os infinitos processos que estão em execução no nosso sistema são classificados pelo nível em que estão sendo executados. Sobre as classificações de níveis, temos **Foreground** (primeiro plano) e **Background** (segundo plano).

A maioria dos programas, ou seja, processos que estão em execução no sistema, é executada em segundo plano (Background), o que significa que não prendem nenhum terminal durante sua execução. Já os programas em primeiro plano (Foreground), como top e less, quando executados, prendem o terminal, tornando-o indisponível para uso até que sejam finalizados.

É possível gerenciar o plano em que os programas se encontram, permitindo que esses transitem de um plano a outro. Para gerenciar esses processos dessa maneira, existem comandos

específicos. Também é possível obter uma lista dos processos que estão em execução em segundo plano.

Enviando processos para segundo plano

A seguir, veremos algumas formas de enviar processos para segundo plano.

CTRL + Z Uma vez que o programa está sendo executado em primeiro plano, prendendo o terminal atual, é possível enviá-lo para segundo plano, sem precisar finalizar sua execução, utilizando o atalho CTRL + Z.

Ao enviar um processo para o segundo plano com a sequência de teclas CTRL + Z, seu estado é alterado para Stopped (parado), e, por meio do comando bg (apresentado logo a seguir), torna-se possível alterar o estado desse processo de Stopped para Running (executando).

Veja um exemplo da utilização do atalho **CTRL + Z**:

```

1 | # ping www.4linux.com.br
2 | PING www.4linux.com.br (107.170.49.145) 56(84) bytes of data.
3 | 64 bytes from www.4linux.com.br (107.170.49.145): icmp_seq=1 ttl=50 time=143 ms
4 | 64 bytes from www.4linux.com.br (107.170.49.145): icmp_seq=2 ttl=50 time=141 ms
5 | ^Z
6 | [1]+  Parado                  ping www.4linux.com.br
```

& Outra forma de enviar um processo para segundo plano é no momento de sua execução, bastando apenas incluir o caractere & no final do comando que será executado. Isso permite que o processo seja imediatamente enviado para segundo plano, em estado Running, sem precisar utilizar algum outro comando para tal função. É a forma mais prática de enviar programas ao segundo plano.

```

1 | <programa> &
```

O exemplo a seguir esboça essa ação:

```

1 | # ping www.4linux.com.br > log &
2 | [2] 8611
```

No exemplo anterior, assim que o programa é executado, ele já é enviado automaticamente para segundo plano, porém, sem alterações em seu estado de execução, como poderemos acompanhar no comando jobs apresentado a seguir, que lista os processos executados em

segundo plano.

jobs É possível enviar vários programas para o segundo plano e, para obter uma lista dos programas que estão em execução no segundo plano e seu respectivo estado, utilizamos o comando **jobs**. Sua sintaxe é a seguinte:

```
1 | jobs [opcoes]
```

A tabela a seguir mostra opções que podem ser usadas com o **jobs**:

Opção	Descrição
-p	Exibe somente o número de PID dos processos em segundo plano.
-l	Além de exibir as informações normais dos processos em segundo plano, exibe também o número de PID de cada um desses processos.

Todo programa enviado ao segundo plano ganha um número de controle de acordo com a ordem que ele foi enviado para o segundo plano. Esse número não tem relação direta com o PID e pode ser utilizado para trazer tal processo, que atualmente está em execução no segundo plano, novamente para o primeiro plano. O número de controle, também conhecido como **job**, poderá ser utilizado para alterar o estado de programas que estão em segundo plano, de **Stopped** para **Running**, por intermédio do comando **bg**.

O exemplo a seguir ilustra o uso de **jobs**:

```
1 | $ jobs
2 | [1]-  Executando          ping www.4linux.com.br > log2 &
3 | [2]+  Parado              vim
```

Comando bg O comando **bg** é utilizado para alterar o estado de um processo que se encontra em segundo plano, de **Stopped** para **Running**. Também é possível fazer referência aos processos em segundo plano pelos seus respectivos nomes, porém, isso pode gerar problemas, caso processos em segundo plano tenham nomes idênticos. A sintaxe de **bg** é a seguinte:

```
1 | bg [job]
```

Vejamos o exemplo a seguir:

```

1 | $ jobs
2 | [2]-  Parado                vim
3 | [3]   Executando           ping www.4linux.com.br > log2 &
4 | [4]+  Parado                ping www.4linux.com.br > log3
5 |
6 | $ bg 4
7 | [4]+  ping www.4linux.com.br > log3 &
8 |
9 | jobs
10 | [2]+  Parado                vim
11 | [3]   Executando           ping www.4linux.com.br > log2 &
12 | [4]-  Executando           ping www.4linux.com.br > log3 &

