

Uma Introdução ao GNU/Linux

Fábio Emilio Costa

Esse documento é licenciado segundo Creative Commons ATRIBUIÇÃO-COMPARTILHAMENTO PELA MESMA LICENÇA 2.5 BRASIL

Você pode:

- copiar, distribuir, exibir e executar a obra
- criar obras derivadas
- fazer uso comercial da obra

Sob as seguintes condições:

- **Atribuição.** Você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante.
- **Compartilhamento pela mesma Licença.** Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.
- Para cada novo uso ou distribuição, você deve deixar claro para outros os termos da licença desta obra.
- Qualquer uma destas condições podem ser renunciadas, desde que Você obtenha permissão do autor.

Qualquer direito de uso legítimo (ou “fair use”) concedido por lei, ou qualquer outro direito protegido pela legislação local, não são em hipótese alguma afetados pelo disposto acima.



Sumário

Lista de Trechos de Código	xvii
Introdução	xxi
1 Um pouco de História	1
1.1 O que é GNU/Linux?	1
1.2 O projeto GNU	2
1.2.1 Richard Stallman	3
1.2.1.1 O conceito de Software Livre	5
1.2.2 O projeto inicia	6
1.3 Linus Torvalds e o Kernel Linux	9
1.4 As distribuições	11
1.5 Algumas distribuições	12
1.5.1 Debian GNU/Linux	12
1.5.2 Red Hat	13
1.5.3 Mandriva	13
1.5.4 Slackware	13
1.5.5 Knoppix	14
1.5.6 Kurumin	14

1.5.7	Ubuntu	14
1.5.8	SuSE	15
1.5.9	Arch Linux	15
1.5.10	Gentoo Linux	15
2	Iniciando com o GNU/Linux	17
2.1	Entrando no sistema	17
2.1.1	Filosofia do sistema	18
2.1.1.1	O usuário <code>root</code>	19
2.1.1.2	Cuidados com o <code>root</code>	19
2.1.2	<i>Login</i>	19
2.2	O <i>Shell</i>	21
2.3	Obtendo ajuda de comandos	22
2.3.1	<code>man</code>	22
2.3.1.1	Navegando pela <i>manpage</i>	24
2.3.2	<code>info</code>	25
2.3.3	Navegando na <i>infopage</i>	25
2.4	Alguns comandos básicos	28
2.4.1	Listando arquivos: <code>ls</code>	28
2.4.1.1	Permissões de arquivo	32
2.4.2	Manipulação de arquivos	34
2.4.2.1	<code>cp</code>	34
2.4.2.2	Coringas <i>wildcards</i> no GNU/Linux	36
2.4.2.3	<code>mv</code>	39
2.4.2.4	<code>rm</code>	40
2.5	Diretórios	40

2.5.1	<code>cd</code>	40
2.5.2	<code>mkdir</code>	41
2.5.3	<code>mv</code>	42
2.5.4	<code>rmdir</code>	42
2.5.5	Aonde Estou? <code>pwd</code>	43
2.6	Passando para superusuário	43
2.6.1	<code>su</code>	44
2.6.1.1	Usando <code>su</code> para acessar como usuário comum	46
2.7	Permissões de Acesso	46
2.7.1	Revisando Permissões de Acesso	46
2.7.1.1	<code>setuid</code>	47
2.7.1.2	<code>setgid</code>	47
2.7.1.3	<i>sticky bit</i> — Bit de cola	48
2.7.2	Mudando permissões — <code>chmod</code>	49
2.7.2.1	Referência Octal	49
2.7.2.2	Referência Simbólica	50
2.7.3	Mudando o dono do arquivo — <code>chown</code>	51
2.7.4	Mudando o grupo dono do arquivo — <code>chgrp</code>	53
2.7.5	Definindo as permissões padrão — <code>umask</code>	55
2.8	Vendo o conteúdo de arquivos	57
2.8.1	<code>cat</code>	57
2.8.2	Visualizando arquivos grandes: <code>more</code> e <code>less</code>	58
2.8.2.1	Entrada padrão, Saída padrão, redirecionamento e <i>pipes</i>	59
2.8.3	Vendo o final de arquivos: <code>tail</code>	60
2.8.4	Acompanhando arquivos: <code>watch</code>	61

2.9	Ligações	62
2.9.1	O que são ligações?	62
2.9.2	Vantagens e desvantagens das ligações	62
2.9.3	O comando <code>ln</code>	63
2.10	Apelidos para comandos: <code>alias</code>	64
3	Estrutura de diretórios do GNU/Linux	65
3.1	Entendendo os diretórios em GNU/Linux	65
3.1.1	O diretório ‘.’	66
3.1.2	O diretório ‘..’	67
3.1.3	O diretório ‘~’	67
3.2	A estrutura padrão de diretórios	69
3.2.1	<code>/usr</code>	69
3.2.1.1	<code>/usr/bin</code>	70
3.2.1.2	<code>/usr/doc</code>	70
3.2.1.3	<code>/usr/lib</code>	71
3.2.1.4	<code>/usr/src</code>	71
3.2.1.5	<code>/usr/share</code>	72
3.2.1.6	<code>/usr/local</code>	72
3.2.2	<code>/sbin</code>	72
3.2.3	<code>/boot</code>	73
3.2.4	<code>/root</code>	73
3.2.5	<code>/home</code>	73
3.2.6	<code>/etc</code>	73
3.2.7	<code>/var</code>	75
3.2.8	<code>/mnt</code>	75

3.2.9	/opt	77
3.2.10	/dev	77
3.2.11	/proc	77
4	Acessando dispositivos no GNU/Linux	79
4.1	Conceito de “montagem de dispositivos”	79
4.2	Pontos de montagem	80
4.2.1	Dispositivos removíveis e pontos de montagem	82
4.3	Montando e desmontando dispositivos: <code>mount</code> e <code>umount</code>	82
4.4	O <code>/etc/fstab</code>	86
5	Administrando o sistema	89
5.1	Criando usuários: <code>adduser</code>	90
5.1.1	O cuidado com os UID	92
5.1.2	O diretório <code>/etc/skel</code>	93
5.1.3	O arquivo <code>/etc/passwd</code> e o arquivo <code>/etc/shadow</code>	94
5.1.3.1	O arquivo <code>/etc/shadow</code>	96
5.2	Criando grupos: <code>addgroup</code>	97
5.2.1	O cuidado com os GID	98
5.2.2	O arquivo <code>/etc/group</code>	98
5.2.3	Usando <code>adduser</code> para adicionar um usuário a um grupo extra	100
5.2.4	Acessando documentos com outra identificação de grupo: <code>sg</code> , <code>newgrp</code> e <code>id</code>	102
5.2.5	Alterando senhas: o comando <code>passwd</code>	103
5.2.6	Removendo usuários e grupos: <code>userdel</code> e <code>groupdel</code>	104
5.3	Entendendo processos e <code>jobs</code>	105

5.3.1	O que é um processo	105
5.3.1.1	Processos de primeiro e segundo plano	106
5.3.2	Diferença entre programas multi-thread e programas de múltiplos processos	107
5.3.3	Estado de processos	109
5.3.4	O comando ps	110
5.3.5	Mandando um processo para segundo plano e o trazendo de volta: bg e fg	112
5.3.6	Vendo os processos de segundo plano: jobs	115
5.3.7	“Matando” processos: kill e killall	116
5.3.8	Aumentando a prioridade do processo: nice e renice	118
5.3.9	“Congelando” um processo: nohup	119
5.4	<i>Backup</i>	120
5.4.1	Os utilitários tar , gzip e bzip2	121
5.4.2	Criando um <i>backup</i> com o utilitário tar	122
5.4.3	Recuperando um <i>backup</i> com o utilitário tar	123
6	Rede e Internet	125
6.1	Bases do TCP/IP	126
6.1.1	História da Internet e do TCP/IP	126
6.1.2	Definindo um IP	127
6.1.3	Definindo máscara de sub-rede	128
6.1.4	Endereços especiais	130
6.1.5	Classes de rede	130
6.1.6	IPs privativos ou “inválidos”	132
6.2	Conectando uma máquina a uma rede IP: os comandos ifconfig e route	134

6.2.1	Explicando o comando <code>ifconfig</code>	135
6.2.2	Explicando o comando <code>route</code>	137
6.2.3	Um exemplo de configuração de rede	138
6.3	Arquivos de informações sobre redes	140
6.3.1	<code>/etc/HOSTNAME</code>	141
6.3.2	<code>/etc/hosts</code>	141
6.3.3	<code>/etc/networks</code>	142
6.4	Arquivos de <i>resolver</i>	146
6.4.1	O arquivo <code>/etc/host.conf</code>	147
6.4.2	O arquivo <code>/etc/resolv.conf</code>	148
6.5	O arquivo <code>/etc/rc.d/init.d/network</code> e o arquivo <code>/etc/sysconfig/network</code>	150
6.6	Alguns sistemas de troca de arquivos	151
6.6.1	NFS	159
6.6.1.1	Configurando um <i>share</i> NFS	159
6.6.1.2	Acessando um <i>share</i> NFS	161
6.6.2	FTP	163
6.6.2.1	Configurando um servidor FTP	165
6.6.2.2	Acessando um servidor FTP	168
6.6.2.3	Modos de transferência no FTP	169
6.6.3	SMB (SaMBa)	172
6.6.3.1	Configurando um <i>share</i> SaMBa	172
6.6.3.2	A seção <code>[global]</code>	175
6.6.3.3	A seção <code>[homes]</code>	184
6.6.3.4	A seção <code>[printers]</code> e configuração para im- pressão	184

6.6.3.5	Configurando as seções dos compartilhamentos	185
6.6.3.6	Habilitando um usuário/máquina no SaMBa .	185
6.6.3.7	Acessando um <i>share</i> SaMBa	185
6.7	Introdução ao Firewall	185
7	Editores de texto	187
7.1	Por que é importante manipular um editor de texto	188
7.1.1	Qual deles é o melhor?	188
7.2	Introdução ao vi	189
7.2.1	Edição e navegação por arquivos	191
7.2.2	Abrindo e salvando arquivos	194
7.2.3	Cortar, Copiar, Colar e Apagar	195
7.2.4	Truques de edição	196
7.2.5	Mudando a codificação do arquivo	196
7.2.6	O Arquivo <code>/.exrc</code> ou <code>/.vimrc</code>	197
7.3	Introdução ao emacs	198
7.3.1	A História do emacs	198
7.3.1.1	Convenções do emacs	202
7.3.2	Edição e navegação por arquivos	203
7.3.3	Abrindo e salvando arquivos	204
7.3.4	Pesquisa e substituição	207
7.3.5	Cortar, Copiar, Colar e Apagar	207
7.3.6	Truques de edição	208
7.3.7	Mudando a codificação do arquivo	209
7.3.8	O arquivo <code>.emacs</code>	209

8	Interfaces Gráficas	213
8.1	O que é o X-Windows	214
8.2	Configurando o X-Windows	214
8.3	O arquivo <code>/etc/X11R6/Xorgconf</code>	214
8.4	O arquivo <code>.Xresources</code>	214
8.5	Algumas interfaces interessantes	214
8.5.1	KDE	214
8.5.2	GNOME	214
8.5.3	WindowMaker	214
8.5.4	IceWM	214
8.5.5	Blackbox	214
9	Instalando Programas	215
9.1	Programas em GNU/Linux	216
9.2	Compilando do Fonte	216
9.2.1	Seqüência mágica	216
9.2.2	O utilitário <code>make</code>	216
9.2.3	Porque no <code>/usr/local</code> e não no <code>/usr</code>	216
9.2.4	Dificuldades de compilar no fonte	216
9.3	Sistemas de empacotamento	216
9.3.1	RPM - RedHat Package Manager	216
9.3.2	DEB	216
9.3.3	TGZ	216
9.4	Atualização via Internet	216
9.4.1	<code>apt</code>	216
9.4.2	<code>smart</code>	216

10 Serviços e inicialização	217
10.1 O que são “serviços”	217
10.1.1 Como “levantar” serviços	217
10.2 Um pouco sobre a inicialização do GNU/Linux	217
10.2.1 O arquivo <code>/etc/inittab</code>	217
10.2.2 Inicialização System V e BSD	217
10.2.3 <i>Runlevels</i>	217
10.2.4 <i>Login</i> e <code>ttys</code>	217
11 Ajuda	219
11.1 Como proceder com ajuda?	220
11.2 Porque tomar nota?	220
11.3 O diretório <code>/var/log</code>	220
11.4 Os comandos <code>uname</code> e <code>dmesg</code>	220
11.5 O log do sistema (<i>syslog</i>)	220
11.6 Fontes de ajuda:	220
11.6.1 Sites de suporte técnico	220
11.6.2 Fóruns	220
11.6.3 Listas de discussão	220
11.6.3.1 Um pouco sobre Netiqueta	220
11.7 Suporte telefônico	220
11.8 Documentações online	220
12 Mantendo-se atualizado	221
12.1 Sites de informação	221
12.2 Newsletters	221

A	Licenciamento dessa obra
---	--------------------------

223

Lista de Tabelas



2.1	Seções de <i>manpages</i>	24
2.2	Permissões simbólicas	51
2.3	Permissões no umask	56
2.4	Símbolos de redirecionamento	60
4.1	Alguns exemplos de nomes de dispositivos	83
4.2	Alguns sistemas de arquivos que podem ser acessados pelo GNU/Linux	84
5.1	Valores típicos do campo password do /etc/passwd	95
5.2	Sinais do sistema segundo POSIX	117
6.1	Tabela de IPs privativos	133
6.2	Prefixos de Dispositivos de Rede	136
6.3	Comandos FTP	170

Lista de Figuras

3.1	Exemplo de estrutura de diretórios	67
3.2	Estrutura padrão do GNU/Linux	70
3.3	Estrutura do diretório <code>/usr</code>	71
3.4	Estrutura do diretório <code>/etc</code>	74
3.5	Estrutura do diretório <code>/var</code>	76
3.6	Exemplo de uma estrutura de pontos de montagem em <code>/mnt</code> .	76
6.1	NAT e IPs privados	133
6.2	Rede exemplo <code>hogwarts</code>	139
6.3	Redes exemplo <code>hogwarts</code> , <code>durmstrang</code> e <code>beauxbatons</code>	145

Lista de Trechos de Código

2.1.1 Exemplo de mensagem de <i>Login</i>	20
2.1.2 <i>Prompt</i> do <i>password</i>	20
2.1.3 <i>Prompt</i> do <i>shell</i>	21
2.3.1 Exemplo do comando man	22
2.3.2 Resultado de man bash	23
2.3.3 Exemplo do comando man com número de seção	24
2.3.4 Exemplo do comando info	25
2.3.5 Resultado do comando info coreutils cat	26
2.3.6 Exemplo do comando info -apropos	26
2.4.1 Exemplo do comando ls	29
2.4.2 Exemplo do comando ls -lh	31
2.4.3 Exemplo do comando cp -v	36
2.4.4 Listagem de exemplo para mostrar coringas	37
2.4.5 Listagem de exemplo com o coringa *	38
2.4.6 Listagem de exemplo com o coringa ?	39
2.4.7 Listagem de exemplo com o uso de dois ?	39
2.5.1 Exemplo do comando mkdir -pv	42
2.5.2 Exemplo do comando rmdir -pv	42
2.5.3 Exemplo do comando pwd	43
2.6.1 Exemplo do comando su	45
2.7.1 Exemplo do comando su	48
2.7.2 Lista de um diretório antes de ter seu usuário dono trocado . .	52
2.7.3 Lista de um diretório após troca de posse de alguns arquivos .	53
2.7.4 Lista de um diretório após troca maciça de posse de arquivos .	54
2.7.5 Lista de um diretório após troca de grupos	55
2.8.1 Exemplo do comando cat	57
2.8.2 Exemplo do comandos cat -n e cat -b	58
2.9.1 Exemplo de uma ligação simbólica	63

4.3.1 Exemplo do comando <code>ls</code> em um dispositivo “montado”	85
4.4.1 Exemplo de <code>/etc/fstab</code>	86
5.1.1 Exemplo de <code>/etc/passwd</code>	95
5.2.1 Exemplo de <code>/etc/group</code>	99
5.2.2 Exemplo de alteração em <code>/etc/group</code>	100
5.2.3 Exemplo do uso do comando <code>adduser</code> para adicionar usuários a outros grupos, assim como o resultado desse uso	101
5.3.1 Exemplo do comando <code>ps aux</code>	113
5.3.2 Exemplo de resultado de  +  e do seu número do trabalho (<i>job id</i>)	114
5.3.3 Exemplo da saída de um comando <code>jobs</code>	115
6.3.1 Exemplo de <code>/etc/HOSTNAME</code>	141
6.3.2 Exemplo de <code>/etc/hosts</code>	142
6.3.3 Exemplo de <code>/etc/hosts</code> separado por linhas	143
6.3.4 Exemplo de <code>/etc/hosts</code> separando nomes de máquina e <i>aliases</i>	143
6.3.5 Exemplo de <code>/etc/network</code> para a rede da Figura 6.3	144
6.3.6 Exemplo de <code>/etc/hosts</code> para a rede da Figura 6.3	146
6.4.1 Exemplo de <code>/etc/hosts.conf</code>	148
6.4.2 Exemplo de <code>/etc/resolv.conf</code>	150
6.5.1 Exemplo de <code>/etc/rc.d/init.d/network</code>	152
6.5.2 Exemplo de <code>/etc/rc.d/init.d/network</code> (Continuação)	153
6.5.3 Exemplo de <code>/etc/rc.d/init.d/network</code> (Continuação)	154
6.5.4 Exemplo de <code>/etc/rc.d/init.d/network</code> (Continuação)	155
6.5.5 Exemplo de <code>/etc/rc.d/init.d/network</code> (Continuação)	156
6.5.6 Exemplo de <code>/etc/rc.d/init.d/network</code> (Continuação)	157
6.5.7 Exemplo de <code>/etc/sysconfig/network</code>	158
6.6.1 Exemplo de <code>/etc/proftpd.conf</code>	166
6.6.2 Exemplo de <i>script</i> com conexão FTP	169
6.6.3 Exemplo de <code>/etc/samba/smb.conf</code>	174
7.2.1 Janela inicial do <code>vim</code>	191
7.2.2 Janela do <code>vim</code> em modo de edição	192
7.3.1 Janela inicial do <code>emacs</code>	201
7.3.2 Janela do <code>emacs</code> com <i>buffer</i>	205
7.3.3 Exemplo de arquivo <code>/.emacs</code>	211

Introdução

GNU/Linux (ou apenas *Linux*) não é mais *apenas* o futuro, mas sim também é uma alternativa séria e viável para ambientes computacionais que precisem de alta *performance* e confiabilidade mas não possuem condições para montar uma infraestrutura baseada em sistemas como o Windows e a maioria das variantes do Unix. Baseado em *software livre*, ele é barato, confiável e poderoso, além de oferecer a oportunidade de aprendizado com o seu código fonte. Porém, ele não é exatamente simples de aprender, uma vez que são muitos comandos, programas e utilitários (mais de 3000 em uma distribuição GNU/Linux típica atualmente)¹.

Essa é uma apostila básica do GNU/Linux, que foi produzida com o objetivo de servir de referência para um curso básico no mesmo, e tem como objetivo ser *apenas* uma introdução ao uso do GNU/Linux. O objetivo final dessa apostila é oferecer as condições para que a pessoa possa seguir a diante no GNU/Linux sem problemas. Em resumo, o objetivo aqui é oferecer o “caminho das pedras”, explicando os principais recursos do mesmo de maneira rápida e sucinta.

Essa apostila foi desenvolvida para ser adotada em um curso de 20 horas e sem ser voltada a nenhuma distribuição em especial, podendo ser adotada aquela que o tutor (ou mesmo o aluno) achar mais adequada. A idéia é que o aluno saia com alguns conceitos e comandos úteis para “caminhar por suas próprias pernas”, além de contar com a sugestão de recursos úteis e comandos alternativos para o aluno pesquisar.

Não é minha intenção aqui mostrar todas as potencialidades e comandos do GNU/Linux, até porque não seria possível fazer isso. Como exemplo,

¹Abril de 2006

o projeto *Linux Documentation Project* (LINUX DOCUMENTATION PROJECT, 2006) já disponibiliza mais de 20 mil páginas em documentação sobre GNU/Linux, boa parte delas traduzida para outros idiomas (Português do Brasil incluído) e ainda assim é pouco. Existem na Internet bons guias de diversos tipos, desde obras veteranas como o *The Linux Manual* (CINEIROS, 2005) de Hugo Cineiros, existente desde Novembro de 1997 e que (apesar do nome) foi o primeiro documento de grande nível escrito sobre o GNU/Linux no Brasil, a obras novas como o “Entendendo e Dominando o Linux” (MORIMOTO, 2005), de Carlos E. Morimoto, “criador” da distro *Kurumin Linux* e referências completas como o “Guia Foca GNU/Linux” (SILVA, 2005a), de Gleydson Mazioli da Silva, uma verdadeira referência sobre o GNU/Linux.

Considero essa apostila a minha “paga” ao movimento do Software Livre, que ofereceu esses sistemas de altíssima qualidade como o GNU/Linux, sendo que este modesto documento espero que chegue aos pés das obras citadas anteriormente.

Sobre a licença

Essa apostila é licenciada segundo a Creative Commons, em sua versão 2.5, com os atributos *Atribuição-Compartilhamento pela Mesma Licença*. De maneira mais direta, isso quer dizer que:

1. Você pode usar essa apostila em cursos comerciais. Não me importo muito com isso, mas gostaria de um reconhecimento, principalmente pela indicação do autor (eu). Uma cópia dos materiais usados ou um “agradinho” também não é nada mal. :-P;
2. Você pode personalizar essa apostila para suas necessidades, além de poder gerar *slides* e apresentações especiais conforme suas necessidades. Porém, esses materiais *devem* ser divulgados segundo as licenças Creative Commons. Cá entre nós, não é nada justo que você pegue meu material, crie em cima e saia dizendo que é seu, né? :-P

Se você precisar de contato para maiores esclarecimentos, agradecimentos, comentários, reclamações e sugestões, me envie um *email* para

fabiocosta0305gmail.com.

Agradecimentos

Queria agradecer a:

- Ao meu amigo Leonardo Prado (DNA), o primeiro a ver essa apostila;
- Aos amigos do Serviço Municipal de Vigilância Sanitária da Prefeitura de Ouro Fino/MG (VISA), pela compreensão;
- Aos professores Eduardo Augusto Cosa, Takaite Takehara, Thiago Zucarelli e Tarcísio Nunes, pelo estímulo;
- À professora Dalva Gonzales Santiago, coordenadora do Curso de Desenvolvimento de Software da ASMEC, pelo espaço aberto;
- A Linus Torvalds, Richard Stallman e outros, pelo software livre, que está cada dia melhor;
- Ao Gleydson Mazioli da Silva, pelo “Guia Foca GNU/Linux”(SILVA, 2005a), que serviu de referência e inspiração a essa apostila;
- Ao Donald Knuth, pelo T_EX e ao Leslie Lamport, pelo L^AT_EX, poderosos sistemas de tipografia aonde essa apostila foi produzida;
- A Jean-Michel Jarre, Enya, Pato Fu, Loreena McKennitt, Silly Wizard, Kitaro, Kraftwerk, Ceolbeg, Wolfstone, Legião Urbana, John Willians, Manu Chao, Mano Negra, Hot Pants e outros, pelas músicas que acompanharam o desenvolvimento desse material;
- Aos meus pais, pela paciência e força nos momentos difíceis;
- A Deus, pela vida que Ele me deu e dá todos os dias;

Sobre o autor

Fábio Emilio Costa é Tecnólogo em Desenvolvimento de Software pelas Faculdades ASMEC de Ouro Fino/MG e Técnico de Processamento de Dados pela Escola Agrotécnica Federal de Inconfidentes (EAFI). Desenvolvedor nas linguagens C/C++, PHP, Python, Ruby, Delphi, Visual Basic, Java e Clipper, atualmente trabalha como Analista de Software Básico no Serviço Federal de Processamento de Dados (SERPRO), em São Paulo. Já atuou como consultor em Desenvolvimento de Software, Programador e Palestrante em GNU/Linux, tendo conhecido o primeiro em 1998. Já trabalhou com as distribuições Conectiva/Mandriva, Slackware, Kurumin, Ubuntu e SuSE. Tem 28 anos.

Produzido em L^AT_EX

Capítulo 1

Um pouco de História

Parece bobagem, mas muitas vezes, precisamos entender o começo das coisas para entendermo-as e apreciarmo-as. Esse capítulo não é obrigatório, e nem vai o tornar mais ou menos conhecedor do GNU/Linux. Porém, vai lhe dar uma retrospectiva do que é que aconteceu até agora, e portanto irá lhe oferecer um entendimento sobre o que você verá e qual é o seu valor.

Bem, chega de delongas. Vamos ao que interessa. Vamos a uma viagem na história... Na história do Software Livre.

1.1 O que é GNU/Linux?

O que a maioria das pessoas entendem como *Linux* é o sistema operacional GNU/Linux. Na verdade, o GNU/Linux é a soma do *kernel*¹ *Linux* com as ferramentas do projeto GNU e vários pacotes de outras fontes, agrupadas de forma a criar um poderoso sistema operacional livre. Em geral, para facilitar as coisas, muitos chamam essa combinação apenas de *Linux*. De qualquer modo, a não ser em caso contrário, não importa o termo que usarmos, estaremos falando sempre do sistema operacional GNU/Linux.

¹*Kernel* é o coração do sistema operacional. Gerencia e controla o acesso a sistemas de arquivos, memória, processos em execução e o acesso aos dispositivos e periféricos, entre outras atribuições. Um erro no kernel pode representar falha grave no sistema operacional. Muitas vezes, erros diretos no boot do sistema são causados por falhas no kernel.

O GNU/Linux já é muito usado em vários ambientes e empresas, principalmente pela sua portabilidade: pelo código livre, o GNU/Linux já foi portado do x86 (PCs) para, entre outras, as seguintes plataformas:

- Compaq Alpha AXP;
- Sun SPARC e UltraSPARC;
- Motorola 68000, PowerPC e PowerPC64 (Amiga e Macintosh);
- Intel ARM;
- Hitachi SuperH;
- IBM S/390 (*mainframes*);
- MIPS;
- HP PA-RISC;
- Intel IA-64 (Itanium);
- DEC VAX;
- AMD x86-64 (Athlon64);

Vamos então desvendar as partes do que forma o GNU/Linux, começando pela mais antiga: o projeto GNU.

1.2 O projeto GNU

O projeto GNU tem como objetivo “*desenvolver um sistema operacional completo, compatível com o UNIX, que fosse software livre: o sistema GNU. (GNU é um acrônimo recursivo² para ‘GNU Não é UNIX’ e é pronunciado*

²*Recursão* é uma técnica de programação aonde uma rotina chama a si mesma em seqüência. Alguns problemas matemáticos famosos que podem ser resolvidos através da recursão são a Série de Fibonacci (aonde um número fx é o resultado da soma dos dois números anteriores ($f(x) = f(x-1) + f(x-2)$, onde $f(1) = 1$ e $f(2) = 1$) e o cálculo de um fatorial ($n! = (n-1)! \times n$).

como ‘guh-noo.’)”(PROJETO GNU, 2006b). Esse projeto envolve milhares de programadores de todo o mundo, sendo que ele já substituiu todos os principais componentes do Unix por versões livres.

Esse projeto se iniciou em 1984 com um homem de certa forma visionário, Richard Matthew Stallman, e vem continuando até hoje a aperfeiçoar as suas ferramentas, que atualmente são consideradas, em muitos casos, o padrão *de facto* do mundo Unix.

1.2.1 Richard Stallman

Richard Stallman, o fundador do projeto GNU, começou sua carreira de desenvolvedor de software desenvolvendo sistemas para o Laboratório de Inteligência Artificial (*AI Lab*) do MIT — *Massachusetts Institute of Technology* (Intituto de Tecnologia de Massachussets). Nesse ambiente, na época frutífero para a tecnologia, Stallman desenvolveu-se em meio a uma comunidade de *hackers*, segundo o sentido original da palavra, definido por Eric S. Raymond(RAYMOND, 2006)³. Nessa comunidade, a troca dos códigos-fontes (isso é, as “receitas de bolo” capazes de gerar um programa *binário* que poderá ser usado no computador) era saudável e até vital, pois na época a troca de software binário não era fácil, por causa da grande quantidade de plataformas de *hardware* incompatíveis entre si. Então, não existia a necessidade do conceito de *software livre*, uma vez que todo o software era livre.

Stallman já vinha percebendo a um certo tempo que a cultura *hacker* estava morrendo, e isso era ruim. Mas a coisa foi pior ainda quando, uma certa vez, o *AI Lab* recebeu uma recém criada impressora *laser* da Xerox. Isso era normal, pois muitos bons códigos eram recebidos em troca pelas grandes corporações, que deixavam os fontes dos *drivers* dos periféricos doados às grandes instituições. E Stallman vivia feliz, até que certa vez, depois de esperar por horas um trabalho que tinha mandado imprimir, foi ver o que havia acontecido. Ele veio a descobrir que a impressora estava obstruída com papel. Stallman ficou chateado, pensando que de certa forma ele podia ter perdido menos tempo se a impressora alertasse o fato de estar obstruída. E

³Nessa definição, um *hacker* é uma pessoa dotada de grande talento e desejo de aprender em uma área qualquer do conhecimento humano. Embora esse seja normalmente relacionado à informática, isso não é obrigatório. Não confundir aqui com invasores de sistemas.

ele procurou então, como viria a ser definido no futuro por Raymond, “coçar a própria sarna”(RAYMOND, 2000).

O problema foi que a Xerox mandou apenas *drivers* binários para a impressora. Stallman pensou em solicitar o fonte para a Xerox, pois pensou, como era costume na época, que a Xerox não perderia a oportunidade de ter um programador de ponta desenvolvendo *drivers* para ela, sem custo nenhum...

...mas a Xerox negou.

O pior para Stallman foi quando ele pediu ajuda a um amigo da Carnegie-Mellon University (CMU) o acesso ao código, pois havia descoberto que este amigo tinha o código da impressora. Ele pediu o código para o corrigir...

...e seu amigo o negou, alegando um Acordo de Sigilo (*Non-Disclosure Agreement* – NDA).

Isso chocou terrivelmente Stallman, que meio que viu no evento o tiro de misericórdia contra a comunidade *hacker* original. Como ele viria a dizer em sua biografia *Free as in Freedom*: “Esse foi meu primeiro encontro com Acordos de Sigilo, e me ensinou que sempre temos vítimas neles. No caso, eu fui a vítima. [Eu e todo o laboratório] fomos vítimas”(WILLIAMS, 2002).

Stallman gosta de retratar que esse foi o momento de decisão por parte dele. Como Stallman descreveu várias vezes, ele tinha algumas opções. A primeira seria desenvolver software proprietário, sabendo que poderia ganhar dinheiro, mas iria dividir as pessoas. A segunda seria abandonar o desenvolvimento de *software*, mas ele sabia que isso, embora o deixasse “com a consciência tranqüila”, seria uma perda de tempo e de talento.

Mas ainda havia uma alternativa.

Ele era um programador de ponta, da elite, um dos melhores do mundo. Ele poderia aplicar seus talentos para construir software que não estivesse nem nunca ficaria travado por Acordos de Sigilo.

Então ele começou a escrever software livre.

1.2.1.1 O conceito de Software Livre

É importante salientar aqui o que vem a ser Software Livre.

A idéia por trás do software livre é que o software deve ser livre *para o usuário final* acima de tudo. De nada adianta que o software seja livre para o desenvolvedor se um desenvolvedor puder pegar o código, fazer alterações (muitas ou poucas ou até mesmo nenhuma), e “fechar” o código por meio de NDAs (“*Non-Disclosure Agreements*” – Acordos de Sigilo) ou EULAs (“*End-User License Agreements*” – Acordos de Licença do Usuário Final).

Perceba que isso não impede que o software seja vendido. Como Stallman diz em seu artigo para o livro *Open Sources : Voices from the Open Source Revolution*(STALLMAN, 1999):

Como o livre no caso se relaciona a liberdade, e não a preço, não existe nenhuma contradição em vender cópias de software livre. De fato, a liberdade de vender cópias do software é crucial, pois é importante que existam coleções de software livre que possam ser vendidas em CD-ROM, e sua venda é uma fonte importante de fundos para o desenvolvimento do software livre. Portanto, um programa que não permita a sua inclusão em tais coleções não é um software livre.

No começo do projeto GNU, o próprio Stallman se sustentava vendendo software livre, já que ele tinha pedido demissão do MIT por não concordar com as licenças não-livres do mesmo: ele vendia uma fita DAT com o pacote GNU por US\$ 150,00. Atualmente o preço pode parecer absurdo, mas numa época em que as telecomunicações eram precárias, com certeza isso parecia um bom negócio. Claro que essas pessoas podiam fazer o que quisessem com essa fita, inclusive copiá-las: isso é parte do software livre.

Então, na prática, pode-se vender software livre. O erro de concepção de muitas pessoas foi provocado pela ambigüidade da palavra *free* em *free software*, o termo original do software livre. Em inglês, Stallman costuma dizer que *free software* é “*free as in speech, not as in beer*”. Em uma tradução grosseira, mas que ajuda a entender o objetivo de Stallman, podemos dizer que software livre é “*livre como em liberdade de expressão, não como em cerveja liberada.*”

Bem, então, o que posso fazer com um software livre?

Todo software livre obedece um critério conhecido como “as quatro liberdades”. Esse critério define quais são as coisas que você pode fazer com um software livre. Se um software obedece essas quatro liberdades, então ele pode ser considerado livre. Essas liberdades são:

1. A liberdade de usar o programa para qualquer fim que você deseje ou precise;
2. A liberdade de modificar o programa de forma que ele atenda a suas necessidades;
3. A liberdade de redistribuir cópias do programa, modificadas ou não, gratuitamente ou por um preço;
4. A liberdade de redistribuir as alterações feitas, de modo que a comunidade possa aproveitar suas alterações, e vice-versa;

Perceba que no caso do segundo e do quarto item, existe a obrigatoriedade do código fonte estar disponível sem nenhum tipo de ofuscação ou impedimento. Por isso que em geral o software livre é associado ao fato de ter-se o código fonte disponível.

1.2.2 O projeto inicia

Inicialmente, Stallman começou a desenvolver o seu projeto de *software livre* (foi nele que surgiu esse conceito e a expressão foi cunhada pelo próprio Stallman) sozinho, mas percebeu que não chegaria muito longe se permanecesse sozinho. Oras, ele acreditava que a comunidade *hacker* estava morrendo, mas sabia que ela não morreria sem luta. Portanto, ele decidiu então fazer uma espécie de “chamado às armas” da comunidade: no caso, utilizando-se da Usenet⁴, Stallman mandou uma mensagem em 27 de Setembro de 1983

⁴Usenet (do inglês *Unix User Network*) é um meio de comunicação onde usuários postam mensagens de texto (chamadas de “artigos”) em fóruns que são agrupados por assunto (chamados de *newsgroups*). Ao contrário das mensagens de e-mail, que são transmitidas quase que diretamente do remetente para o destinatário, os artigos postados nos newsgroups são retransmitidos através de uma extensa rede de servidores interligados. (WIKIPEDIA EM PORTUGUÊS, 2006)

para o grupo de discussões sobre Unix (`net.unix-wizards`), para mostrar seus objetivos:

“Iniciando nesta ação de graças eu vou escrever um sistema completo compatível com o Unix chamado GNU (Gnu Não é Unix), e fornecê-lo gratuitamente para todos que possam utilizá-lo. Contribuições de tempo, dinheiro, programas e equipamento são bastante necessárias.

Para começar, GNU será um kernel e todos os utilitários necessários para se escrever e executar programas em C: editor de textos, shell⁵, compilador, linkeditor, montador e algumas outras coisas. Depois disso nós adicionaremos um formatador de textos, YACC, um jogo do Império (Empire), uma planilha eletrônica, e centenas de outras coisas. Nós esperamos, eventualmente, fornecer tudo de útil que normalmente vem com um sistema Unix, além de quaisquer outras coisas úteis, incluindo documentação on-line e impressa.

*GNU será capaz de rodar programas do Unix, mas não será idêntico ao Unix. Nós faremos todos os aperfeiçoamentos que forem convenientes, baseados em nossa experiência com outros sistemas operacionais. Em particular, nós planejamos ter nomes de arquivos longos, números de versão de arquivos, um sistema de arquivos à prova de falhas, talvez auto-preenchimento de nomes de arquivos, suporte a vídeo independente de terminal, e eventualmente um sistema de janelas baseado no Lisp, de modo que vários programas Lisp e programas Unix comuns possam compartilhar uma tela. Tanto C quanto Lisp serão disponibilizados como linguagens de programação de sistemas. Nós teremos software de rede baseado no protocolo **chaosnet** do MIT, bastante superior ao UUCP. Nós também teremos algo compatível com o UUCP.”(STALLMAN, 1983)*

Com esse “chamado às armas”, Stallman estava convocando a ajuda da comunidade. Essa ajuda não precisava ser necessariamente com códigos: *“Eu estou pedindo aos fabricantes de computadores por doações de máquinas e dinheiro. Eu estou pedindo às pessoas por doações de programas e trabalho.*

⁵Interpretador de comandos

(...) *Programadores individuais podem contribuir escrevendo uma duplicata compatível de algum utilitário do Unix e doando para mim.*”(STALLMAN, 1983)

Para que não ficasse aquela idéia de que Stallman queria apenas aproveitar-se de mão-de-obra barata, ele criou uma licença que incorpora os princípios do software livre e que sofreu muito poucas revisões nos últimos tempos, a GPL (*General Public License* – Licença Pública Geral), que viria a tornar-se popular em projetos de software livre. Essa licença garante que:

1. O código criado pela pessoa não poderá ser fechado;
2. O código adicionado de/para o software se tornará livre, sendo esse aspecto conhecido como o aspecto “viral” da GPL;

Para contornar o aspecto “viral” da GPL, viria a ser criada a LGPL (*Lesser General Public License* – Licença Pública Menos Geral), que permite que bibliotecas livres sejam usadas em software proprietário, sem “contaminar” o software proprietário, *enquanto não forem feitas alterações **na biblioteca***.

O projeto GNU foi substituindo, um a um, os componentes proprietários do Unix, melhorando-os a ponto de acabarem se tornando os padrões *de facto* do mercado. Muitas instalações de Unix, assim que implementadas, tinham suas ferramentas proprietárias substituídas pelas ferramentas GNU, como o **bash** (*Bourne Again Shell*), **emacs** (*Editing Macros* — Editor de Texto) e o GNU **make** (Ferramenta para automação de compilação de sistemas), tornando-as muito populares, ao ponto de serem portadas para outras plataformas fora do mundo Unix.

Mas ainda assim, Stallman não tinha alcançado seu objetivo. Ele estava trabalhando em um *kernel* derivado de um projeto acadêmico do CMU, o MACH, chamado HURD. Esse *kernel* viria a ser considerado estável apenas em 2005. Nesse meio tempo, apareceu uma outra figura que conseguiria criar, de certa forma, para o GNU o que eles não tinham, o *kernel*.

1.3 Linus Torvalds e o Kernel Linux

Em 1991, Linus Benedict Torvalds era apenas mais um graduando em Ciências da Computação na Universidade de Helsinque, na Finlândia. Nessa época ele tinha um problema: ele queria fazer seus trabalhos acadêmicos em um ambiente Unix, mas os servidores da faculdade viviam com a agenda cheia. Então, ele pensou na possibilidade de recorrer a um Unix para o recém-lançado PC 386. O grande problema é que *nenhum* Unix para os PC 386, como o Xenix ou o Minix, eram satisfatórios. Além disso, em casos como o do Xenix, o *software* era proprietário e *excessivamente* caro, sendo que Linus não tinha dinheiro para comprar esse software.

Linus sabia que existia o Minix, criado pelo professor Andrew Tanenbaum, um renomado especialista em sistemas Unix e redes da Universidade Vrije, na Holanda. Era a melhor opção gratuita que ele conhecia, mas não era livre na época: embora tivesse o código aberto, seu uso era restrito às atividades acadêmicas. Porém, na prática o que Linus mais queria era um bom SO para seu recém comprado PC 386. As opções da época eram o DOS+Windows e o OS/2, ambos proprietários e caros e, no caso do DOS+Windows, muito fraco para suas necessidades. Com o Minix à mão e um certo talento de programação, Linus decidiu desenvolver ele próprio um SO.

Linus não era estúpido e conhecia as ferramentas do projeto GNU. Armado com elas e com o código fonte do Minix, começou a escrever seu próprio *kernel*. Originalmente iria chamar-se Freax (Free + Unix⁶). Mas Freax lembrava *Freak* (maluco, em inglês). Um amigo sugeriu então que ele chamasse seu projeto de Linux (Linus' Unix). E o nome acabaria pegando.

Rapidamente, Linus percebeu que não chegaria muito longe sozinho. Então, da mesma forma que Stallman, Linus acabou recorrendo a um “chamado às armas”. Nesse caso, dirigido à comunidade de usuários de Minix na Usenet (`comp.os.minix`):

*“Você sente falta dos dias do Minix/1.1 quando homens eram
homens e escreviam seus próprios drivers? Você está sem nenhum*

⁶quase todos os sistemas Unix e baseados em Unix têm seus nomes terminados em *X* de modo a indicar sua herança do Unix original

projeto legal e está ansioso para mexer num sistema operacional que você possa modificar para atender às suas necessidades? Você está achando chato quando tudo funciona no minix? Não ter mais de ficar mais a noite inteira tentando arrumar um programa legal? Então esta mensagem pode ser para você.

Como eu disse há um mês (?) atrás, eu estou trabalhando numa versão grátis de um similar para o Minix, para computadores AT-386. Ela finalmente atingiu o estágio onde já é usável (apesar de talvez não ser, dependendo do que você quer), e eu estou a fim de colocar (online) o código fonte para uma distribuição melhor. É apenas a versão 0.02 (com mais um patch) mas eu já rodei `bash/gcc/gnu-make/gnu-sed/ compress` dentro dela.