```

O exemplo anterior nos mostra a utilização do comando **bg**, em que havia um processo de ordem, em segundo plano, em estado Stopped (Parado). Após a execução do comando **bg** em tal processo, o mesmo teve seu estado alterado para Running (Executando).

```

1 | $ jobs
2 | [2]+  Parado                vim
3 | [3]   Executando           ping www.4linux.com.br > log2 &
4 | [4]-  Executando           ping www.4linux.com.br > log3 &
5 |
6 | $ fg 3
7 | ping www.4linux.com.br > log2

```

Alterando o comportamento dos processos

Após conhecermos comandos que nos permitem obter informações sobre os processos em execução, abordaremos o gerenciamento avançado de processos, que lida com o envio de sinais e permite alterar o comportamento desses por meio de modificações em seus estados de execução ou prioridade.

kill

O comando **kill** é utilizado para enviar sinais aos processos que estão sendo executados pelo sistema. Esses sinais poderão mudar o comportamento dos processos de diversas maneiras, podendo encerrá-los, pará-los ou até mesmo reiniciá-los. Sua sintaxe é a seguinte:

```
1 | kill [-sinal] <pid1> <pid2> ...
```

Para visualizarmos a lista completa de sinais que poderão ser utilizados pelo comando **kill**, devemos digitar **kill -l**. A tabela a seguir descreve os principais sinais utilizados com o comando **kill** no envio de processos:

Sinal	Código	Descrição
SIGHUP	1	Reinicia um serviço (daemon) apenas relendo seu arquivo de configuração.
SIGKILL	9	Finaliza um processo de forma forçada.
SIGTERM	15	Finaliza um processo de forma normal, não forçando ou comprometendo a funcionalidade.
SIGCONT	18	Inicia um processo que esteja em estado Stopped.
SIGSTOP	19	Para momentaneamente a execução de um processo.

Caso nenhum sinal seja especificado ao comando `kill` para envio a algum processo pelo PID, o sinal padrão de envio será o **SIGTERM**.

No exemplo a seguir, após localizar o número de PID relacionado ao programa `cron`, foi enviado o sinal **SIGKILL** ao mesmo:

```

1 | $ ps aux | grep -v grep | grep --color=auto cron
2 | root      1226  0.0  0.0  23656  1052 ?        Ss   Dez18   0:00 cron
3 |
4 | $ sudo kill -9 1226

```

killall

O comando **killall** é muito parecido com o comando **kill** anteriormente apresentado. Ele envia aos processos exatamente os mesmos tipos de sinais que o comando **kill**.

A diferença, entretanto, é que o comando **killall** faz referência aos processos pelo nome, e não pelo número de PID.

Sua sintaxe é a seguinte:

```
1 | killall [opcoes] [-sinal] <processo> <processo> ...
```

Caso nenhum sinal seja especificado ao comando `killall`, o sinal padrão **SIGTERM** é enviado ao processo de nome especificado.

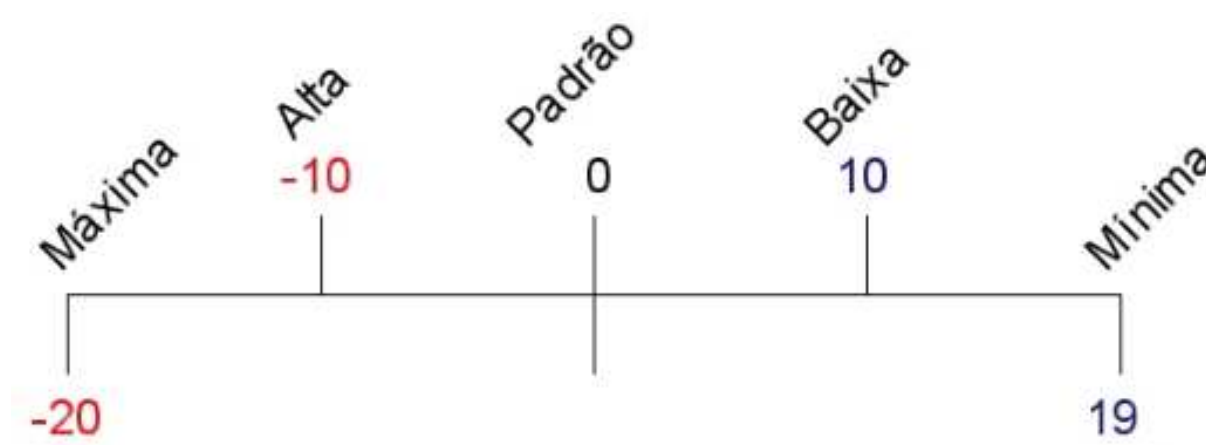
A forma de envio de sinais é exatamente a mesma do comando `kill`. O comando **killall** trabalha com algumas opções interessantes, conforme veremos a seguir:

Unix	GNU	Descrição
-I ("i" maiúsculo)	-ignore-case	Desabilita a característica case sensitive do nome do processo informado.
-i	-interactive	Questiona se o sinal especificado pode ser enviado ou não ao processo.
-u	-user	Envia o sinal determinado a todos os processos pertencentes ao usuário especificado.
-v	-verbose	Reporta informações do envio de sinal ao processo especificado.