(...)

O sistema precisa de um monitor EGA/VGA e um disco rígido compatível (IDE serve). Se você ainda está interessado, pegue no FTP o `readme/relnotes` e/ou me mande um e-mail para saber mais.

(...)

Eu também estou interessado em alguém que tenha escrito alguns dos utilitários/ bibliotecas para o Minix. Se o seu trabalho pode ser distribuído publicamente (registrado ou mesmo domínio público), eu gostaria de ouvir comentários de vocês, e para que eu possa adicioná-los ao sistema.(Retirado de (ALECRIM, 2003))

Como pode ver, da mesma forma que no caso de Stallman, Linus pediu a ajuda da comunidade, e ela veio: em pouco tempo, o Linux tornou-se um *kernel* muito poderoso e rápido, superando as expectativas e os receios da comunidade na questão do seu desenvolvimento (o Linux é um *kernel* monolítico, aonde todo o sistema fica no espaço do *kernel*, diferentemente do HURD, que é baseado em *microkernel*, aonde a maior parte dos recursos fica no espaço do usuário, o que tornaria o Linux, ao menos na teoria, inferior ao HURD⁷), tornando-se assim um projeto considerado *sexy*, ou seja, realmente

⁷Alguns dizem que a estratégia de Kernel Monolítico do Linux é uma das garantias do seu sucesso, pois a maioria dos sistemas baseados em *microkernel* acabaram tendo, cedo ou tarde, problemas de estabilidade decorrentes de sua metodologia de colocar sistemas do Kernel no espaço do usuário (basta travar uma aplicação no espaço do usuário que existe alguma chance dela “acertar” partes do *microkernel* no espaço do usuário e comprometer totalmente a estabilidade)

interessante, para a comunidade.

Com o tempo, o GNU/Linux cresce rapidamente:

- 1993 — Patrick Volkerding cria a distribuição Slackware, a partir dos pacotes de instalação SLS, criados em 1992 por Peter McDonald;
- 1994 — Linux 1.0: já é capaz de rodar o X-Windows (sistema de interface gráfica do Unix) e a maioria dos servidores importantes para Internet (Apache, BIND, Sendmail, ...). Nesse mesmo ano, surge a Red Hat, a mais importante distribuição GNU/Linux de todos os tempos;
- 1996 — Linux 2.0: o Linux conta com uma comunidade de mais de 2 milhões de usuários;
- 1998 — Surge a Mandrake e a Conectiva, distribuições que no futuro viriam a se fundir e gerar o Mandriva Linux. Nesse ano também é cunhada a expressão “Código Aberto” (“*Open Source*”), para meio que “focar” o movimento software livre nos detalhes técnicos.
- 2000 — A China desenvolve sua própria distribuição GNU/Linux, a Red Flag, confiando na disponibilidade de código do Software Livre;
- 2001 — Linux 2.4: foco cada vez maior na confiabilidade e escalabilidade do sistema. Entre as novidades está o sistema de *firewall* IPTables;
- 2002 — Marcelo Tosatti passa a ser o mantenedor chefe da série 2.4, substituindo Alan Cox como o “número 2” de Linus Torvalds. Ele deixaria o “cargo” apenas em 2003, com o surgimento da série 2.6 do Linux, responsabilidade de Andrew Morton;
- 2005 — GNU/Linux completa 15 anos de existência sendo uma alternativa viável ao software proprietário;

1.4 As distribuições

Uma das coisas que ajudaram à popularizar o GNU/Linux foi o surgimento das *distribuições Linux*. Uma distribuição é basicamente uma versão em-

pacotada do GNU/Linux, com instaladores que facilitam o processo para a maioria das pessoas, além de suporte técnico e uma grande quantidade de programas e documentação agregada. Cada distribuição também possui um enfoque, que poderíamos dividir em: técnico, usuário final e *power user* (desenvolvedores de sistemas e usuários avançados).

Embora a grande quantidade de distribuições (mais de 300!(LINUX WEEKLY JOURNAL, 2006)) pareçam tornar o GNU/Linux uma grande “salada”, na realidade isso contribui para sua popularidade, uma vez que cada distribuição é voltada para um público diferente e para necessidades diversificadas, o que quer dizer é que, em geral, sempre haverá um GNU/Linux com a cara do usuário, conforme seu nível de conhecimento e familiaridade com o GNU/Linux e suas necessidades.

Além disso, em geral não existem grandes diferenças de distribuição (ou *distro* para resumir): por volta de 75 a 95% de tudo que você aprende em uma distro funcionará em outra. O mesmo vale para programas: é *extremamente* raro o caso de programas GNU/Linux que sejam específicos para uma determinada distro.

Vejamos então algumas das principais distros atualmente:

1.5 Algumas distribuições

1.5.1 Debian GNU/Linux

- **Perfil:** Técnico

Atualmente é a maior distro mantida por voluntários, tem como principal característica a segurança e o fato de procurar ser uma distro 100% software livre (ela possui um CD com software proprietário, mas não instala nenhum de maneira automática). Também tornou-se conhecida pela grande quantidade de software disponibilizado com ela (a versão mais atual, Debian 3.1r0 “Sarge”, completa, exige 14 CDs ou 2 DVDs para conter todos os seus pacotes) e pelas suas ferramentas de instalação, como o `dpkg` e o sistema `apt`.

1.5.2 Red Hat

- **Perfil:** Usuário Final

A maior distro comercial de todos os tempos, a Red Hat recentemente deixou de divulgar CDs de instalação via Internet de seu sistema RHEL (*Red Hat Enterprise Linux*), mas compensou a comunidade através do projeto *Fedora*, que funciona como uma versão pessoal do RHEL. Sua popularidade deve-se principalmente ao seu poderoso sistema de empacotamento, o RPM (*Red Hat Package Manager* — Gerenciador de Pacotes da Red Hat). É considerada a distro de referência de 9 entre 10 distribuições. Utiliza como sistema de atualização remota o **smart**, que atua em conjunto com o RPM.

1.5.3 Mandriva

- **Perfil:** *Power user*

A fusão entre as distribuições Mandrake (Francesa), Lycoris (Norte-americana) e Conectiva (Brasileira), a Mandriva é uma distro voltada ao usuário corporativo e ao usuário “*Power User*”. É baseada em RPM, como a Red Hat (já que todas as distros originais eram basadas no RPM), e possui uma grande versatilidade que é interessante ao usuário corporativo.

1.5.4 Slackware

- **Perfil:** técnico

A primeira grande distro de todos os tempos, Slackware é, assim como Debian, uma distro baseada em uma comunidade. O Slackware possui um sistema de pacotes próprio (o **tgz**) e é considerada complexa, pela falta de ferramentas visuais de configuração. Porém, seus usuários afirmam que, uma vez que você aprenda a mexer no Slackware, você consegue mexer com qualquer outra distro Linux e com quase todos os ambiente Unix que você precisar. É conhecida por ser enxuta, extremamente rápida e muito segura, sendo uma ótima opção para servidores. Possui alguns sistemas de atualização via Internet, como o **slapt-get** (um porte do **apt** do Debian para o Slackware) e o **slapget**.

1.5.5 Knoppix

- **Perfil:** Usuário Final

A Knoppix foi criada na Alemanha como uma “prova de conceito”, de que era possível criar uma distro GNU/Linux capaz de rodar a partir de um CD, baseada em Debian, originando o conceito de *Live CD*. Atualmente, em sua versão 5.0, ela roda a partir de um DVD e inclui quase tudo que pode-se desejar de um GNU/Linux. Tornou-se ponto de partida para uma grande quantidade de distros, principalmente distros com funções específicas, como a FIRE (usada para Informática Forense, ou seja, análise de discos que sofreram invasão ou foram corrompidos, acidental ou propositalmente, e precisam ser investigados juridicamente) ou a SystemRescueCD (usada para recuperação de dados e manutenção de sistemas).

1.5.6 Kurumin

- **Perfil:** Usuário Final

Criada pelo brasileiro Carlos Morimoto como uma “remasterização” (adaptação) do Knoppix, o Kurumin tornou-se rapidamente uma das distros mais importantes no Brasil, principalmente pela sua versatilidade e facilidade de instalação (o Kurumin é instalado como LiveCD, e pega todas as configurações feitas durante o uso como LiveCD). Gerou algumas distros alternativas, como Kalango e Kokar.

1.5.7 Ubuntu

- **Perfil:** Usuário Final

Criada pela Canonical Inc., uma empresa sul-africana, a Ubuntu é uma distro de extrema simplicidade, originalmente baseada em Debian e Knoppix (sua versão mais atual, porém, desviou-se para seguir seu próprio caminho). Vem em duas versões: uma como LiveCD e uma de instalação no disco rígido. A idéia do Ubuntu é “*Linux for Human Beings*” (Linux para as pessoas

comuns), e é muitíssimo simples, adotando o melhor da Debian e criando uma distribuição que uma pessoa comum pode usar sem problemas. Parte de sua filosofia vem de seu nome: como descrito na sua *homepage*(UBUNTU LINUX, 2006) *ubuntu* é uma palavra antiga africana que quer dizer “*Humanidade para os outros*” ou “*Sou o que sou pois todos somos assim.*”.

1.5.8 SuSE

- **Perfil:** *Power User*

Recentemente adquirida pela Novell, a SuSE é uma distro conhecida pelo seu caráter corporativo, incluindo aí facilidades para instalação em massa, configuração em massa, rotinas de atualização centralizadas e suporte técnico de alto calibre. Baseada em RPM, a SuSE seguiu o mesmo caminho que a Red Hat, “fechando” sua distribuição corporativa e criando um projeto paralelo, o OpenSuSE. Utiliza o *yum* (*Yellow Dog Updater, Modified* — Atualizador do Yellow Dog, Modificado), pego do Yellow Dog Linux, uma distro para Macintosh.

1.5.9 Arch Linux

- **Perfil:** Técnico

Também baseada em uma comunidade, a Arch é uma distro baseada no conceito de simplicidade e tradição. Utiliza um sistema chamado *pacman* para instalar seus pacotes e não é das mais simples, embora tenha uma comunidade cativa quase tão grande e empolgada quanto as de suas “primas” Debian e Slackware.

1.5.10 Gentoo Linux

- **Perfil:** Técnico

Gentoo parte de um princípio completamente diferente quanto a instalação de uma distribuição: ao invés de instalar binários pré-compilados, ele instala

algumas ferramentas básicas, como compiladores, *kernel*, ferramentas GNU básicas (*binutils*) e o seu gerenciador de pacotes, o **emerge**, e depois baixa da Internet o código-fonte dos programas a serem instalados e os compila especificamente para a máquina em questão. Como dito, utiliza o **emerge**, que baixa os códigos-fonte da Internet e os compila. Também aceita binários pré-compilados através do RPM.

Capítulo 2

Iniciando com o GNU/Linux

Agora que já temos uma boa compreensão do que veio antes de nós, podemos seguir a diante.

A idéia desse capítulo é que veremos comandos básicos que nos permitam usar o sistema GNU/Linux de maneira rápida, para podermos passar adiante. Na realidade, não é o objetivo nosso aqui tornar o usuário um mestre em GNU/Linux, pois não seria possível em um curso de Introdução lidar com todos os comandos e utilitários do GNU/Linux, além de haver farta documentação do GNU/Linux disponível na Internet.

De qualquer modo, nosso objetivo é oferecer a você o suficiente em comandos para que você consiga seguir adiante sem maiores problemas.

Então, seja bem vindo ao GNU/Linux. Você vai gostar, pode crer :D

2.1 Entrando no sistema

Ao inicializar o computador, o GNU/Linux apresenta algumas mensagens sobre informações do sistema, assim como o resultado a alguns comandos especiais iniciais: coisas como checagens de sistemas de arquivos, “*levantar*” (inicializar) servidores e *firewall*, realizar configurações de rede ... Neste momento, basta saber que o GNU/Linux realiza uma série de configurações básicas para que o sistema esteja pronto para rodar.

Uma vez que tais configurações e checagens tenham sido executadas, o sistema irá o entregar um *login*. Vamos primeiro entender porque ele faz isso, depois vamos ver como passar pelo *login* e entrar normalmente no sistema.

2.1.1 Filosofia do sistema

O GNU/Linux é um *Unix-Like*, ou seja, um sistema operacional *similar* ao Unix, embora não seja um Unix. Como todo sistema similar ao Unix, o GNU/Linux é multi-usuário e multi-tarefa, o que quer dizer que vários usuários podem estar usando o mesmo sistema ao mesmo tempo, e realizando várias operações ao mesmo tempo. Um determinado usuário pode estar ouvindo MP3 e editando um texto, enquanto outro pode estar compilando um programa e baixando arquivos da Internet, e um terceiro estar editando planilhas, lendo mensagens recebidas dentro da Internet e navegando via Web para obter informações necessárias ao seu documento.

Para que um usuário não interfira no ambiente do outro, garantir que usuários inexperientes não causem danos acidentais ao sistema, e para garantir que todos os usos do sistema possam ser controlados e auditados, o Unix adota um sistema de *usuários*. Sistemas Unix não podem ser acessados sem uma *conta de usuário*. Cada conta de usuário é protegida por uma *senha*. Algumas contas para usuários ocasionais podem ser criadas (por exemplo, é comum em instalações Unix a existência de uma conta **guest** – *convidado*) com senhas publicamente conhecidas, mas o normal é que a senha de um usuário seja conhecida *apenas* por ele.

A cada usuário o sistema estabelece alguns *privilégios*, ou seja, o que ele pode ou não realizar dentro do sistema. O ambiente Unix possui um esquema de privilégios muito simples, mas muito poderoso quando bem utilizado. Esse sistema permite que usuários convencionais não tenham acessos a ferramentas de administração, mas possam tranquilamente trabalhar em um ambiente seguro e confiável. Boa parte da segurança do Unix vêm desse sistema, desenvolvido a mais de 40 anos e que continua sendo usado em muitas corporações.

Em geral, existem usuários mais privilegiados, os *administradores*, que podem manipular arquivos de configuração e ajustar parâmetros do sistema. Mas mesmos estes possuem operações que não podem ser realizadas. Para

essas situações, existe uma conta de usuário especial, a conta **root**.

2.1.1.1 O usuário root

O **root** é também chamado de conta de *super-usuário*. Esse usuário é o primeiro e mais importante a ser criado em uma instalação Unix em geral (e Linux em particular), pois ele tem acesso pleno a *todos* os programas, recursos e arquivos do sistema.

O **root** é a mais poderosa conta, pois ele pode reparticionar discos e partições, formatar discos, configurar arquivos de acesso básico ao sistema, realizar procedimentos de instalação de programas de baixo nível, entre outras funções.

2.1.1.2 Cuidados com o root

O **root** deve ser usado *o mínimo possível*, pois uma das características mais poderosas (e portanto perigosas) do **root** é que ele *ignora **totalmente** a proteção oferecida pelos privilégios do Unix*. Portanto, comandos aleatórios lançados como **root** possuem uma grande chance de causar problemas. Na prática, o ideal seria *nunca, **jamaís*** use o **root** como conta cotidiana, mesmo que você seja o administrador do sistema. Crie uma conta comum para uso cotidiano, e deixe o **root** para situações importantes. Veremos mais adiante como alternar para super-usuário enquanto logado como usuário normal. No caso, peça para uma pessoa criar uma conta de usuário comum ou logue-se como **root** de qualquer modo. Se fizer o último, tome *muito*, MUITO cuidado mesmo com os comandos que for realizar. E lembre-se: *você foi avisado!!*

2.1.2 Login

Agora voltemos ao GNU/Linux: atualmente, a maioria dos *logins* são feitos por meio de ferramentas como o GDM (*GNOME Desktop Manager*). Mas vamos trabalhar na Linha de Comando, de modo a mostrar os principais comandos do GNU/Linux. Além disso, pode ser que o seu ambiente, principalmente se for um servidor ou efetuar um *login* remoto, você receberá uma mensagem como a abaixo:

```
Mandriva Linux release 2006.0 (Official) for i586
Kernel 2.6.12-12mdk on a i686/tty1
hufflepuff login:
```

Trecho de Código 2.1.1: Exemplo de mensagem de *Login*

A primeira linha indica qual o ambiente em uso. No caso, é um GNU/Linux, distribuição Mandriva Linux, versão 2006.0, para computadores PC (i586).

A segunda é uma informação específica do sistema. No caso, o sistema possui um *kernel* Linux versão 2.6.12, com versão interna ‘12mdk’ e está rodando em um sistema i686 (AMD Sempron) e que esse é o 1º Terminal do sistema.

A terceira linha é o prompt do *login*. No caso, ele apresenta primeiro o nome da máquina (no caso, *hufflepuff*) e em seguida a mensagem *login:*, indicando que ele está pronto para receber o seu nome de usuário. Digite-o. Logo em seguida você receberá uma mensagem pedindo a sua senha:

```
Password:
```

Trecho de Código 2.1.2: *Prompt do password*

Digite sua senha. Diferentemente do nome de usuário, a sua senha *não* será ecoada (ou seja, mostrada na tela). Essa é uma das maneiras do GNU/Linux (e dos Unixes em geral) garantir uma boa segurança quanto ao acesso ao sistema: alguns especialistas consideram que o eco de asteriscos (muito comum em sites da Web) não é uma boa idéia, pois dá a um invasor em potencial, principalmente um que esteja vendo o usuário se logar, uma idéia do tamanho de sua senha e do esforço que ele terá para a quebrar.

Se você digitar erroneamente sua senha, você receberá uma mensagem de erro como a seguinte:

```
Login incorrect
```


Atenção: apesar de ser uma boa idéia digitar sua senha rapidamente, *tome cuidado*: a maioria dos sistemas Unix (GNU/Linux incluído) possui um limite de quantas tentativas (normalmente 3) um usuário pode fazer de se logar com senhas erradas antes da conta ser bloqueada por algum tempo (ou até um administrador ou o `root` a reativar). Isso foi feito para impedir (ou ao menos dificultar) ataques de força bruta¹.

Se você fez tudo corretamente, você irá receber o *prompt*² do sistema. Ele será similar ao seguinte.

```
Last login: Wed Apr 12 00:06:43 on tty1
[fe costa@hufflepuff ~]$
```

Trecho de Código 2.1.3: *Prompt* do *shell*

O *prompt* do *shell* poderá variar bastante, conforme as configurações do sistema em questão.

Você acaba de entrar o Linux, passando para o *shell*.

2.2 O *Shell*

O *shell*, ou Interpretador de Comandos, é o programa principal que o usuário irá manipular. Ele recebe os comandos do operador e executa as funções que o operador estipulou, retornando informações desejadas ou mensagens de erro sobre comandos mau sucedidos.

Na prática, o *shell* atua como um tradutor dos comandos do usuário para operações internas do sistema. Em geral, o *shell* oferece uma linguagem de *scripts*³ para a automação de tarefas do sistema. Essa linguagem pode ser simples ou poderosa, dependendo apenas dos desenvolvedores do *shell*

¹Um ataque de força bruta é um ataque ao sistema aonde o invasor, através de programas ou *scripts*, tenta logar em um sistema selecionando um nome de usuário aleatório (ou não) e tentando todas as combinações de senhas possíveis, até que o invasor adentre o sistema e/ou bloqueie a conta de usuário

²*Prompt* é uma espécie de indicador que o sistema está pronto para receber novos comandos

³programas simples interpretados, que automatizam funções do sistema

Existem muitos *shells* diferentes para o GNU/Linux. O principal é o **bash**, ou *Bourne Again Shell*, desenvolvido a partir do *Bourne Shell* (**sh**) original. Além dele, outros *shells* de importância são o **tcsch**, um *shell* baseado no *C Shell* (**csh**) original e o **ksh**, ou *Korn Shell*, um shell aparentado ao **sh** original e ao **bash**.

2.3 Obtendo ajuda de comandos

Parece uma bobagem, mas antes de seguirmos adiante, veremos um pouco sobre como obter ajuda dentro do GNU/Linux para seus comandos. Mas essa opção tem seu motivo: quase todos os comandos e utilitários do GNU/Linux possuem ajuda através dos sistemas de ajuda do GNU/Linux, o **man** e o **info**, além de boa parte das bibliotecas do sistema e arquivos de configuração. Saber usar convenientemente essa ajuda é uma “mão na roda” quando se está com dúvidas ou não se sabe usar um determinado comando ou utilitário. Desse modo, saber usá-las corretamente é quase tão importante quanto saber usar outros comandos. Vamos a elas então.

2.3.1 man

A principal ferramenta de ajuda do mundo Unix, incluindo o GNU/Linux, o **man** (redução de *manual*) exibe na tela uma página de informações sobre o programa conhecida no mundo Unix como *manpage*. Para acessar a *manpage* de um determinado comando, você utiliza o comando **man** <comando>. Por exemplo, se você quiser chamar a *manpage* do *shell* **bash**, digite:

```
[fecosta@hufflepuff ~]$ man bash
```

Trecho de Código 2.3.1: Exemplo do comando **man**

Que ele retornará uma janela com as informações sobre o comando, como a apresentada no Código 2.3.2, na Página 23.

Uma coisa importante a se notar no exemplo do Trecho de Código 2.3.2 é na seção *See also*, aonde ele mostra algumas outras *manpages* que o usuário

```

BASH(1)                                                    BASH(1)

NAME
    bash - GNU Bourne-Again SHell

SYNOPSIS
    bash [options] [file]

COPYRIGHT
    Bash is Copyright (C) 1989-2004 by the Free Software Foundation, Inc.

DESCRIPTION
    Bash is an sh-compatible command language interpreter that executes
    commands read from the standard input or from a file. Bash also incor-
    porates useful features from the Korn and C shells (ksh and csh).

+ldots

SEE ALSO
    Bash Reference Manual, Brian Fox and Chet Ramey
    The Gnu Readline Library, Brian Fox and Chet Ramey
    The Gnu History Library, Brian Fox and Chet Ramey
    Portable Operating System Interface (POSIX) Part 2: Shell and Utili-
    ties, IEEE
    sh(1), ksh(1), csh(1)
    emacs(1), vi(1)
    readline(3)

+ldots

    Array variables may not (yet) be exported.

GNU Bash-3.0                2004 June 26                BASH(1)

```

Trecho de Código 2.3.2: Resultado de `man bash`

pode consultar. Perceba que o formato apresentado é `emacs(1)`. Isso é importante, pois todas as *manpages* são divididas em grupos, ou *seções*, aonde cada seção é sobre um determinado assunto. Isso ajuda principalmente em casos aonde um comando ou utilitário tem o mesmo nome, por exemplo, de uma chamada de sistema (*system call*⁴), de modo que você pode usar esse número que vêm após o comando (chamado de número de seção) para acessar a informação desejada. No caso, você utilizará o comando `man <seção> <comando>`. Por exemplo, se você quiser acessar a *manpage* do `ksh(1)` (*Korn*

⁴*system call* é uma rotina interna do *kernel* que o usuário só pode acessar através de (a) um utilitário, como um *shell* ou; (b) programação direta

SEÇÃO	DOCUMENTAÇÕES
0	Todas as Seções
1	Comandos
2	Chamadas do Sistema
3	Funções de Bibliotecas C
4	Arquivos de Configuração ou Especiais
5	Formatos e Conversões de arquivo
6	Jogos (apenas Linux)
7	Convenções e Pacotes de Macro
8	Pacotes de Gerenciamento
9	Rotinas do Kernel

Tabela 2.1: Seções de *manpages*

Shell), digite o comando indicado no Trecho de Código 2.3.3, na página 24.

```
[fecosta@hufflepuff ~]$ man 1 ksh
```

Trecho de Código 2.3.3: Exemplo do comando **man** com número de seção

No caso, as seções padronizadas pelo POSIX⁵ são como mostradas na Tabela 2.1, na página 24.

2.3.1.1 Navegando pela *manpage*

O **man** utiliza o editor **vi** como visualizador de arquivos ao exibir as *manpages*. No caso, vamos ensinar apenas alguns truques básicos.

Use as setas direcionais do teclado para mover-se pelo sistema. Pode-se usar também as teclas **H**, **J**, **K** e **L** para navegar pelo texto, respectivamente, para a esquerda, para baixo, para cima e para a direita. Para sair da *manpage*, aperte **Esc**, e em seguida use **:q**.

Se quiser procurar alguma informação dentro da *manpage*, aperte **Esc** e em seguida digite **/** e o texto a ser localizado.

⁵*Portable Operating System Interface*, uma Interface padronizada adotada por todos os sistemas Unix

Isso é o básico do básico, mas deve dar informações suficientes para que você possa consultar as *manpages* sempre que precisar.

2.3.2 info

Algumas aplicações, porém, não possuem *manpages*, ou possuem uma documentação tão complexa que seria impossível usar apenas uma *manpage* para realizar toda a documentação do sistema. Para isso, o projeto GNU criou um novo sistema de documentação, o Texinfo. Seu navegador é o **info**.

Para navegar em uma página **info** de um programa, basta digitar **info <comando>**. Em alguns casos, o comando estará incluído em um pacote de documentação. Nesse caso, utilize o comando **info <pacote> <comando>**. Um exemplo disso é o comando **cat**, para mostrar e concatenar (ajuntar) arquivos, que faz parte do pacote **coreutils**. Portanto, para chamar a ajuda do **cat**, digite o comando do Trecho de Código 2.3.4, na Página 25. Você receberá uma janela similar à do Trecho de Código 2.3.5, na Página 26.

Você também pode pesquisar por um determinado assunto dentro das *infopages* usando o comando **info -apropos=<assunto>**. Por exemplo, para pesquisar-se nas *infopages* por **cat**, digite o comando do Trecho de Código 2.3.6, Página 26.

```
[fecosta@hufflepuff ~]$ info coreutils cat
```

Trecho de Código 2.3.4: Exemplo do comando **info**

2.3.3 Navegando na *infopage*

O GNU **info** é baseado em um *subset* de comandos do editor de texto GNU **emacs**, portanto seus comandos de navegação são diferentes dos do **man**, baseado no **vi**. Se desejar, pode forçar a navegação no **info** a ser igual a do **man** com o uso da opção **-vi-keys**.

O deslocamento dentro de uma página **info** é feito com as setas direcionais do teclado, ou com as combinações de tecla **Ctrl** + **n**, **Ctrl** + **p**,

```
File: coreutils.info, Node: cat invocation, Next: tac invocation, Up: Output\
of entire files

3.1 'cat': Concatenate and write files
=====

'cat' copies each FILE ('-' means standard input), or standard input if
none are given, to standard output. Synopsis:

    cat [OPTION] [FILE]...

The program accepts the following options. Also see *Note Common
options::.

'-A'
'--show-all'
    Equivalent to '-vET'.

'-B'
'--binary'
    On MS-DOS and MS-Windows only, read and write the files in binary
    mode. By default, 'cat' on MS-DOS/MS-Windows uses binary mode
--zz-Info: (coreutils.info.bz2)cat invocation, 84 lines --Top-----
Welcome to Info version 4.8. Type ? for help, m for menu item.
```

Trecho de Código 2.3.5: Resultado do comando `info coreutils cat`

```
[fecosta@hufflepuff ~]$ info --apropos=cat
```

Trecho de Código 2.3.6: Exemplo do comando `info -apropos`

Ctrl + **b** e **Ctrl** + **f**, para ir-se para a baixo, para cima, para a direita e para a esquerda, respectivamente. Pode-se usar ainda **Home** e **End** para ir ao início e para o fim de uma linha, ou então **Ctrl** + **a** e **Ctrl** + **e**. Para ir palavra por palavra, use **Alt** + **f** e **Alt** + **b**⁶. Use **Page** para rolar o texto da ajuda uma janela para baixo e **Del** para voltar o texto uma janela. Essas funções podem ser feitas também com as teclas **Page** e **Page** respectivamente. Para sair do **info**, digite **Ctrl** + **x** e depois **Ctrl** + **c**.

Todas as *infopages* são divididas em *nós*. Principalmente na ajuda de programas grandes, como o **emacs** é importante atentar para esse detalhe, na medida em que você poderá precisar navegar de nó em nó, utilize as teclas **n** ou **p**. Use **t** para ir ao nó raiz do nó atual (ou seja, para a lista de tópicos da seção atual) e keystroked para voltar para a lista de *infopages* disponível. Use **<** para ir ao primeiro nó do pacote selecionado e **>** para ir ao último nó. Se quiser saltar diretamente para um determinado nó, digite **g** e digite o nome do nó em questão. Para saltar diretamente a um determinado nó, digite o número do nó. **=>** leva a navegação entre as referências a nós dentro do nó atual. Pressionando **Enter**, você vai até o nó que está sendo referenciado no momento.

Para pesquisar alguma informação dentro do nó atual, use **s** ou **S**, se desejar que a pesquisa seja *case-sensitive* (diferencie maiúsculas de minúsculas). Para continuar procurando o mesmo item, utilize **Ctrl** + **n**, ou **Ctrl** + **N** para ir para itens anteriores.

Você também pode usar a linha que fica abaixo da janela que mostra o nó para inserir o comandos manualmente. Para ir até ela, digite **Alt** + **x** (**Meta** + **x**). Digite o comando desejado e pressione **Enter**. Um exemplo de comando é **print-node**, que imprime o conteúdo do nó atual. Caso queira abortar um comando ou operação que esteja nessa linha (o *minibuffer*), digite **Ctrl** + **G**.

Esse é o básico de **info** que você precisa saber para pesquisar informações sobre comandos do GNU/Linux. Para mais informações, recomendamos que consulte o manual do **info**, digitando **info info**, ou então indo até (PRO-

⁶na verdade, os atalhos são **Meta** + **f** e **Meta** + **b**, mas na maioria dos sistemas GNU/Linux essa tecla especial, que existe em algumas *workstations* Unix, foi mapeada para a tecla **Alt**.

JETO GNU, 2006a).

2.4 Alguns comandos básicos

Essa seção tem como objetivo apresentar uma série de comandos básicos para a manipulação, acesso e visualização de arquivos no sistema. No caso, utilizaremos como referência o *shell* padrão do GNU/Linux, o **bash**. No caso de outros *shells*, o melhor a ser feito é consultar suas *manpages* ou páginas **info**.

Atenção: No GNU/Linux, comandos, nomes de arquivos e opções são *case-sensitive*, ou seja, fazem diferenciação de maiúsculas e minúsculas. Portanto, **man** é diferente de **Man** e de **MAN**, por exemplo.

2.4.1 Listando arquivos: **ls**

De início, você sempre precisará saber aonde seus arquivos estão. Os Unix em geral, incluindo aqui o GNU/Linux, incluem o comando **ls**, que lista o conteúdo de um diretório para o usuário, exibindo na tela os arquivos disponíveis dentro do diretório.

Atenção: No GNU/Linux, todos os diretórios são separados por **/**, e o diretório mais importante é o **/** (*raiz*). Portanto, utilize a **/** para separar um diretório do outro, e não a ****, como seria normal de imaginar-se⁷.

O comando básico é simplesmente **ls**, que irá lhe retornar uma lista contendo os arquivos que estão dentro do diretório atual, de maneira similar à mostrada no Trecho de Código 2.4.1, na Página 29. Você também pode utilizar **ls <diretório>** para visualizar os arquivos contidos em outro diretório. Mas o **ls** possui uma série de opções bastante úteis quando precisamos de mais informações.

⁷baseando-se na experiência de uso no Windows


```
[fecosta@hufflepuff ~]$ ls
aif-mount/      #.emacs#      InstallShield/  QLOG
comanche3-0b4/  Fotos/         KDE/             street2.sta
Desktop/        GNUstep/       mac-3.99-u4-b4/  tmp/
Documentos/     GUI.cs         mac-3.99-u4-b4.tar.gz*  Vídeo/
Download/       HelloWorld.cs  Música/          vpd.properties
drakx/          HelloWorld.exe* nohup.out        x.log
```

Trecho de Código 2.4.1: Exemplo do comando `ls`

Atenção: Todos os comandos do GNU/Linux, e no Unix em geral, permitem que opções sejam combinadas. Por exemplo, o comando `foo` pode ter as opções `-x`, `-y` e `-z`. Nesse caso, caso se você quiser usar as opções `-x` e `-y` você pode usar tanto `foo -x -y` quanto `foo -xy`.

A primeira opção seria a `-R`(*recursive*), que lista o conteúdo do diretório apontado e de *todos* os subdiretórios abaixo dele. Isso ajuda quando você precisa saber, por exemplo, se uma página Web já está com todos os arquivos dentro dele.

Outra opção útil é a opção `-a`(*all files*, todos os arquivos), que mostra os arquivos *ocultos* (também chamados de terminologia Unix de *desinteressantes*) do sistema. Um arquivo em Unix é considerado oculto quando ele começa com um `'.'`(ponto). Em geral, usa-se isso para ocultar arquivos/diretórios de configuração.

A opção `-l`(*one file per line*, um arquivo por linha), mostra um arquivo por linha apresentada. Essa opção é útil quando se usa um *shell script*⁸ para automatizar-se alguma função e exige-se uma listagem dos arquivos a serem manipulados. O comportamento normal do `ls` é exibir o máximo de comandos por linha, formando colunas de arquivos.

Uma opção presente em algumas versões do `ls`⁹ é a opção `-color`, que gera uma listagem *colorida* dos arquivos, sendo que essa cor muda conforme o tipo de arquivo (executável, oculto, diretório, arquivo comum, ...).

⁸“programa” que é interpretado pelo *shell*

⁹na verdade, do *shell* que o implementa

A opção `-h` (*human readable*, legível pelo ser humano) mostra os tamanhos de arquivo (se visível) em um formato mais compreensível pelas pessoas. Em geral, o `ls` (e demais comandos que envolvam tamanhos em disco) representa os tamanhos dos arquivos em número de *inodes*¹⁰. Embora um *inode* em geral ocupe 512 bytes, isso é muito dependente de implementação, sistema de arquivos adotado e formatação adotada, portanto essa representação nem sempre é a melhor disponível. O `-h` obriga o `ls` a determinar o tamanho aproximado do arquivo ao listá-lo. No Trecho de Código 2.4.2, na Página 31, mostramos um exemplo aonde o tamanho dos arquivos é apresentado de maneira mais legível.

Mas talvez a opção mais importante que o `ls` possui é a `-l` (*Long description*, descrição Longa), que apresenta todas as informações relacionadas ao arquivo. Nesse caso, cada “coluna” representa uma informação diferente:

1. Informações de arquivo, incluindo permissões de arquivos (veremos mais sobre isso logo a seguir);
2. Arquivos ou diretórios dentro desse diretório. O valor mínimo é 2, graças aos diretórios especiais `.` e `...`. Para arquivos em geral, esse valor é 1.
3. Dono do arquivo (usuário que o criou ou ao qual foi passada a posse do arquivo — veremos mais sobre isso adiante, na Seção 2.7.3, na página 51);
4. Grupo do arquivo (grupo que pode manipular esse arquivo com maiores permissões — veremos mais sobre isso adiante, na Seção 2.7.4, na página 53);
5. Tamanho (descrito em *inodes*, a não ser que as opções `-k` ou `-h` estejam ativas);
6. Data da última alteração do arquivo;
7. Nome do arquivo;

No Trecho de Código 2.4.2, na Página 31, mostramos um exemplo aonde o tamanho dos arquivos é apresentado de maneira mais legível.

¹⁰i-nodos, ou nós dentro da estrutura do sistema de arquivos

```
[fecosta@hufflepuff CursoGNUlinux]$ ls -lh
total 1,6M
-rwxrwxrwx 1 root root 865 Abr 13 11:46 cap10.aux*
-rwxrwxrwx 1 root root 92 Abr 10 16:41 cap10.tex*
-rwxrwxrwx 1 root root 1,8K Abr 13 11:46 cap11.aux*
-rwxrwxrwx 1 root root 449 Abr 10 16:46 cap11.tex*
-rwxrwxrwx 1 root root 448 Abr 10 16:43 cap11.tex~*
-rwxrwxrwx 1 root root 855 Abr 13 11:46 cap12.aux*
-rwxrwxrwx 1 root root 88 Abr 10 16:44 cap12.tex*
-rwxrwxrwx 1 root root 2,4K Abr 13 11:46 cap1.aux*
-rwxrwxrwx 1 root root 0 Abr 11 00:42 cap1.bbl*
-rwxrwxrwx 1 root root 866 Abr 11 00:42 cap1.blg*
-rwxrwxrwx 1 root root 24K Abr 10 15:54 cap1.log*
-rwxrwxrwx 1 root root 31K Abr 11 17:41 cap1.tex*
-rwxrwxrwx 1 root root 31K Abr 11 16:50 cap1.tex~*
-rwxrwxrwx 1 root root 5,8K Abr 13 11:46 cap2.aux*
-rwxrwxrwx 1 root root 262K Abr 12 17:00 cap2.log*
-rwxrwxrwx 1 root root 34K Abr 13 11:45 cap2.tex*
-rwxrwxrwx 1 root root 32K Abr 13 11:18 cap2.tex~*
-rwxrwxrwx 1 root root 2,1K Abr 13 11:46 cap3.aux*
-rwxrwxrwx 1 root root 597 Abr 10 16:02 cap3.tex*
-rwxrwxrwx 1 root root 1,1K Abr 13 11:46 cap4.aux*
-rwxrwxrwx 1 root root 243 Abr 10 16:13 cap4.tex*
-rwxrwxrwx 1 root root 2,0K Abr 13 11:46 cap5.aux*
-rwxrwxrwx 1 root root 639 Abr 10 16:16 cap5.tex*
-rwxrwxrwx 1 root root 1,6K Abr 13 11:46 cap6.aux*
-rwxrwxrwx 1 root root 380 Abr 10 16:18 cap6.tex*
-rwxrwxrwx 1 root root 993 Abr 13 11:46 cap7.aux*
-rwxrwxrwx 1 root root 176 Abr 10 16:38 cap7.tex*
-rwxrwxrwx 1 root root 1,6K Abr 13 11:46 cap8.aux*
-rwxrwxrwx 1 root root 341 Abr 10 16:39 cap8.tex*
-rwxrwxrwx 1 root root 1,9K Abr 13 11:46 cap9.aux*
-rwxrwxrwx 1 root root 492 Abr 10 16:47 cap9.tex*
-rwxrwxrwx 1 root root 492 Abr 10 16:46 cap9.tex~*
-rwxrwxrwx 1 root root 531 Abr 13 11:46 configuracao.aux*
-rwxrwxrwx 1 root root 1,5K Abr 12 14:09 configuracao.tex*
-rwxrwxrwx 1 root root 1,5K Abr 12 00:46 configuracao.tex~*
-rwxrwxrwx 1 root root 788 Abr 13 11:46 curso.aux*
-rwxrwxrwx 1 root root 957 Abr 13 11:37 curso.bbl*
-rwxrwxrwx 1 root root 1,7K Abr 11 16:41 curso.bib*
-rwxrwxrwx 1 root root 961 Abr 11 13:15 curso.bib~*
-rwxrwxrwx 1 root root 1,3K Abr 13 11:37 curso.blg*
-rwxrwxrwx 1 root root 144K Abr 13 11:46 curso.dvi*
-rwxrwxrwx 1 root root 12K Abr 13 11:46 curso.log*
-rwxrwxrwx 1 root root 898 Abr 13 11:46 curso.lop*
-rwxrwxrwx 1 root root 240K Abr 12 17:02 curso.pdf*
-rwxrwxrwx 1 root root 480K Abr 12 17:02 curso.ps*
-rwxrwxrwx 1 root root 1,7K Abr 12 15:18 curso.tex*
-rwxrwxrwx 1 root root 1,7K Abr 12 00:13 curso.tex~*
-rwxrwxrwx 1 root root 11K Abr 13 11:46 curso.toc*
-rwxrwxrwx 1 root root 0 Abr 13 11:47 lslh.txt*
-rwxrwxrwx 1 root root 164K Abr 12 00:58 playsh-0.1.tgz*
-rwxrwxrwx 1 root root 431 Abr 10 23:55 referencias.bib*
-rwxrwxrwx 1 root root 431 Abr 10 23:55 referencias.bib~*
-rwxrwxrwx 1 root root 646 Abr 13 11:42 x.log*
```

Trecho de Código 2.4.2: Exemplo do comando `ls -lh`

Antes de seguirmos adiante, vamos falar de um tópico que já falamos por alto tanto no *Login* quanto no caso do comando `ls`: as *permissões de arquivo*.

2.4.1.1 Permissões de arquivo

Como dissemos anteriormente, na Seção 2.1.1, página 18, quando discutimos a filosofia do GNU/Linux, uma das garantias de segurança de todos os sistemas *Unix-like* é o fato de haver um sistema de permissões de acesso aos arquivos e programas. Esse sistema é baseado em certas *permissões de arquivo* que são definidas pelo dono do arquivo, envolvendo:

- O dono do arquivo (*User*);
- O grupo ao qual o arquivo foi atribuído (*Group*);
- Os demais usuários do sistema (*Other*);

A idéia básica é que cada um desses grupos de usuário possui direito a:

- Ler o arquivo (*read*): aqui inclui-se a cópia do arquivo, o arquivo aparecer em uma lista fornecida pelo `ls` ou sua abertura por um programa qualquer;
- Escrever no arquivo (*write*): envolve também modificar seu nome, seu conteúdo, movimentá-lo para outro diretório ou o apagar. No caso de diretórios também envolve criar novos arquivos dentro do diretório (embora não necessariamente alterar os que estão dentro dele);
- Executar o arquivo (*execute*): Executar esse arquivo como um programa dentro do sistema (seja um binário ou um *script*). Para diretórios, essa permissão indica que o usuário pode acessar o diretório.

Como posso descobrir quais são as permissões que tenho de acessar um arquivo? O comando `ls -l` oferece informação suficiente para conseguirmos definir nossas permissões dentro do sistema.

A primeira coluna do `ls -l` é a coluna de “Informações de Arquivo”. Ela é formada por uma seqüência de 10 caracteres, podendo serem letras ou o

hífen(-). O primeiro caracter dessa seqüência indica o tipo de arquivo: os mais importantes aqui são o *arquivo comum*(-), o *diretório*(d) e a *ligação simbólica*(l). Existem outros tipos, mas se for o caso falaremos caso sejam necessários.