É necessário atentar para a utilização do comando `killall`, pois, caso haja processos com nomes idênticos em execução no sistema, todos eles receberão o sinal especificado.

nice

O comando **nice** é capaz de trabalhar com o gerenciamento das prioridades dos processos a serem executados, podendo inicializar um processo com a prioridade desejada. A prioridade a ser definida segue o padrão descrito na figura abaixo:



Sua sintaxe é a seguinte:

```
1 | nice [opcoes [prioridade]] [comando]
```

A principal opção utilizada com o **nice** é **-n**, que define a prioridade conforme o intervalo permitido. A opção GNU equivalente é **—adjustment=**.

Vejamos o exemplo da figura abaixo:

```
debian:~# ps -l NI -e f |grep ntpd
0 S    0   2948   2175  0  78  -2 -    786      - pts/0      0:00      \_ grep ntpd
5 S    0   2946     1  2  78  -2 -   1066     - ?          0:00 ntpd
debian:~# pkill ntpd
debian:~# ps -l NI -e f |grep ntpd
0 S    0   2951   2175  0  78  -2 -    786      - pts/0      0:00      \_ grep ntpd
debian:~# nice -n -10 ntpd
debian:~# ps -l NI -e f |grep ntpd
0 S    0   2955   2175  0  78  -2 -    786      - pts/0      0:00      \_ grep ntpd
5 S    0   2953     1  2  68 -12 -   1066     - ?          0:00 ntpd
debian:~#
```

No exemplo anterior, o comando **ps** foi utilizado para visualizar o processo **ntpd**. Notamos que sua prioridade padrão é **-2**. Depois, o comando **pkill** é utilizado para finalizar o processo **ntpd**. O **pkill** mata todos os processos existentes com o nome especificado.

Depois, iniciamos o processo com a prioridade **-10**. Podemos notar que será adicionado **-10** à prioridade padrão do processo (geralmente essa prioridade é **0**, porém, para o NTP, ela se inicia com **-2**). Por fim, utilizamos novamente o comando **ps** para visualizarmos que a prioridade do processo **ntpd** mudou para **-12**, ou seja, o valor **-2** da inicialização padrão mais **-10**, que foi adicionado.

renice

O comando **renice** também é utilizado para gerenciar prioridades, porém ele tem a capacidade de alterar a prioridade de processos que já estão em execução no sistema. Essa alteração é feita em tempo real, com o processo em execução, e a alteração da prioridade também deve estar de acordo com o intervalo apresentado anteriormente.

Sua sintaxe é a seguinte:

```
1 | renice <prioridade> [opcoes]
```

A tabela a seguir descreve as principais opções utilizadas com **renice**:

Opção	Descrição
-g	Indica um nome de grupo para alterar todos os processos relacionados a ele.
-u	Indica um nome de usuário para alterar a prioridade de todos os seus processos.

Opção	Descrição
-p	Indica o número de PID cuja prioridade se pretende alterar.

Usuários comuns podem gerenciar apenas a prioridade de seus próprios processos.

Considerando que o processo já esteja sendo executado com a prioridade **-12**, vamos utilizar o comando `renice` para reestabelecer o valor de prioridade **-2**, como exibido a seguir na figura:

```

debian:~# ps -l NI -e f |grep ntpd
0 R    0   2970   2175   0   78   -2   -    783      - pts/0        0:00      \_ grep ntpd
5 S    0   2953     1   0   68  -12   -   1066     - ?          0:00 ntpd
debian:~# renice -2 -p 2953
2953: prioridade antiga = -12; prioridade nova = -2
debian:~# ps -l NI -e f |grep ntpd
0 R    0   2973   2175   0   78   -2   -    783      - pts/0        0:00      \_ grep ntpd
5 S    0   2953     1   0   78   -2   -   1066     - ?          0:00 ntpd
debian:~#

```

Para visualizar o resultado da alteração, basta executar o comando **ps** novamente.

nohup

```
1 | nohup <comando>
```

O comando **nohup** é utilizado para ignorar sinais do tipo **SIGHUP** enviados a um determinado processo, em alguma circunstância. Podemos tomar como exemplo um programa que esteja em execução no segundo plano de um terminal qualquer e que poderá continuar sua execução mesmo após o encerramento desse terminal, transformando, indiretamente, o programa em um daemon.

O comando **nohup** foi herdado do Unix. Atualmente, os interpretadores de comandos mais conhecidos, como o Bash, já trazem algumas implementações automatizadas, que não permitem que os processos em segundo plano sejam finalizados, mesmo após o encerramento de uma sessão.