Os nove caracteres seguintes são divididos em três grupos de permissões de arquivo. Cada um desses demonstra as permissões para um dos três grupos que citamos anteriormente: no caso, na seqüência *Dono/Grupo/Outros*. Dentro de cada um desses grupos, os caracteres podem ser um caracter ou o hífen. No caso, os caracteres são sempre na seqüência **rw**x e indicam se o atributo em questão está ativado: no caso, na seqüência **L**eitura(*read* - **r**), **E**scrita(*write* - **w**) e **E**xecução(*execute* - **x**). Se um desses caracteres estiver faltando, substituído por um hífen, o atributo (ou permissão) correspondente está desativada.

Por exemplo, um arquivo `trab.doc` com permissões `-rw-r-r-` quer dizer:

- O arquivo é um *arquivo comum* (1º caracter -);
- Seu usuário pode ler o arquivo e gravar nele, mas não pode o chamar como um program (1º grupo de caracteres — 2º caracter ao 4º caracter — `rw-`);
- O grupo ao qual o arquivo foi designado pode o ler, mas não pode gravar nele ou o chamar como um programa (2º grupo de caracteres — 5º caracter ao 7º caracter — `r--`);
- Outros usuários do sistema podem o ler, mas não gravar ou o chamar como um programa (3º grupo de caracteres — 8º caracter ao 10º caracter — `r--`);

Enquanto um outro arquivo, um *script* de administração chamado `backup`, com permissões `-rwxr-x--` quer dizer:

- O arquivo é um *arquivo comum* (1º caracter -);
- Seu usuário pode ler o arquivo, gravar nele e o chamar como um programa (1º grupo de caracteres — 2º caracter ao 4º caracter — `rwx`);

- O grupo ao qual o arquivo foi designado pode o ler e o chamar como programa, mas não pode gravar nele (2º grupo de caracteres — 5º caracter ao 7º caracter — **r-x**);
- Outros usuários do sistema não podem o ler, gravar nele ou o chamar como um programa (3º grupo de caracteres — 8º caracter ao 10º caracter — **---**);

Basta combinar as informações embutido nesse código e você poderá descobrir suas permissões de acesso aos arquivos. No caso, é possível alterar-se tais permissões caso você seja o dono do arquivo (ou o **root**). Veremos isso na Seção 2.7, página 46, aonde trataremos melhor desse assunto com mais detalhe. Por enquanto, o que foi explicado é o suficiente para facilitar a compreensão dos comandos a seguir.

2.4.2 Manipulação de arquivos

Agora que sabemos como listar os arquivos do sistema, podemos passar a operações de manipulação de arquivos. No caso, falaremos principalmente de cópia, movimentação e exclusão de arquivos. Procure atentar ao que será falado a diante, pois o GNU/Linux, apesar de seu sistema de permissão de acesso, não costuma ser muito gentil em caso de “falha humana”. Todo cuidado é pouco quando se usa o GNU/Linux, principalmente como **root**.

Atenção: Todas as operações a seguir são sujeitas ao sistema de permissão de acesso, exceto se você estiver fazendo as operações como **root**.

2.4.2.1 cp

O comando **cp** realiza uma cópia de um arquivo de um diretório para outro, ou fazer cópias de um arquivo com outro nome (principalmente para efeito de *backup*). O uso básico desse comando é com a sintaxe **cp <arquivo-origem> <arquivo-destino>**. Se você for fazer a cópia do arquivo com o mesmo nome para outro diretório, use apenas o diretório no destino (não é necessário nesse caso dar um nome para o arquivo de destino). Você pode usar o *caminho*

absoluto (partindo do diretório raiz — /) ou o *caminho relativo* (partindo do diretório atual) do arquivo tanto na origem quanto no destino. Por exemplo:

- `cp /etc/hosts hostscopia` cria uma cópia de `/etc/hosts` no diretório atual, com o nome de `hostscopia`;
- `cp meuarq /etc/backup` cria uma cópia do arquivo `meuarq` no diretório `/etc/backup`, ou então em `/etc`, com o nome de `backup`, se o diretório `/etc/backup` não existir;
- `cp trabalhos/work.doc /home/felipe` copia o arquivo `work.doc` no diretório `trabalhos` abaixo do diretório atual no diretório `/home/felipe`;

Atenção: O `cp`, assim como todos os comandos de manipulação de arquivos e diretórios do GNU/Linux (e do Unix em geral) ***não alertam se o arquivo de destino existe!*** Muita cautela ao manipular arquivos: embora o sistema de permissão de acessos ofereça alguma proteção contra sobrescrita acidental, ele pode falhar, principalmente em arquivos aonde o usuário tenha permissão de acesso.

Para a maioria das pessoas, o comando `cp` simples é muito útil, mas existem várias opções úteis para ele:

Talvez a mais interessante para muitos seja a opção `-i` (*Interactive*, interativo), que, *caso o arquivo de destino já exista*, pede para o usuário confirmar a operação. Se essa opção, no seu sistema estiver ativa por um apelido ao `cp` (mais sobre isso na Seção 2.10, na página 64), utilize a opção `-f` (*Force*, forçar) para ignorar esse mecanismo.

A opção `-R` já foi comentada no `ls`, mas ela também funciona no `cp`. Esse comando é muito útil para cópias de *backup* de um ou mais diretórios.

Uma opção bastante interessante é `-v` (*verbose* — informativo). Essa opção faz com que o comando `cp` mostre tudo o que ele está fazendo. Um exemplo do comando `cp` com a opção `-v` ativa pode ser vista no Trecho de Código 2.4.3, na Página 36.

```
[fecosta@hufflepuff ApostilaGNU/Linux]$ cp -v * ~/Apostilas/
'apostila-linux-apendA.pdf' -> '/home/fecosta/Apostilas/apostila-linux-apendA.pdf'
'apostila-linux-cap10.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap10.pdf'
'apostila-linux-cap11.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap11.pdf'
'apostila-linux-cap1.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap1.pdf'
'apostila-linux-cap2.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap2.pdf'
'apostila-linux-cap3.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap3.pdf'
'apostila-linux-cap4.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap4.pdf'
'apostila-linux-cap6.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap6.pdf'
'apostila-linux-cap7.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap7.pdf'
'apostila-linux-cap8-part1.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap8-part1.pdf'
'apostila-linux-cap8-part2.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap8-part2.pdf'
'apostila-linux-cap8-part3.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap8-part3.pdf'
'apostila-linux-cap8.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap8.pdf'
'apostila-linux-cap9.pdf' -> '/home/fecosta/Apostilas/apostila-linux-cap9.pdf'
'apostilaLinuxJonathan.pdf' -> '/home/fecosta/Apostilas/apostilaLinuxJonathan.pdf'
'Cap47-ControleDeVersoes-CVS.pdf' -> '/home/fecosta/Apostilas/Cap47-ControleDeVersoes-CVS.pdf'
'curso_linux.pdf' -> '/home/fecosta/Apostilas/curso_linux.pdf'
'Dicionario_de_Termos_de_informatica-3ed.pdf' -> '/home/fecosta/Apostilas/Dicionario_de_Termos_de_informatica-3ed.pdf'
'DiscSoflivre_Aula_ComoUsaroTexLatex.pdf' -> '/home/fecosta/Apostilas/DiscSoflivre_Aula_ComoUsaroTexLatex.pdf'
'LENEP-GuiaAluno-Graduacao.pdf' -> '/home/fecosta/Apostilas/LENEP-GuiaAluno-Graduacao.pdf'
'LIVRO-06-RunningLinux_4th_Edition.pdf' -> '/home/fecosta/Apostilas/LIVRO-06-RunningLinux_4th_Edition.pdf'
'tesemaurocamara.pdf' -> '/home/fecosta/Apostilas/tesemaurocamara.pdf'
```

Trecho de Código 2.4.3: Exemplo do comando `cp -v`

No exemplo, utilizamos um diretório especial, o `~`, que pode ser usado para indicar o diretório pessoal (chamado de diretório *home*) de um determinado usuário. Caso seja usado apenas `~`, o usuário irá ao seu *home*. Para referenciar-se o *home* de um outro usuário, use `~<usuario>`. Por exemplo, se você precisar acessar o diretório de um usuário chamado `hpotter`, use `cd ~hpotter`.

Por fim, existe a opção `-preserve` (preservar), que permite que, na cópia, sejam preservados as permissões de acesso (ou *modo* — *mode*), a posse do arquivo (*ownership*) ou a data de última alteração (*timestamps*). No caso de cópias recursivas também podem ser preservadas quaisquer ligações simbólicas dentro de diretórios (*links*). Para preservar tudo, utilize `ALL`. Essa opção não pode ser combinada com as demais por ser uma *long option* (opção com nome longo).

2.4.2.2 Coringas *wildcards* no GNU/Linux

Muitas vezes, você precisa achar um determinado arquivo entre vários dentro de um conjunto de arquivos. Para isso, o uso de coringas (ou *wildcards*) ajuda bastante. Quase todo sistema operacional ou *shell* é capaz de trabalhar com coringas, e o GNU/Linux não é exceção.

O conjunto de caracteres coringas do GNU/Linux é derivado do sistema

de expressões regulares (*regular expressions*) do Unix, que é um sistema poderoso de reconhecimento de padrões. Não iremos falar muito sobre eles, mas é muito interessante aprender mais sobre eles, pois várias ferramentas úteis como **grep** e **find** dependam desse sistema para serem de total utilidade. Para mais informações, consulte a *manpage* do **regex(3)**, a documentação *infopage* do utilitário **grep** ou as especificações do OpenGroup, que mantem o POSIX (THE OPEN GROUP, 1997). Além disso, existe o bom livro “*Expressões Regulares — Guia de Referência Rápida*”, de Aurélio Jargas (JARGAS, 2003), que pode ser consultado *online* ou então comprado (se possível compre e ajude ao autor. Eu, ele e toda a comunidade do software livre agradecemos...:-P),

A idéia dos coringas é que você, mesmo que não conheça o nome do arquivo em questão, é que você conhece *uma parte do mesmo*, e pode usá-la, em combinação com os coringas, para encontrar o arquivo desejado.

```
[fecosta@hufflepuff CursoGNUlinux]$ ls
cap10.aux  cap1.tex~  cap6.tex      curso.bbl      hifenizacao.aux
cap10.tex  cap2.aux   cap7.aux      curso.bib      hifenizacao.tex
cap11.aux  cap2.log   cap7.tex      curso.bib~     hifenizacao.tex~
cap11.tex  cap2.tex   cap8.aux      curso.blg      intro.aux
cap11.tex~ cap2.tex~  cap8.tex      curso.dvi      intro.tex
cap12.aux  cap3.aux   cap9.aux      curso.dvi.pdf  lslh.txt
cap12.tex  cap3.tex   cap9.tex      curso.log      playsh-0.1.tgz
cap1.aux   cap4.aux   cap9.tex~     curso.lop      referencias.bib
cap1.bbl   cap4.tex   configuracao.aux curso.pdf      referencias.bib~
cap1.blg   cap5.aux   configuracao.tex curso.tex      x.log
cap1.log   cap5.tex   configuracao.tex~ curso.tex~
cap1.tex   cap6.aux   curso.aux     curso.toc
```

Trecho de Código 2.4.4: Listagem de exemplo para mostrar coringas

Por exemplo, considere uma lista de arquivos como a do Trecho de Código 2.4.4, na Página 37. Como podemos ver, existe uma grande quantidade de arquivos dentro desse diretório. Imaginemos que você precise apenas dos arquivos **.tex**¹¹. Você poderia tentar usar uma série de ferramentas do Unix para separar dessa listagem apenas os arquivos que lhe interessavam, mas:

1. Isso seria muito trabalhoso;

¹¹Arquivos do sistema de marcação de texto T_EXe/ou L^AT_EX, no qual esse documento foi produzido

2. Poderia acontecer de surgir *falsos positivos*, principalmente no caso dos arquivos com extensão `.tex`¹²

Para dar uma pesquisa adequada, vamos usar um coringa, o `*` (Asterisco). Esse coringa indica que no espaço em que ele entra esperam-se *zero ou mais caracteres quaisquer*. No caso, se usarmos `ls *.tex` no diretório do Trecho de Código 2.4.4, na Página 37, receberemos como resultado a listagem do Trecho de Código 2.4.4, na Página 37.

```
[fecosta@hufflepuff CursoGNUlinux]$ ls *.tex
cap1.tex*      cap2.tex*      cap6.tex*      configuracao.tex*
cap10.tex*     cap3.tex*      cap7.tex*      curso.tex*
cap11.tex*     cap4.tex*      cap8.tex*      hifenizacao.tex*
cap12.tex*     cap5.tex*      cap9.tex*      intro.tex*
```

Trecho de Código 2.4.5: Listagem de exemplo com o coringa `*` — asterisco

Esse uso é similar o do DOS/Windows, mas a versão do coringa no GNU/Linux é *muito mais poderosa*. Por exemplo, imagine que você queira acar todos os arquivos `.tex` que comecem com a letra `c`. O normal seria você usar `ls c*.tex`, mas na prática você pode utilizar o comando `ls c*tex`, pois o coringa também considera o `.` um caracter.¹³ O único problema com essa variante é que, caso exista um arquivo, por exemplo, `context`, ele será pego também, pois ele *casa com o padrão*.

Outro curinga útil é o `?` (ponto de interrogação), que casa apenas zero ou um caracter. Por exemplo, imagine que você queira pegar todos os arquivos `.tex` começados com `cap`, com um caracter qualquer e que tenham a extensão `.tex` na listagem do Trecho de Código 2.4.4, na Página 37. Você pode nesse caso utilizar o comando `ls cap?.tex`, obtendo o resultado do Trecho de Código 2.4.6, na Página 39.

Para mostrar que ele pode casar zero ou um caracter, veja o resultado do Trecho de Código 2.4.7, na Página 39, quando usamos o comando `ls`

¹²Em Unix, toda vez que um arquivo é editado, a maioria dos utilitários criam uma cópia de *backup* do arquivo alterado, com o nome do arquivo, incluindo extensões, terminado com um `~`.

¹³Isso deve-se ao fato de no Unix não existir realmente o conceito de extensão. A extensão em Unix é mais uma convenção que o usuário consiga detectar arquivos facilmente. Mesmo a extensão pode ser desnecessária, já que existem utilitários no GNU/Linux, como o `file`, que detectam o tipo de arquivo baseando-se em sua formatação interna.

```
[fecosta@hufflepuff CursoGNUlinux]$ ls cap?.tex
cap1.tex* cap3.tex* cap5.tex* cap7.tex* cap9.tex*
cap2.tex* cap4.tex* cap6.tex* cap8.tex*
```

Trecho de Código 2.4.6: Listagem de exemplo com o coringa ? — ponto de interrogação

`cap??.tex`. Perceba que também foram selecionados os arquivos da listagem do Trecho de Código 2.4.6, na Página 39, mesmo tendo apenas um caracter antes da extensão `.tex`.

```
[fecosta@hufflepuff CursoGNUlinux]$ ls cap??.tex
cap1.tex* cap11.tex* cap2.tex* cap4.tex* cap6.tex* cap8.tex*
cap10.tex* cap12.tex* cap3.tex* cap5.tex* cap7.tex* cap9.tex*
```

Trecho de Código 2.4.7: Listagem de exemplo com o uso de dois ?

Isso deve bastar por agora sobre o comando `cp` e sobre coringas.

2.4.2.3 mv

Depois de copiar arquivos, umas das principais operações com eles é os movimentar, retirando-os de um diretório para os colocar em outro. Para fazer isso, usamos o comando `mv`, que movimenta diretórios e arquivos para outro diretório, usando `mv <arquivodirorigem> <dirdestino>`. Se você usar contra apenas um arquivo, você pode usar `mv` para renomear um arquivo, usando `mv <nomeatual> <nomeantigo>`.

O `mv` aceita todas as opções comentadas em `cp`, exceto `-R`, que é redundante (ao mover um diretório, `mv` move todos os diretórios internos). É muito útil o uso de `-i` ou `-f`, dependendo do caso. Algumas vezes, `-v` também é bastante útil.

Como exemplo, imagine que eu queira mover um arquivo `hufflepuff` do *home* do usuário `cdiggory` para o meu (sendo que eu estou fazendo a operação). O comando para isso seria `mv ~cdiggory/hufflepuff ~`, lembrando que `~` é um diretório especial, que indica o diretório pessoal (*home*) do usuário em questão.

2.4.2.4 `rm`

Se queremos apagar arquivos no GNU/Linux, utilizamos o comando `rm`. Esse comando é muito perigoso quando usado inadequadamente, pois ele não oferece nenhum tipo de proteção quanto a deleção acidental, exceto se você usar a opção `-i`, como no caso do `cp`. O `rm` utiliza todas as opções do `cp`, como `-r` (*recursive*), `texttt-i` e `-f`. Na realidade, utilizar `rm -rf /` é o equivalente a destruir todos os dados em todas as partições disponíveis (veremos mais adiante, no Capítulo 3, página 65, porque esse comando é tão perigoso, quando entendermos como funciona a estrutura de diretórios do GNU/Linux). Fica o conselho: *nunca, JAMAIS utilize `rm -rf /`, exceto se você quiser pirar. Você foi avisado!!!*

Por exemplo, imagine que eu queira remover o arquivo `slytherin` de dentro do diretório do usuário `dmalfoy`. Para isso, utilizaria o seguinte comando: `rm ~dmalfoy/slytherin` (imaginando, claro, que eu tenha permissões para fazer essa operação).

2.5 Diretórios

Como tudo na vida, os arquivos em uma máquina precisam de organização. Na realidade, o GNU/Linux, como os Unix em geral, possuem uma boa estrutura, que permite uma fácil administração. Na prática, 90% desse sucesso se deve ao sistema de *diretórios* do arquivo. Nessa seção falaremos de operações com diretórios. Não falaremos aqui sobre a estrutura dos diretórios do GNU/Linux, sendo que há um Capítulo inteiro sobre isso, o Capítulo 3, página 65.

2.5.1 `cd`

O comando `cd` (*change directory* — mudar de diretório) permite que o usuário alterne de diretório livremente dentro do sistema (desde que ele possua privilégio de *execução* para aquele diretório — para diretórios, como comentamos na Seção 2.4.1.1, Página 32, isso significa que o usuário pode acessar aquele diretório). Não há nenhuma opção especial nesse comando.

Como exemplo, se quisermos alternar para o diretório `/etc` (diretório de arquivos de configuração — veremos mais sobre isso no Capítulo 3), usamos o comando `cd /etc`.

2.5.2 `mkdir`

Você pode criar suas próprias estruturas de diretórios, desde que você tenha privilégio de *escrita* no diretório aonde deseja-se criar o diretório. Isso é muito útil para organizar arquivos conforme suas necessidades, principalmente relacionadas a projetos de arquivos. Para isso, você usa o comando `mkdir` (*make directory* — criar diretório). Esse comando cria diretórios aonde você quiser, dentro de suas permissões de acesso.

Existem três opções úteis para o `mkdir`. A primeira é usar a já discutida opção `-v` (*verbose*) para mostrar o que o sistema está fazendo. Como a maioria dos comandos no GNU/Linux (e no Unix em geral), ele parte do princípio “se não deu pau, não fala nada”. O uso de `-v` ajuda a saber o que está acontecendo.

A segunda opção é a opção `-m` (*mode* — modo), que permite que, no momento da criação do diretório, você defina as permissões que deseja, sem ficar restrito ao `umask` (ou seja, aos padrões do sistema, que normalmente são `rw-r--r--` ou, no caso de diretórios, `rwxr-xr-x`). Isso pode ser muito útil para não ter que configurar-se posteriormente diretórios. Veremos mais sobre como configurar corretamente as permissões na Seção 2.7, Página 46.

A última opção em questão é a opção `-p` (*parents* — diretórios-pai). Normalmente, se você tentar criar um diretório dentro de outro, e esse outro não existir, o `mkdir` acusa erro. Com `-p`, você força o `mkdir` a criar os diretórios superiores ao diretório a ser criado, caso o mesmo não seja encontrado.

Vejamos um exemplo: imaginemos que eu queira criar no meu *home* um diretório `cdiggory` dentro do diretório `hufflepuff`, que eu sei não existir, e quero ver as coisas acontecer. Para isso, utilizo o comando `mkdir -pv ~/hufflepuff/cdiggory`, como no exemplo do Trecho de Código 2.5.1, página 42.

```
[fecosta@hufflepuff ~]$ mkdir -pv ~/hufflepuff/cdiggory
mkdir: created directory '/home/fecosta/hufflepuff'
mkdir: created directory '/home/fecosta/hufflepuff/cdiggory'
```

Trecho de Código 2.5.1: Exemplo do comando `mkdir -pv`

2.5.3 mv

Como dissemos anteriormente, para movimentar diretórios e arquivos, podemos usar o comando `mv`. Nós não iremos tratar dele aqui, pois ele já foi tratado na Seção 2.4.2.3, em 39.

2.5.4 rmdir

Para apagarmos diretórios, podemos usar o comando `rmdir` (*remove directory* — remover diretório). Ele elimina uma entrada de diretório, mas o diretório deve estar *vazio* para poder ser removido desse modo. Uma solução, se não houver arquivos em uma é usar a opção `-p` que foi discutida no comando `mkdir`, aonde ele vai apagando os diretórios pais, desde que estes *também* estejam vazios. Uma solução melhor para remoção recursiva é usar `rm -r`, pois ele elimina arquivos também. Além disso, podemos sempre contar com a opção `-v` (*verbose*) para alcançarmos nossos objetivos.

Por exemplo, imaginemos que queremos remover o diretório `~/hufflepuff/cdiggory` que criamos no exemplo do `mkdir`. Imaginando que não hajam mais arquivos ou diretórios em `hufflepuff`, podemos usar o comando `rmdir -pv ~/hufflepuff/cdiggory`, como no exemplo do Trecho de Código 2.5.2, página 42.

```
[fecosta@hufflepuff ~]$ rmdir -pv ~/hufflepuff/cdiggory
rmdir: removing directory, hufflepuff/cdiggory/
rmdir: removing directory, hufflepuff
```

Trecho de Código 2.5.2: Exemplo do comando `rmdir -pv`

2.5.5 Aonde Estou? `pwd`

Em todos os os nossos exemplos anteriores, o *shell* tem dado uma idéia de aonde estamos. Mas em algumas instalações do GNU/Linux, isso não é possível. Além disso, algumas vezes precisamos saber *exatamente* aonde estamos. Para isso, podemos recorrer ao comando `pwd` (*present working directory* — diretório atual de trabalho), que mostra o *caminho absoluto* (ou seja, a partir do diretório /) do diretório aonde você está. Basta digitar no *shell* `pwd`, como no exemplo do Trecho de Código 2.5.3, página 43.

```
[fecosta@hufflepuff ~]$ pwd
/home/fecosta
```

Trecho de Código 2.5.3: Exemplo do comando `pwd`

2.6 Passando para superusuário

Como dissemos ao falar do usuário `root` (Seção 2.1.1.1, Página 19), não é uma boa idéia (na verdade é uma **péssima** idéia) utilizar o `root`. O primeiro motivo é que sempre pode haver um pequeno erro de operação que comprometa *tudo* no seu sistema: basta uma barra mal-adicionada ou não adicionada e você pode complicar sua vida como administrador. O segundo motivo é que sempre existe a possibilidade de algum espertinho aproveitar aquela sua saidinha para o café para instalar uma ferramenta que comprometa a segurança do seu sistema, por exemplo.

Portanto, não há mais o que discutir: logar como `root`, NUNCA! Na realidade, existem algumas *poucas* situações na qual logar-se como `root` pode ser útil. Uma delas é quando você precisa realizar um *backup* geral da máquina, aonde é bom que nenhum usuário comum esteja logado (manutenção programada). Outra situação é no caso de uma invasão ter comprometido seu sistema: você deveria, se possível, isolar completamente o sistema e, como `root`, tirar uma imagem do sistema para análise forense.

De qualquer modo, continua o conselho: logue diretamente como `root` APENAS QUANDO **EXTREMAMENTE NECESSÁRIO**.

Mas e para aquelas atividades cotidianas do **root**, o que fazer? Graças aos céus, os gurus do Unix criaram duas ferramentas extremamente úteis para esses casos. A primeira, sobre a qual falaremos, é o comando **su**(*super-user* — super-usuário). Ela permite que você alterne para **root** sem precisar logar-se em um terminal por ela. Um exemplo de sua utilidade é se você for obrigado a se logar por um terminal remoto: é sempre mais interessante para despistar um potencial invasor logar-se como usuário normal e passar para **root** uma vez logado no sistema.

A segunda ferramenta, e muito adotada atualmente, é a **sudo**. Essa ferramenta libera que *certos comandos* administrativos possam ser executados por certos usuários. A grande vantagem é que o **sudo** exige, em geral, a senha *do próprio usuário*, não a do super-usuário. Ou seja, mesmo que um invasor esteja monitorando seu sistema, ele não conseguirá a senha do super-usuário. Isso pode não parecer grande coisa, mas qualquer buraco que você tampar é um a menos que o invasor contará para penetrar no seu sistema. Não iremos tratar nesse documento dessa ferramenta, primeiro pois ela não é um padrão nos sistemas Unix (embora sua popularidade seja crescente). A segunda razão é que a maior parte dos comandos que exigiriam **root** possuem formas de liberar potencialidades limitadas (mas úteis) para usuários comuns. A terceira é que existe muito material sobre o **sudo** na Internet. Um bom artigo sobre o **sudo** pode ser encontrado em Linux sem Mistério (2005).

Portanto, vamos ao nosso foco: o comando **su**.

2.6.1 su

O comando **su** permite que o usuário passe para super-usuário, mantendo algumas características do seu usuário normal. Por exemplo, se você estiver como super-usuário via **su** em um ambiente gráfico, você pode abrir programas que exijam instalação gráfica (como o *Netbeans* ou o JAVA JDK), o que pode ser muitíssimo útil em muitas situações.

Normalmente, você precisará apenas utilizar **su**. Você receberá um pedido de senha similar ao feito no *login*. Digite a senha de **root** e você passará para super-usuário, como mostrado no Trecho de Código 2.6.1, Página 45.

Perceba que após o “login” bem sucedido, o *prompt* mudou do **\$** para um **#**. Esse símbolo indica o *prompt* do **root**. Agora você pode fazer tudo


```
[fecosta@hufflepuff CursoGNUlinux]$ su  
Password:  
[root@hufflepuff CursoGNUlinux]#
```

Trecho de Código 2.6.1: Exemplo do comando `su`

o que você precisa, e depois basta digitar `exit` para sair do modo superusuário e voltar a sua conta comum, como faria para encerrar uma sessão normalmentoe.

Mas o melhor ainda está por vir... Vamos debater as opções úteis.

A opção `-c`(*command* — comando) é uma das melhores: com ela, você pode passar executar um único comando (fornecido junto com o comando `su`), sem precisar se logar como `root`. Isso é ótimo para aquelas pequenas tarefas, como aumentar a prioridade de um determinado aplicativo e afins. Por exemplo, se quisermos usar o comando `su` para compilar um programa com alta prioridade, poderia usar o comando `su` para lançar o comando adequado no *shell* do `root`, ou então posso simplesmente utilizar o comando `su -c 'nice -n -15 make'`, por exemplo. Ele pedirá-nos a senha do `root`, executará o comando e depois voltará ao nosso *shell* de usuário comum.

A opção `-s`(*shell*) permite que você escolha em que *shell* você irá se logar. Pode ser uma boa idéia, pois alguns *scripts* podem não rodar corretamente no `bash`, ou se o usuário possuir pouca experiência com o *shell* do `root` e desejar usar outro *shell*.

A última opção `-m`(*preserve environment*) não troca as variáveis de ambiente do sistema¹⁴ no *shell* do `root`. Isso é vantajoso quando você quer fazer algumas manutenções mas não quer dar “chance para o azar” e acabar acidentalmente lançando comandos que possam danificar o sistema.

¹⁴variáveis que preparam o ambiente de uso do sistema para o usuário, configurando, entre outras coisas, qual o editor padrão do sistema, aonde procurar programas, entre outras coisas

2.6.1.1 Usando su para acessar como usuário comum

Uma outra utilidade do **su** é que ele permite que você acesse como outro usuário, mesmo que não seja o **root**. Isso pode ser muito útil para aquele acesso rápido que você precisa dar no terminal de um amigo. Para fazer isso, basta digitar **su <usuario>** e digitar sua senha.

Uma vantagem disso é que você pode passar a usar usuários que normalmente não podem logar (em geral, usuários cujo *shell* são *null shell*, como **/bin/true** e **/bin/false**). Alguns programas, como o *MySQL*, exigem que você faça alguns comandos antes deles poderem operar normalmente pela primeira vez, mas são instalados com contas de usuário que não possuem *shell*. Nesse caso, pode-se optar por executar tais comandos como **root** (o que nem sempre funciona), ou, como é mais recomendável, entrar com a conta de usuário *null shell*. Nesse caso, o **su** adotará como *shell* o *shell* do usuário que está logado.

2.7 Permissões de Acesso

Na Seção 2.4.1.1, Página 32, começamos a falar sobre o sistema de permissão de arquivos. Nessa seção, falaremos sobre como configurar tais permissões de maneira adequada e falaremos sobre detalhes como *setuid*, *setgid*, *sticky bit* e *umask*.

2.7.1 Revisando Permissões de Acesso

Como dissemos anteriormente, as permissões de acesso são determinadas pelos, digamos assim, grupos **rwX**. Cada um desses grupos determinam as permissões de arquivos para o dono do arquivo (*owner*), para o grupo designado para o arquivo(*group*), e para os demais usuários do sistema(*other*). As letras **rwX**, vêm de *read*(ler), *write*, e *execute*(executar). Em geral, essas são as únicas permissões que um usuário típico irá precisar. Porém, existem mais três permissões bem interessantes (e potencialmente perigosas): *setuid* (defina como o ID do usuário), *setgid* (defina como o ID do grupo), e *sticky bit* (trave o arquivo). Vamos ver o que cada uma delas faz.

2.7.1.1 *setuid*

Algumas vezes, você precisa que usuários comuns executem programas ao qual não teriam permissão, como, por exemplo, trocar a sua senha com o utilitário *passwd*. Para isso, o Unix em geral (inclui-se o GNU/Linux) possui o conceito de *setuid*. Como definido em SetUID.org (2006), “*um componente de software que denominamos setuid altera o ID efetivo de usuário para o sistema a ser executado. Tipicamente, os programas são ativados com setuid para root, permitindo que o comando seja executado por um usuário normal como se ele tivesse sido invocado pelo super-usuário, o root*”.

O problema desse tipo de ativação são as potenciais falhas de segurança, como um usuário comum executar um programa *setuid* que sofra um *buffer overflow*¹⁵ e acabe o deixando com permissões superiores.

É importante tomar muito cuidado com *setuid* (*suid* para facilitar), pois ela é muito perigosa em termos de segurança de ambientes GNU/Linux. Um conselho sobre como proceder com o *suid* pode ser lido na *manpage* *setuid(7)*

2.7.1.2 *setgid*

O *setgid* é meio confuso, pois um programa ou arquivo *setgid* fica sob o mesmo tipo de efeito do *setuid*, mas com relação ao grupo definido para o arquivo/programa. Algumas vezes, isso pode ser útil para arquivos que as pessoas possam ler e editar, mas não há muita necessidade disso, em geral.

Em caso de diretórios, definir o *setgid* irá fazer com que qualquer arquivo criado dentro desse diretório passe a ser criado tendo como grupo dono definido o grupo dono do diretório em questão.

¹⁵Uma falha de segurança que acontece em certos programas aonde, por causa de uma alocação ou uso indevido de memória, o programa acaba “invadindo” o espaço de outro programa. Isso pode levar o programa a “invadir” o espaço do *shell* e permitir o lançamento de comandos arbitrários, dependendo do caso com permissões maiores que as que o usuário poderia ter

2.7.1.3 *sticky bit* — Bit de cola

Quando o *Sticky Bit* surgiu, sua função, como mostrado no *site* Unix for Advanced Users (2006) era manter programas no espaço de memória alocada, de modo que sua carga fosse mais rápida. Essa função original não é mais tão útil.

No entanto, nos sistemas modernos, como o GNU/Linux, o *stick bit* passou a ser usado para possibilitar a criação de diretórios públicos dentro do sistema com facilidade. Ao ativar *stick bit* para um diretório, o sistema é informado de que qualquer pessoa pode manipular os arquivos e diretórios dentro do diretório em questão. *Porém*, apenas o dono do arquivo, usuários do grupo do arquivo ou o **root** é que podem mover, renomear ou apagar o arquivo. Isso permite criar documentos públicos sem risco de alguém apagar seu conteúdo.

Perceba que essas permissões especiais podem ser detectadas com o uso de `ls -la`, como no caso do exemplo 2.7.1, na Página 48. No caso, perceba que arquivos com *setuid* e *setgid* podem ser identificados pelo **S** no lugar do atributo de execução (**x**), do usuário e do grupo, respectivamente. No caso do *sticky bit*, perceba que o atributo de execução dos outros usuários é substituído por um **t**.

```
[fecosta@hufflepuff teste]$ ls -la
total 20
drwxr-xr-x   3 root   root    4096 Abr 16 20:10 ./
drwxr-xr-x 123 fecosta fecosta 12288 Abr 16 20:09 ../
-rw-r-Sr--   1 root   root      0 Abr 16 20:10 setgid
-rwSr--r--   1 root   root      0 Abr 16 20:09 setuid
drwxr-xr-t   2 root   root    4096 Abr 16 20:10 sticky/
```

Trecho de Código 2.7.1: Exemplo do comando `su`

Agora que vimos estas três novas permissões, vejamos como alterar as permissões de arquivo.

Atenção: Os comandos a seguir *só podem ser executados* pelo **root** ou pelo usuário dono do arquivo, sendo que este pode executar os comandos (obviamente) apenas em arquivos de sua posse.

2.7.2 Mudando permissões — `chmod`

O primeiro e principal comando envolvendo permissões de arquivos é o `chmod` (*change mode* — mudar o modo), que permite alterar as permissões de acesso a um determinado arquivo. Para isso, ele utiliza o comando `chmod <permissões> <arquivo_dir>`. Antes de falarmos das permissões, que é um tópico um pouco mais complexo, iremos falar das opções interessantes.

As opções `-R` (*recursive*) e `-v` (*verbose*, já discutidas, também valem para o `chmod`. A elas vem se somar as opções `-c` (*change*), que funciona de maneira similar a `-v`, mas apenas alertando em caso de troca de permissões, e `--reference`. Esta é muito útil pois ela aparece para alterar as permissões de vários arquivos baseando-se em um determinado arquivo, permitindo ajustes de atributos conforme a necessidade.

As permissões podem ser atribuídas de duas formas, quanto ao preenchimento do campo `<permissões>`: baseando-se em *referência octal* ou *simbólica*. Vejamos as diferenças de cada uma.

2.7.2.1 Referência Octal

A *referência octal* basicamente utilizam quatro números de 0 a 7¹⁶ para traduzir as permissões de arquivo.

O primeiro número é considerado opcional, e pode ser descartado, pois ele trata das permissões especiais que tratamos anteriormente. No caso, você utiliza a soma dos números correspondentes aos atributos a serem definidos:

- 4 — *setuid*
- 2 — *setgid*
- 1 — *sticky bit*

Por exemplo, no caso de desejar-se definir um arquivo com *setuid* e *sticky bit*, você soma os valores 4 e 1, obtendo 5, que será o valor desse atributo.

¹⁶daí vem o *octal* — do sistema octal de numeração, que utiliza 8 dígitos: 0, 1, 2, 3, 4, 5, 6 e 7

Os três números seguintes serão os das permissões normais, que o sistema irá atribuir a dono, grupo e outros usuários. Para isso, o método é o mesmo do caso dos atributos especiais (escolhem-se os atributos para cada caso, soma-se os números respectivos e usa-se o número obtido como valor na representação octal)¹⁷. Nesse caso, nenhum dos três números pode ser descartado, mas se um deles não for receber permissões, pode-se usar o número 0 no lugar. Os valores para cada permissão nesse caso são:

- 4 — leitura;
- 2 — escrita;
- 1 — execução;

Por exemplo: se você quiser que apenas o usuário possa ler(4), gravar(2) e executar(1) um arquivo, que usuários comuns possam apenas o ler(4) e executar(1), e que outros usuários não possuam qualquer permissão de acesso(0), utilize a seqüência 750 ($4+2+1=7$, $4+1=5$).

Como você pode ver, é muito complexa essa configuração. Por isso, nas versões mais atuais do `chmod`, incluindo a versão GNU, pode-se usar uma representação simbólica, que falaremos a seguir.

2.7.2.2 Referência Simbólica

Nessa representação, utiliza-se caracteres que informam o tipo de atributo a ser definido. Em geral, eles são definidos da seguinte forma: primeiro vem uma ou mais letras que identificam quem terá a permissão alterada, em seguida um sinal que indica a alteração a ser feita, e por fim uma ou mais letras que indicam quais serão as permissões alteradas. A Tabela 2.2, na Página 51, mostra a estrutura da representação simbólica no `chmod`.

Por exemplo, para a definição que fizemos anteriormente, poderíamos usar o comando `chmod u=rwx,g=rx,o-rwx arquivo`. A vantagem dessa representação é que podemos modificar atributos com facilidade. Imagine que arquivo

¹⁷toda essa concepção tem a ver com o uso de lógica binária para o funcionamento do sistema de permissão de arquivos

USUÁRIO	PERMISSÃO	ATRIBUTO
<i>Em arquivos</i>		
u — Usuário	+ — ativar	r — ler arquivo
g — Grupo	- — desativar	w — gravar arquivo
o — Outros usuários	= — definir	x — executar arquivo
a — Todos os usuários		s — <i>(no usuário) setuid</i>
		s — <i>(no grupo) setgid</i>
		t — <i>sticky bit</i>
<i>Em diretórios</i>		
u — Usuário	+ — ativar	r — ler diretório
g — Grupo	- — desativar	w — gravar no diretório
o — Outros usuários	= — definir	x — acessar diretório
		t — <i>sticky bit</i>

Tabela 2.2: Permissões simbólicas

tenha permissões **rw-r--r--**. Se vemos o que queremos, basta remover os atributos de leitura e execução de arquivo, com um **chmod o-rx**.

Agora imagine um segundo arquivo, **arquivo2**, que tenha permissões **rw-r--r--**. Se queremos o tornar executável para todos os usuários, podemos usar **chmod ugo+x arquivo2** ou **chmod a+x arquivo2**, pois o **a** substitui **ugo**.

Isso deve ser o suficiente de **chmod**. Passemos agora para outro comando, o **chown**.

2.7.3 Mudando o dono do arquivo — **chown**

Algumas vezes, você precisará trocar a posse de arquivos e diretórios, seja por motivo de segurança (evitando que um usuário qualquer consiga privilégios altos no sistema), seja por conveniências (por exemplo, um usuário que saiu de férias e outro vai assumir seu lugar). Para isso, utilizamos o comando **chown** (*change ownership* — mudar a posse), que tem como sintaxe padrão **chown <usuario> <arquivo_dir>**. Nesse caso, **<usuario>** pode ser representado por um UID (difícil de ser lembrado, veremos mais sobre isso quando comentarmos os arquivos de administração) ou pelo nome de usuá-

rio. Pode-se também colocar o grupo ao qual ele pertence (por meio de `usuario:grupo`).

Existem muitas opções poderosas em questão. As já citadas `-R`(*recursive*), `-v`(*verbose*), assim como as opções `-c`(*change*) e `--reference`, citadas no `chmod`, também são válidas no `chown`, sendo que `--reference` copia o nome de usuário e grupo do arquivo de referência.

Uma opção nova e muito útil é a opção `--from(de)` que permite que o dono de um arquivo seja alterado apenas se o arquivo tiver pertencido a um determinado usuário. Isso é ótimo em diretórios públicos, principalmente se for usado em combinação com `-R`, pois evita que todos os arquivos passem acidentalmente a ser de um usuário, quando apenas determinados arquivos deveriam passar a ser posse do usuário.

Por exemplo, imagine a listagem do Trecho de Código 2.7.2, na página 52. O usuário `adumbledore` é o dono tanto do diretório `hogwarts` quanto dos arquivos dentro do diretório. Ele pode passar a posse dos arquivos em questão para outros usuários e tudo o mais. Imaginemos que existam três usuários `mmcgonagall`, `ssnape` e `hslughorn`. O usuário `adumbledore` passa a posse do arquivo `gryffindor` para o usuário `mmcgonagall` e a posse do arquivo `slytherin` para o usuário `ssnape`. Ele usa os comandos da listagem do Trecho de Código 2.7.3, na página 53, obtendo o resultado listado no mesmo Trecho de Código.

```
[adumbledore@hufflepuff hogwarts]$ ls -la
total 24
drwxr-xr-x  6 adumbledore hogwarts  4096 Abr 16 20:41 ./
drwxr-xr-x  4 adumbledore adumbledore 4096 Abr 16 20:40 ../
drwxr-xr-x  2 adumbledore hogwarts  4096 Abr 16 20:41 gryffindor/
drwxr-xr-x  2 adumbledore hogwarts  4096 Abr 16 20:41 hufflepuff/
drwxr-xr-x  2 adumbledore hogwarts  4096 Abr 16 20:41 ravenclaw/
drwxr-xr-x  2 adumbledore hogwarts  4096 Abr 16 20:41 slytherin/
[adumbledore@hufflepuff hogwarts]$
```

Trecho de Código 2.7.2: Lista de um diretório antes de ter seu usuário dono trocado

Perceba que agora aparece os usuários `mmcgonagall` e `ssnape` aparecem como os donos dos arquivos `gryffindor` e `slytherin`.


```
[adumbledore@hufflepuff hogwarts]$ chown mmcgonagall gryffindor
[adumbledore@hufflepuff hogwarts]$ chown ssnape slytherin
[adumbledore@hufflepuff hogwarts]$ ls -la
total 24
drwxr-xr-x  6 adumbledore hogwarts    4096 Abr 16 20:41 ./
drwxr-xr-x  4 adumbledore adumbledore 4096 Abr 16 20:40 ../
drwxr-xr-x  2 mmcgonagall hogwarts    4096 Abr 16 20:41 gryffindor/
drwxr-xr-x  2 adumbledore hogwarts    4096 Abr 16 20:41 hufflepuff/
drwxr-xr-x  2 adumbledore hogwarts    4096 Abr 16 20:41 ravenclaw/
drwxr-xr-x  2 ssnape        hogwarts    4096 Abr 16 20:41 slytherin/
[adumbledore@hufflepuff hogwarts]$
```

Trecho de Código 2.7.3: Lista de um diretório após troca de posse de alguns arquivos

Agora, imagine que o usuário `adumbledore` perca a posse do diretório `hogwarts` e de seus arquivos para o usuário `mmcgonagall`. Ao mesmo tempo, esse usuário passa a posse do arquivo `slytherin` para o usuário `hslughorn`. Nesse caso, primeiro o `root` lança um comando para o usuário `mmcgonagall` passar a ser o dono do diretório `hogwarts` e de seus arquivos, e então o usuário `mmcgonagall` pode passar a posse do arquivo `slytherin` para `hslughorn`. Essa sequência de operações e o diretório após essas alterações é demonstrada na listagem do Trecho de Código 2.7.4 na Página 54.