As saídas geradas pelo processo que foi iniciado com o comando `nohup` serão armazenadas (por padrão) em um arquivo de nome **nohup.out**, que é gravado no mesmo diretório em que o comando `nohup` foi inicializado.

O exemplo a seguir mostra essa possibilidade:

```

1 | $ nohup ping www.4linux.com.br &
2 | [3] 11734
3 | $ nohup: ignorando entrada e anexando saída a "nohup.out"

```

No exemplo anterior, o comando **ping** foi executado junto com o comando **nohup**, permitindo, assim, que o mesmo se torne imune ao sinal **SIGHUP**.

Dicas LPI

- O **udev** não super popula o diretório **/dev** do nosso sistema, além de nos proporcionar um método de configuração disponível em **/etc/udev/**.
- O uso dos métodos de **LABEL** ou **UUID** em conjunto com o arquivo **/etc/fstab** nos proporciona uma solução inteligente para o dia-a-dia e para nossa prova.
- Existem muitos comandos para descobrirmos o que temos conectado em nossas máquinas, dentre eles: **lspci**, **lsusb** e **lsusb**.
- As informações providas pelo comando **dmesg** são providas pelo arquivo **/var/log/dmesg**.

Uma listagem completa dos sinais possíveis pode ser vista na seção **STANDARD SIGNALS** do **man 7 signal**. Alguns dos sinais mais utilizados pelo kill podem ser vistos a seguir:

- **SIGHUP (1)** Term Hangup detected on controlling terminal or death of controlling process;
- **SIGKILL (9)** Term Kill signal;
- **SIGTERM (15)** Term Termination signal;
- **SIGCONT (18)** Continue if stopped;
- **SIGSTOP (19)** Stop process

No diretório **/proc** você verá diversos arquivos com informações gerais que o kernel nos fornece, por exemplo os arquivos: **cpuinfo**, **ioport**, **memstat**, **interrupts**, **mounts** e **swaps**. Com o comando **free**, conseguimos ver a quantidade de swap e memória utilizadas, e com o comando **uptime**, conseguimos ver o tempo que a máquina está ligada.

Para rodar um comando, um script ou um programa em **Background**, é só acrescentar o caracter: **&** ao final do comando.

Tópicos para revisão do capítulo

Atente para os tópicos a seguir. Eles devem ser estudados com muita atenção, pois representam os pontos mais importantes do capítulo.

- O subdiretório **/dev** é um sistema de arquivos especial onde estão localizados todos os arquivos referentes aos dispositivos do computador, exceto a interface de rede, manipulada diretamente no nível do kernel;
- Em um sistema GNU/Linux, um dispositivo é acessado por meio de seu respectivo arquivo de dispositivo, que pode ser de caractere, de bloco, FIFO ou socket;

- Dispositivo de armazenamento é qualquer hardware onde se possa gravar dados. Como exemplos de dispositivos de armazenamento, temos CDs, pendrives, HDs, entre outros. Em computadores pessoais, os discos mais comuns, até pouco tempo atrás, eram os do tipo IDE. Atualmente, os mais utilizados são os do tipo SATA;
- O **udev** é um gerenciador de dispositivos que cria os arquivos de dispositivo dinamicamente, conforme os itens de hardware são instalados no computador;
- Um processo pode ser um programa ou comando que, ao ser executado, passa a consumir determinados recursos do sistema e de hardware. Sempre que houver o consumo de algum tipo de recurso por parte de alguma aplicação, programa, comando, etc., teremos, então, um processo;
- Os principais estados de um processo são **Running**, **Stopped**, **Sleeping** e **Zombie**. O funcionamento de um processo só pode ser modificado pelo seu proprietário ou pelo usuário root. As prioridades de execução dos processos podem ser alteradas com os comandos **nice** e **renice**;
- Para visualização de processos, usamos os comandos **ps**, **top**, **htop**, **pidof** e **pstree**. Os três primeiros são utilizados para obter informações sobre os processos que estão sendo executados no sistema. O comando **pidof** informa o **PID**, número de identificação de um processo em execução. Já com o comando **pstree**, os processos em andamento no sistema são mostrados no formato de árvore genealógica;
- A maioria dos programas é executada em segundo plano (**Background**), o que significa que não prendem nenhum terminal durante sua execução. Já os programas em primeiro plano (**Foreground**), como **top** e **less**, quando executados, prendem o terminal, tornando-o indisponível para uso até que sejam finalizados. É possível gerenciar o plano em que os programas se encontram;
- Os comandos **kill** e **killall** são utilizados para enviar sinais aos processos que estão sendo executados pelo sistema. A diferença é que o comando **killall** faz referência aos processos pelo nome, e não pelo número de **PID**, como o **kill**.