Esse exemplo deve ter dado uma boa idéia de como funciona o comando `chown`¹⁸. Agora vamos seguir em frente, para o comando `chgrp`.

2.7.4 Mudando o grupo dono do arquivo — `chgrp`

O comando `chgrp` (*change group* — muda grupo) muda o grupo ao qual um arquivo faz referência. Aos poucos, graças à funcionalidade de trocar-se o grupo do arquivo junto com o usuário dono por `chown`, o comando `chgrp` está caindo em desuso, mas ainda assim ele é muito útil. Sua sintaxe é similar à de `chown`: `chgrp <grupo> <arquivo_dir>`. Suas opções mais interessante já foram explicadas: `-R` (*recursive*), `-v` (*verbose*), `-c` (*change*) e `-reference`.

¹⁸E dado uma boa idéia de um *spoiler* (segredo da série) de “Harry Potter e o Enigma do Príncipe”

```
[root@hufflepuff hogwarts]# chown -R mmcgonagall .
[root@hufflepuff hogwarts]# ls -la
total 16
drwxr-xr-x  2 mmcgonagall hogwarts  4096 Abr 16 21:03 ./
drwxr-xr-x 121 mmcgonagall adumbledore 2288 Abr 16 21:04 ../
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 gryffindor
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 hufflepuff
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 ravenclaw
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 slytherin

\ldots

[mmcgonagall@hufflepuff hogwarts]$ chown hslughorn slytherin
[mmcgonagall@hufflepuff hogwarts]$ ls -la
total 16
drwxr-xr-x  2 mmcgonagall hogwarts  4096 Abr 16 21:03 ./
drwxr-xr-x 121 mmcgonagall mmcgonagall 2288 Abr 16 21:04 ../
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 gryffindor
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 hufflepuff
-rw-r--r--  1 mmcgonagall hogwarts    0 Abr 16 21:03 ravenclaw
-rw-r--r--  1 hslughorn  hogwarts    0 Abr 16 21:03 slytherin
```

Trecho de Código 2.7.4: Lista de um diretório após troca maciça de posse de arquivos

Como um exemplo, imaginemos o diretório `/GrimmauldPlace.12`, de propriedade do usuário `sblack`, que faz parte dos grupos `blackfamily` e `orderphoenix`¹⁹. O usuário deseja que outros usuários do grupo `orderphoenix` possam entrar no diretório `/GrimmauldPlace.12`, mas não os usuário do grupo `blackfamily`. Então, ele usa o comando `chgrp orderphoenix /GrimmauldPlace.12` e passa o grupo designado do diretório a `orderphoenix`. Com as permissões atuais (`rwxr-x---`), os usuários do grupo `orderphoenix` poderão entrar no diretório `/GrimmauldPlace.12`, mas não os de `blackfamily` (que entram agora como outros, sem privilégios de acesso, nem de ver os arquivos e diretórios dentro de `/GrimmauldPlace.12`). Os comandos são demonstrados na listagem do Trecho de Código 2.7.5, na página 55.

```
[sblack@hufflepuff GrimmauldPlace.12]$ ls -la
total 8
drwxr-x---  2 sblack blackfamily 4096 Abr 15 21:38 ./
drwxr-xr-x 23 root    root      4096 Abr 15 21:38 ../
[sblack@hufflepuff GrimmauldPlace.12]$ chgrp orderphoenix .
[sblack@hufflepuff GrimmauldPlace.12]$ ls -la
total 8
drwxr-x---  2 sblack orderphoenix 4096 Abr 15 21:38 ./
drwxr-xr-x 23 root    root      4096 Abr 15 21:38 ../
```

Trecho de Código 2.7.5: Lista de um diretório após troca de grupos

Com esse exemplo terminamos de demonstrar como mudar o dono e o grupo de arquivos e diretórios. Vamos agora então ver como deixar alguns padrões para a criação de arquivos e diretórios, com o comando `umask`.

2.7.5 Definindo as permissões padrão — `umask`

Algumas vezes, você pode querer definir padrões para as permissões de acesso a arquivos que sejam criados em um determinado diretório. Para isso, você utiliza o comando `umask` (*user's mask* — máscara do usuário).

A `umask` possui valores diferentes conforme o tipo de arquivo a ser criado, se é um arquivo de texto ASCII puro (incluindo aí códigos-fonte, *scripts* e

¹⁹Você pode não ter percebido, mas o autor é um grande fã de Harry Potter

umask	BINÁRIO	TEXTO	DIRETÓRIO
0	<code>rwX</code>	<code>rw-</code>	<code>rwX</code>
1	<code>rw-</code>	<code>rw-</code>	<code>rw-</code>
2	<code>r-x</code>	<code>r--</code>	<code>r-x</code>
3	<code>r--</code>	<code>r--</code>	<code>r--</code>
4	<code>-wX</code>	<code>-w-</code>	<code>-wX</code>
5	<code>-w-</code>	<code>r--</code>	<code>r-x</code>
6	<code>--X</code>	<code>---</code>	<code>--X</code>
7	<code>---</code>	<code>---</code>	<code>---</code>

Tabela 2.3: Permissões no `umask`

documentos como páginas HTML e fontes de documentos \LaTeX , um *binário* (programas compilados, imagens, áudio, vídeo...) ou se é um diretório²⁰. A tabela 2.3 e 56 mostra as permissões do `umask` (retirada do Guia Foca GNU/Linux(SILVA, 2005a)). Apesar dessa tabela complicada, existe uma “regra de dedo” para o dia-a-dia: faça a conta como você normalmente faria com uma *representação octal* de permissões, como falamos no comando `chmod`(Seção 2.7.2.1, Página 49) e subtraia 7. Isso deve ajudar no dia a dia, sem o obrigar a recorrer a tabelas e/ou decorar as permissões padrão²¹;

Uma utilidade do `umask` é para criar arquivos temporários já com restrições de acesso, sem precisar usar-se de `chmod`. Por exemplo, com o comando `umask 111; touch teste2`, cria o arquivo `teste2` com permissões `rw-rw-rw` ($4+2=6$, $7-6=1$)²².

Com isso, acabamos com a questão das permissões de arquivos. Então vamos seguir àdiantes em nossos estudos.

²⁰O Unix trata diretórios como arquivos especiais de funcionamento diferenciado

²¹Há uma ressalva aqui: é impossível criar um arquivo texto com permissões de execução ativas. Se o `umask` receber uma máscara que pudesse oferecer tal recurso, ele mantém a permissão de execução desativada. Isso é uma proteção de segurança oferecida pelo GNU/Linux, impedindo que espertinhos utilizem `umask` para criar arquivos executáveis à vontade

²²O comando `touch` é usado para alterar o rótulo de tempo de um arquivo. Se o arquivo não existir, ele o cria. Esse comando é muito usado por sistemas que exijam travas (*locks*), que o utilizam para criar seus *locks*

2.8 Vendo o conteúdo de arquivos

Muitas vezes, precisaremos ver o conteúdo de arquivos, principalmente arquivos de texto²³. Não trataremos da visualização de arquivos específicos, mas falaremos sobre como visualizar arquivos de texto, pois eles são os mais úteis no ambiente GNU/Linux (e Unix em geral), pois arquivos de configuração e *logs* no GNU/Linux são em geral representados no formato de arquivos texto.

2.8.1 cat

O comando `cat` (*concatenate* — concatenar) é usado como um visualizador de arquivos, mas pode ser usado (sua função original, na realidade, era essa) para *contatenar* (emendar) arquivos de texto. Seu uso normal é `cat <arquivo1> <arquivo2>...`, sendo que um vai em seqüência do outro ao ser exibido na tela. Um exemplo de saída do `cat` pode ser visto no Trecho de Código 2.8.1, na página 57.

```
[fecosta@hufflepuff CursoGNUlinux]$ cat /etc/resolv.conf
search gsmonline.intranet
nameserver 200.167.20.5
nameserver 200.176.2.10
nameserver 200.165.132.147

# ppp temp entry
```

Trecho de Código 2.8.1: Exemplo do comando `cat`

Existem duas opções úteis no `cat`, principalmente para programadores, que são relacionadas à exibição de numeração de linhas na saída do `cat`, que são as opções `-n` (*numbers*) e `-b` (*numbers-nonblank*). A diferença entre elas é que a primeira mostra números para *todas as linhas*, e a segunda *não mostra para linhas em branco*. Os Trecho de Código 2.8.2 (página 58), mostra a saída do Trecho de Código 2.8.1 (página 57), formatada pelos comandos `cat -n` e `cat -b`.

²³Aqui cabe uma ressalva: quando falamos *arquivos de texto* nesse momento, estamos fazendo referência a arquivos de texto puro ASCII. Nesse caso, por exemplo, inclui-se arquivos de códigos-fonte e *scripts*, além de documentos que recebem formatação por meio de marcas (*tags*), como páginas HTML, XML ou documentos L^AT_EX.

```
[fecosta@hufflepuff CursoGNUlinux]$ cat -n /etc/resolv.conf
 1 search gsmonline.intranet
 2 nameserver 200.167.20.5
 3 nameserver 200.176.2.10
 4 nameserver 200.165.132.147
 5
 6 # ppp temp entry
[fecosta@hufflepuff CursoGNUlinux]$ cat -b /etc/resolv.conf
 1 search gsmonline.intranet
 2 nameserver 200.167.20.5
 3 nameserver 200.176.2.10
 4 nameserver 200.165.132.147
 5
 5 # ppp temp entry
```

Trecho de Código 2.8.2: Exemplo do comandos `cat -n` e `cat -b`

Quando você dá um `cat` em um arquivo pequeno, tudo bem, você consegue visualizar todo o conteúdo em sua tela. Mas e quando você tem que visualizar um arquivo *muito grande*. Os comandos que veremos a seguir são muito, muito úteis nesses casos.

2.8.2 Visualizando arquivos grandes: `more` e `less`

Quando temos arquivos muito grandes, existem duas soluções para visualizarmos-os. A primeira é usar editores de texto como `emacs`, `vi` e `kwrite` para os visualizar. Essa é uma solução que nem sempre pode ser interessante, principalmente se analisarmos que sempre existe o risco de uma edição malfeita poder comprometer toda a estabilidade, segurança e até mesmo o funcionamento mínimo do sistema.

Nesse caso, uma idéia maior é usar programas visualizadores. O GNU/Linux traz vários deles, sendo que os principais são o `more` e o `less`. Ambos são usados da mesma maneira: `more <arquivo>` e `less <arquivos>`. Porém, o `more` permite apenas a leitura seqüencial, janela a janela de texto (quando uma janela de texto é ocupada, ambos, `more` e `less`, realizam pausas para que o usuário possa ler o conteúdo da tela e depois pressionar teclas para avançar no texto), enquanto o `less` permite um avanço linha a linha, assim como o recuo na leitura do texto. Portanto, recomendamos o uso de

less.

2.8.2.1 Entrada padrão, Saída padrão, redirecionamento e *pipes*

Aproveitamos aqui para fazer um adendo importante ao uso de comandos como *more* e *less*, que envolve os redirecionamentos e *pipes*.

O Unix original foi projetado com uma filosofia aonde cada comando executaria *apenas uma tarefa*, mas a executaria com maestria. Por sua vez, os comandos poderiam ser encadeados de forma a o resultado de um poder ser usado como entrada de dados do seguinte. Para isso, o Unix (e, por consequência, o GNU/Linux) utiliza o conceito de *entrada padrão*²⁴ e de *saída padrão*²⁵.

A idéia é que os programas não “travem” a entrada e saída de dados a um determinado dispositivo (como o teclado ou o vídeo), mas que aceitem entrada e saída de vários caminhos. O normal seria a entrada via teclado e a saída no monitor de vídeo. Porém, a vantagem desse sistema é permitir, por exemplo, que a saída de dados seja gravada em um arquivo qualquer, que um arquivo seja usado para entrada de informações ou que a saída de um comando seja a entrada de outro. Para isso, utilizam-se redirecionamentos e *pipes*.

Os redirecionamentos permitem que as entradas e saídas padrão, assim como a saída padrão de erros²⁶ possam ser redirecionadas. Na prática isso quer dizer que os dados de/para um programa serão passados para/de um arquivo, como se ele fosse a saída/entrada padrão. Por exemplo, o comando `cmd < teste.txt` executa o comando `cmd`, como se o conteúdo de `teste.txt` tivesse sido digitado diretamente pela entrada padrão (teclado). Da mesma forma, o comando `ls -la > teste.txt` mostra uma listagem completa de todos os arquivos dentro do diretório, mas mandando os resultados para o arquivo `teste.txt`, e não para a tela. A tabela completa de símbolos de redirecionamento pode ser encontrada na tabela 2.4, na página 60.

Os *pipes* (canos) são ainda mais úteis que os redirecionamentos de arquivo. Eles “ligam” virtualmente a saída padrão de um programa à entrada

²⁴também chamado de *standard input*, ou `stdin`

²⁵também chamado de *standard output*, ou `stdout`

²⁶*standard error output*, `stderr`

Símbolo	FUNÇÃO
<	Passa dados do arquivo como entrada padrão
>	Grava os dados da saída padrão para um arquivo.
>>	Anexa os dados da saída padrão a um arquivo.
2>	Como >, mas para a saída padrão de erros
2>>	Como >>, mas para a saída padrão de erros

Tabela 2.4: Símbolos de redirecionamento

padrão de outro, formando uma espécie de “*encanamento*” pelo qual os dados de um programa passam para o outro (por isso do seu nome). O símbolo para chamar-se um *pipe* é `|`. Por exemplo se você usar `ls -la` e tiver uma listagem muito grande, você pode usar o comando `ls -la | less`. Esse comando liga a saída do comando `ls -la` com a entrada padrão de `less`.

Isso deve bastar quanto a encadeamento e *pipes*. Vamos seguir em frente.

2.8.3 Vendo o final de arquivos: `tail`

Algumas vezes, existem arquivos *realmente grandes*, como logs de servidores como o Apache ou do próprio GNU/Linux, aonde mesmo o uso de `less` não ajuda muito. Para esses casos, o GNU/Linux possui um comando muito interessante, o `tail`, que apresenta as últimas linhas de um determinado arquivo. Seu uso normal é `tail <arquivo>` e ele normalmente exibe as 10 últimas linhas de `<arquivo>`.

As opções mais úteis do comando `tail` são `-l` (*lines* — linhas), aonde você define quantas linhas deverão ser exibidas, e `-f` (*follow* — acompanhar), que permite que quaisquer alterações no arquivo sejam monitoradas (muito útil quando usado contra arquivos de *log* do sistema)). Por exemplo, se quisermos monitorar o arquivo `/etc/httpd/log/access_log`, usamos o comando `tail -f /etc/httpd/log/access_log`.

Existe outra opção a `tail -f`, que veremos a seguir.

2.8.4 Acompanhando arquivos: `watch`

É possível acompanhar-se arquivos com o comando `tail -f`. Mas algumas vezes o sistema pode fazer a rotação de *logs*²⁷ o que torna o comando `tail -f` pouco útil. Além disso, o uso de `tail -f` pode “comer” processamento rapidamente, haja visto o fato de ele repetir o mesmo comando algumas dezenas de vezes por segundo.

Mas aqui nos vêem à salvação o comando `watch` (*observar*).

O comando `watch` é usado quando queremos repetir um determinado comando após um determinado período de tempo pré-estabelecido. A vantagem desse comando é que ele só repete o comando passado o período estabelecido (o padrão é a cada segundo), sem repetí-lo indefinidamente, não tendo, portanto, grande impacto no consumo de recursos do sistema.

Existem duas opções muito úteis nesse caso: a primeira é a opção `-n(number of seconds — segundos)`, que permite especificar o tempo mínimo de espera no `watch`, e a segunda é `-d(differences — diferenças)`, que permite acompanhar facilmente as diferenças sofridas pelo resultado do comando. Essa opção é útil, por exemplo, se você monitorar o tamanho de um arquivo via `ls -lh`.

Por exemplo, se você quiser usar `watch` para realizar o monitoramento sugerido no comando `tail` a cada 2 segundos de alteração, pode-se usar o comando `watch -n 2 'tail /etc/httpd/log/access_log'`. As aspas são importantes para fixar opções do comando a ser enviado. O `watch` abrirá uma janela de comando à qual pode ser fechada com o uso do comando `[Ctrl] + [c]`.

Isso encerra o assunto de visualização do conteúdo de arquivos. Então, vamos para nosso próximo tópico, aonde trataremos de um tipo especial de arquivos: as ligações.

²⁷*log rotation*, uma forma de manter os *logs* registrados, mas com um tamanho reduzido, compactando so *logs* mais antigos

2.9 Ligações

Muitas vezes, queremos acessar de maneira mais rápida um determinado arquivo ou diretório. Para isso, o Unix oferece um mecanismo bastante útil, as ligações (ou *links*). Essa seção será dedicada a falar sobre elas.

2.9.1 O que são ligações?

As ligações são *referências* que o sistema cria para um determinado arquivo. Existem dois tipos de ligações:

- **ligações simbólicas (*symlinks*):** *symlinks* são arquivos especiais do sistema (de um tipo especial *link*) que tem como conteúdo o caminho para chegar ao arquivo ou diretório referenciado. Pode ser criado por qualquer usuário, para arquivos e diretórios e de maneira que ele saia do sistema de arquivos atual, indo para um sistema de arquivo de outro dispositivo (falaremos mais sobre isso quando virmos comandos de montagem de sistema);
- ***hardlinks*:** são referências diretas criadas no sistema de arquivos ao *inode* de um determinado arquivo. Só pode ser criado pelo **root** e não pode ser feito para arquivos fora da partição do *hardlink*;

Perceba que no caso de comportamento de comandos, eles variam conforme o comando lançado. Comandos **mv** ou **rm** afetam o *link*, enquanto que comandos **cp** comandos de visualização, como **cat**, **less** e **ls** e comandos de edição, como **vi** ou **emacs**, afetam *o arquivo original*.

2.9.2 Vantagens e desvantagens das ligações

Criar ligações é uma ótima maneira de manter uma estrutura de diretória pseudo-isolada. Por exemplo, você pode colocar todos os arquivos de dados de trabalho dos usuários em uma partição separada e, usando *symlinks*, você pode criar ligações para esses arquivos nos diretórios dos usuários, sem problemas maiores ao usuário.

Porém, essa mesma facilidade pode voltar-se contra o administrador, na medida em que pode-se copiar arquivos e diretórios por meio de ligações, o que pode não ser desejável algumas vezes. Portanto, tome muito cuidado ao criar-se *symlinks* e, como “regra de dedo”, nunca, JAMAIS crie ligações para arquivos ou diretórios administrativos.

Vejamos agora então como criar ligações.

2.9.3 O comando `ln`

O comando `ln` serve para criar ligações de arquivos. Para isso, você irá usar o comando `ln <opcoes> <arquivo_original> <ligacao>`. No caso, existem apenas duas opções realmente úteis: `-s` para gerar ligações simbólicas e `-d` para criar *hardlinks* para diretórios. Por exemplo, se você desejar criar uma ligação simbólica para o arquivo `/etc/fstab` dentro da sua pasta de usuário, com o nome `teste`, use o comando `ln -s /etc/fstab teste`. Perceba que a listagem completa do arquivo `teste` não é nada parecida com o que estamos acostumados a ver no `ls -la`, como demonstrado no Trecho de Código 2.9.1, na Página 63.

```
total 2768
drwxr-xr-x 14 fecosta fecosta    4096 Abr 17 18:45 .
drwxr-xr-x 25 root      root      4096 Abr 10 11:45 ..
drwxr-xr-x 17 fecosta fecosta   12288 Abr 10 19:47 acessa
-rwxr-xr-x  1 fecosta fecosta 2673406 Abr 10 19:19 acessa.tar.gz
...

lrwxrwxrwx  1 fecosta fecosta     10 Abr 17 18:44 teste -> /etc/fstab
-rw-----  1 fecosta fecosta     49 Abr 13 17:45 .Xauthority
-rw-----  1 fecosta fecosta     95 Abr 13 17:45 .xsession-errors
```

Trecho de Código 2.9.1: Exemplo de uma ligação simbólica

Primeiro, note que as permissões são exibidas como `lrwxrwxrwx`. A primeira letra dessa permissão, `l`, mostra que o arquivo em questão é uma ligação simbólica. As permissões mostradas (`rw-rwxrwx`) *não se referem ao arquivo referenciado*, mas ao *link* simbólico. Mas tem uma pegadinha aqui:

as permissões normais do arquivo ligado *continuam valendo*. Portanto, não se preocupe tanto aqui...

Isso deve bastar no assunto ligações. Para terminarmos esses comandos básicos, veremos um comando muito útil, o *alias*, que cria *apelidos* para um comando em questão.

2.10 Apelidos para comandos: *alias*

Como já dissemos anteriormente, muitos comandos do GNU/Linux são muito perigosos quando lançados de maneira inconseqüente (principalmente quando lançados pelo *root*, sendo uma boa idéia se proteger de incidentes. Além disso, sempre é interessante utilizar-se de atalhos para comandos cotidianos. Para isso, o GNU/Linux oferece uma maneira de criar apelidos de comandos cotidianos, através do comando *alias*.

Para usar o comando *alias*, basta digitar *alias <novo_comando>='<comando_a_mapear>'*. Por exemplo, se você mapear um comando *alias longls='ls -l'*, toda vez que você digitar *longls*, você irá obter o resultado equivalente a *ls -l*.

Perceba que isso também funciona como uma forma de *sobrepôr* comando. Por exemplo, se você quiser evitar que o *rm* simplesmente apague tudo em seu diretório sem avisar, você pode usar o comando *alias rm='rm -i'*. Nesse caso, toda vez que você usar o comando *rm*, ele irá interpretar como *rm -i*. A única forma de contornar tais comportamentos seria usando a opção *-f* para sobrepôr o pedido de confirmação.

Com isso, terminamos esse capítulo inicial. O objetivo nosso *não é* tornar-lhe um especialista em *bash* ou em comandos do *shell* do GNU/Linux, até porque existem muitas referências de altíssimo nível na Internet para isso, como o Guia FOCA GNU/Linux(SILVA, 2005a), e sim oferecer-lhe bases para que possa seguir a diante nos estudos do GNU/Linux. A partir de agora, os exemplos serão cada vez mais complexos e conceituais: se puder praticar, ajuda bastante. De qualquer modo, atente para as dicas que serão espalhadas dentro desta apostila, pois elas irão lhe fornecer bases sobre os próximos passos a dar no GNU/Linux.

Capítulo 3

Estrutura de diretórios do GNU/Linux

Agora que vimos os comandos básicos do GNU/Linux, podemos começar realmente a aprofundarmo-nos nos funcionamentos internos do mesmo. Antes de seguirmos para outros tópicos, vamos falar mais aprofundadamente do sistema de arquivos e da estrutura de diretórios do GNU/Linux. Perceba que, ainda assim, não falaremos aprofundadamente sobre ela. Para isso, você pode sempre consultar o documento *Filesystem Hierarchy Standard* (FREE STANDARDS GROUP, 2004). Esse documento funciona como uma especificação da estrutura esperada de arquivos dentro de um sistema GNU/Linux, e faz parte do *Linux Standard Base* (FREE STANDARDS GROUP, 2005). A maioria das distros seguem uma estrutura que é bastante similar à essa, portanto ela deve ser o suficiente para a compreensão inicial.

3.1 Entendendo os diretórios em GNU/Linux

O GNU/Linux, como todos os sistemas operacionais Unix, é um sistema operacional que trabalha com sistemas de arquivo de *árvore única de diretório*. Na realidade, isso quer dizer que todos os dispositivos de armazenamento do GNU/Linux estão montados dentro de uma única estrutura de árvore de diretório, diferentemente, por exemplo, do WindowsTM, aonde cada dispositivo possui uma árvore de diretório independente.

Isso é muito interessante, pois facilita a procura por arquivos dentro de diversos dispositivos, mas ao mesmo tempo é muito perigoso: anteriormente, na Seção 2.4.2.4, na Página 40 comentamos que JAMAIS deveria utilizar-se `rm -rf /`. Como o GNU/Linux é um sistema de *árvore única de diretório*, todos os dispositivos de armazenamento do GNU/Linux estão montados dentro de uma única estrutura de árvore de diretório, que começa no diretório `/`¹. Um comando `rm -rf /` iria destruir não apenas dados de programas GNU/Linux, mas *de qualquer dispositivo o qual não esteja montado como somente-leitura* (veremos mais sobre montagem no Capítulo 4, mais exatamente na Seção 4.3, Página 82). Ou seja, muito provavelmente até mesmo dados de *backup* e dispositivos RAID² seriam destruídos.

Agora que temos uma noção da seriedade do sistema de arquivos do GNU/Linux, vejamos um pouco sobre três diretórios especiais do GNU/Linux, os diretórios `‘.’`, `‘..’` e `‘~’`, antes de passarmos para a estrutura padrão de diretórios do GNU/Linux.

3.1.1 O diretório `‘.’`

O diretório `‘.’` faz referência ao *diretório atual*. O uso de `‘.’` é muito vantajoso em alguns comandos, como o `cp` pois, diferentemente do caso dos comandos no MS-DOSTM ou WindowsTM, os no GNU/Linux não associam um alvo de destino a comandos. Isso é feito para evitar comandos mal-executados que possam sobrepor todo o conteúdo de arquivos, podendo até mesmo danificar o sistema. Nesse caso, por exemplo, se você quiser copiar dados para o diretório atual, use o comando `cp <arquivo> ..`.

Além disso, o uso do `‘.’` pode ser feito para procurar programas dentro do diretório atual. Por padrão, o GNU/Linux *não executa* comandos ou *scripts* dentro do diretório atual, de maneira a evitar que algum espertinho copie comandos administrativos dentro de diretórios aonde o usuário pode acessar e utilizá-los para violar a segurança do sistema. Porém, isso algumas vezes pode ser ruim, principalmente se você precisar testar novos programas

¹chamado de diretório raiz ou *root*

²*Redundant Array of Inexpensive Disks* — Conjunto Redundante de Discos Baratos, um mecanismo usado para *backup* e alta disponibilidade que garante o máximo de *uptime* — tempo de máquina funcional — com o mínimo de custo, através da combinação de discos comuns e tecnologias avançadas de espelhamento de dados

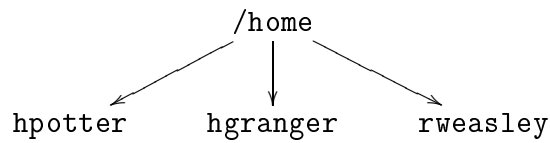


Figura 3.1: Exemplo de estrutura de diretórios

recém-compilados. Nesse caso, você pode usar o comando `./<comando>`, forçando o sistema a procurar por `<comando>` dentro do diretório atual.

De qualquer forma, isso deve bastar para falarmos sobre o *diretório atual* (`.`). Vamos falar agora sobre outro diretório especial, o `..`.

3.1.2 O diretório `..`

O diretório especial `..` (ponto-ponto) é um diretório que leva você ao *diretório anterior* ao seu (diretório-pai). Na prática, o `..` é muito usado para acessar-se diretórios que estão dentro do mesmo diretório. Por exemplo, imagine que você tenha a seguinte estrutura de diretórios, sendo que no momento você está no diretório `/home/hpotter`:

Se você precisar de um arquivo `potionessay` do diretório `/home/hgranger`, você pode usar `../hgranger/potionessay` para o acessar (supondo que você tenha permissões de acesso suficientes). O mesmo vale se você estivesse, por exemplo, no diretório `/home/rweasley`. A isso chama-se *caminho relativo* de um arquivo (ou seja, o caminho que deve-se seguir de um determinado diretório até um outro arquivo/diretório).

Com isso, encerramos esse tipo de diretório. Falta apenas falarmos sobre um diretório especial, o `~`.

3.1.3 O diretório `~`

O diretório `~`, chamado também de diretório *home*, identifica o *diretório pessoal* designado ao usuário. Esse diretório é registrado em geral junto com várias outras informações do usuário no arquivo `/etc/passwd` e normalmente

fica debaixo do diretório `/home`. Falaremos mais sobre esses arquivos e diretórios adiante. Por enquanto, o que importa é saber que usar `~` nos leva ao nosso diretório pessoal, e usar `~<nomeusuario>` nos leva ou referencia o diretório pessoal de outro usuário qualquer.

O diretório pessoal armazena, além dos documentos pessoais do usuário, os arquivos de configuração pessoal do usuário dos programas que ele usa, normalmente em diretórios ou arquivos *desinteressantes* (*ocultos*) dentro do *home*. Por exemplo, as configurações específicas deste usuário para o *shell* ficam em `.bashrc`, enquanto as do `emacs` ficam em `.emacs` e as do programa de edição de imagens The GIMP³ ficam dentro do diretório `.gimp`. É importante ressaltar uma regra do mundo Unix. Arquivos e diretórios ocultos (ou, usando o jargão Unix, “desinteressantes”) devem começar com o `.`. Isso é importante pois os arquivos “desinteressantes” passam por comandos de exclusão como `rm *`, tendo que ser excluídos *explicitamente* ou serem “vítimas” de exclusão recursiva.

Voltando ao exemplo anterior: imagine que o usuário `hgranger` tenha como diretório pessoal `/home/hgranger` e o usuário `hpotter` tenha como diretório pessoal `/home/hpotter`. Imagine que você esteja no diretório pessoal do usuário `hpotter` e precise copiar o arquivo `potionessay` do diretório pessoal do usuário `hgranger`. Qualquer um dos seguintes comandos fará a função⁴:

- `cp /home/hgranger/potionessay /home/hpotter` (comando completo);
- `cp /home/hgranger/potionessay .` (usando `.` para substituir o diretório atual);
- `cp ../hgranger/potionessay .` (usando o `.` para substituir o diretório atual e `..` para definir um *caminho relativo* até o diretório do usuário `hgranger`);
- `cp ~hgranger/potionessay .` (usando o `.` para substituir o diretório atual e `~hgranger` para indicar ao sistema que deseja-se obter um arquivo do diretório pessoal de `hgranger`);

³de *GNU Image Manipulation Program* — programa de manipulação de imagens do projeto GNU

⁴claro, desde que você tenha as permissões necessárias para isso

- `cp ~hgranger/potionessay ~` (usando `~hgranger` para indicar ao sistema que deseja-se obter um arquivo do diretório pessoal de `hgranger` e `~` para indicar que o destino é o diretório pessoal do usuário que está lançando o comando. Nesse caso, esse comando deve ser usado pelo usuário `hpotter` para alcançar-se o objetivo desejado);

Com isso, terminamos de falar sobre os diretórios especiais do GNU/Linux. Vamos agora falar da estrutura padrão de diretórios do mesmo.

3.2 A estrutura padrão de diretórios

O GNU/Linux permite, ao menos em teoria, que os arquivos e programas do sistema fiquem dispostos conforme a conveniência do administrador do sistema. Porém, isso tornaria o sistema uma verdadeira zona. Para desestimular tal comportamento, a hierarquia de diretórios e arquivos esperados em um sistema GNU/Linux foi padronizada na *Filesystem Hierarchy Standard* (Padrão de Hierarquia de arquivos) (FREE STANDARDS GROUP, 2004), parte do *Linux Standard Base* (FREE STANDARDS GROUP, 2005) (Bases de Padronização do Linux). Dessa forma, podemos esperar que determinados arquivos e programas estejam dentro de determinados diretórios, de forma que não precisemos nos preocupar muito com isso. Veremos a seguir como funciona essa estrutura de arquivos, segundo o FHS.

A estrutura típica de arquivos de um sistema GNU/Linux *básico* é a seguinte:

3.2.1 /usr

O diretório `/usr` contém os principais programas dentro do sistema GNU/Linux, além de informações necessárias para sua execução e uso (exceto configurações, que são desviadas para outro diretório, o `/etc`), além de informações de uso geral. Na prática, esse diretório contém a maioria dos programas e arquivos de utilidade geral do sistema. Pela FHS, o diretório `/usr` deveria ser considerado compartilhado (ou seja, todos os usuários deveriam ser capazes de acessar os comandos dentro dele) e apenas-leitura (apenas o `root` poderia

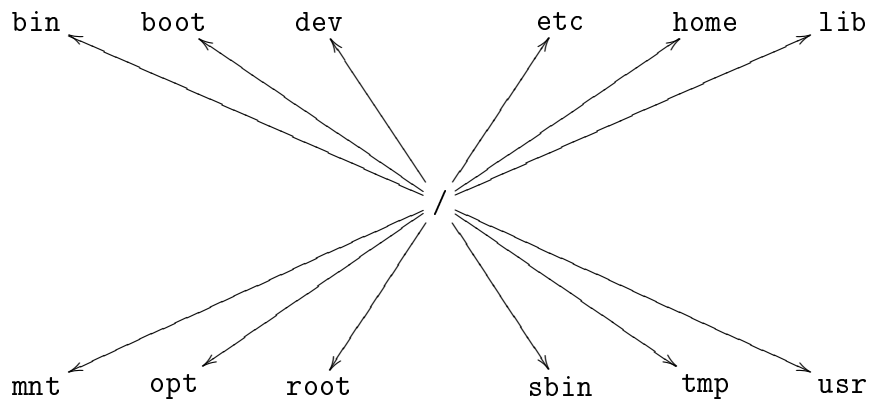


Figura 3.2: Estrutura padrão do GNU/Linux

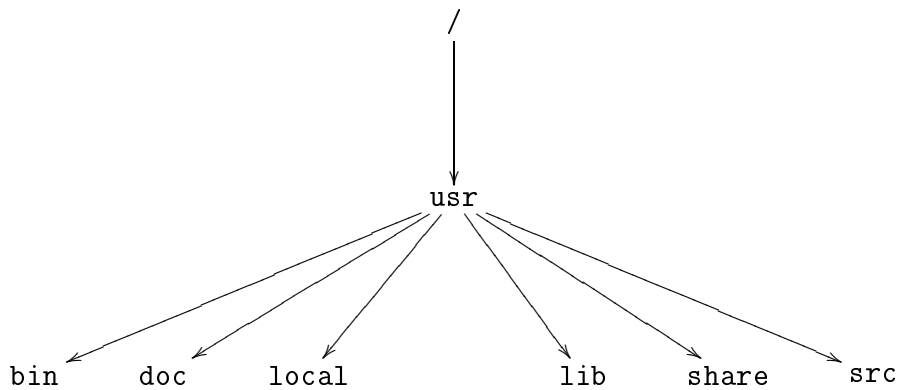
escrever arquivos dentro dele). Vejamos um pouco sobre os diretórios dentro de `/usr`, que são os seguintes:

3.2.1.1 `/usr/bin`

O diretório `/usr/bin` armazena os programas e utilitários principais, exceto no caso de programas utilitários específicos para o super-usuário (que em geral ficam ou em `/sbin` ou dentro de `/usr/sbin`). Normalmente, são colocado dentro desse diretório apenas aqueles aplicativos que vão ser compartilhado em toda uma infraestrutura de rede e/ou aqueles que são instalados com a própria distro, para facilitar sua manutenção.

3.2.1.2 `/usr/doc`

Esse diretório não é parte do FHS, mas em geral é um *link* simbólico para `/usr/share/doc` (que, por sua vez, é *parte* do FHS). Nele, ficam contidos todos os arquivos de documentação comum do sistema. Perceba que isso *não inclui* os arquivos de *manpages* e *infopages*, mas sim, por exemplo, documentações em formatos como TXT e HTML.

Figura 3.3: Estrutura do diretório `/usr`

3.2.1.3 `/usr/lib`

Nesse diretório ficam as bibliotecas de desenvolvimento de programas, além de bibliotecas de ligação dinâmica do GNU/Linux. No caso do desenvolvimento de programas em C/C++, os arquivos de cabeçalho (*header files*) ficam no diretório `/usr/include`.

3.2.1.4 `/usr/src`

A FHS afirma que “*códigos fonte podem ser colocados dentro desse diretório apenas para referências*”(FREE STANDARDS GROUP, 2004). Na mesma FHS, afirma-se que não deveria ser permitido o *build* (compilação) de programas dentro de `/usr/src`. Na prática, em geral, apenas o código fonte do *kernel* do GNU/Linux é compilado dentro de `/usr/src`. Em sistemas cujo gerenciador de pacotes é o `rpm`, o `/usr/src` também é utilizado para empacotar e copiar pacotes RPM gerados dentro do sistema, dentro de `/usr/src/RPM`

3.2.1.5 /usr/share

O diretório `/usr/share` atua como um depósito para arquivos de programa que não sejam dependente de plataforma. Isso permite que você possa compilar os programas conforme a plataforma e manter documentações e informações que sejam independente de plataforma isolados e uniformes para qualquer situação. O principal diretório dentro dele é `/usr/share/man`, que contém os arquivos de *manpage*. Esse é o único diretório que as especificações FHS consideram obrigatório. Além dele, pode-se citar:

- `/usr/share/info`, que guarda os arquivos *infopage*;
- `/usr/share/dict`, que registra listas de palavras para sistemas de dicionário;
- `/usr/share/locale`, que armazena informações de localização de *software* do sistema;

3.2.1.6 /usr/local

O objetivo de `/usr/local` é permitir que programas compilados no sistema sejam instalados sem prejudicar ou serem prejudicados por atualizações do GNU/Linux. Na prática, dentro de `/usr/local` encontra-se uma hierarquia completa similar à de `/usr` (com a óbvia exceção de `/usr/local`). Ou seja, você tem `/usr/local/bin`, `/usr/local/lib`...

Essa é a base do diretório `/usr`. Vamos seguir em frente.

3.2.2 /sbin

`/sbin` armazena os utilitários mais importantes do sistema, comandos de administração e de inicialização e desligamento do sistema. Utilitários como `shutdown` (desliga o computador corretamente), `fsck` (checagem do sistema de arquivos), `init` (responsável pela preparação inicial do sistema), `route` (configuração de roteamento em redes IP) e outros podem estar todos dentro desse diretório. Por esse motivo, aconselha-se que tal diretório possa ser acessado *apenas* pelo `root`,

3.2.3 /boot

Nesse diretório costumam ser armazenados os *boot loaders* (carregadores do sistema) e os arquivos do *kernel* compilado (tanto o *kernel* estático quanto quaisquer módulos dinamicamente carregados). Além disso, contêm arquivos de configuração dos *boot loaders*.

3.2.4 /root

`/root` é o diretório *home* do super-usuário. Normalmente aqui ficam *scripts* de uso pessoal do super-usuário para ajuda na administração do sistema. Como

3.2.5 /home

Dentro desse diretório ficam todos os diretórios *home* de usuários (diretórios pessoais). Essa hierarquia é considerada opcional pela FHS, mas é interessante mantê-la por causa do POSIX. Além disso, essa hierarquia torna muito mais simples o *backup* de arquivos do usuário e a manutenção do sistema (isole o diretório em outra partição e, caso haja uma falha no sistema, é possível recuperar os arquivos do usuário muito rapidamente).

3.2.6 /etc

O diretório `/etc` contém arquivos de configuração do sistema. Pela FHS(FREE STANDARDS GROUP, 2004), um arquivo de configuração é definido como “estático e não podendo ser binário executável”. Além disso, o diretório não devem existir arquivos executáveis. Segundo a FHS, é esperado os seguintes diretórios e arquivos:

- `/etc/opt`: diretório de configuração para os programas instalados em `/opt`;
- `/etc/X11`: diretório com os arquivos de configuração para o ambiente de interface gráfica X11;

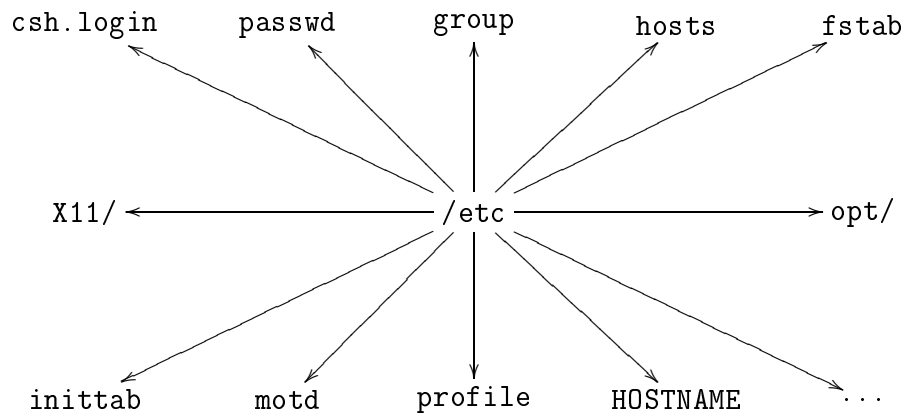


Figura 3.4: Estrutura do diretório /etc

- /etc/passwd e /etc/group: arquivos responsáveis pela manutenção dos registros de autenticação (usuários, senhas e grupos pertencentes) do sistema, quando o mesmo não utilizar-se de outros mecanismos (LDAP/PAM);
- /etc/hosts: arquivo com a configuração dos nomes de máquinas, quando a mesma não está usando DNS;
- /etc/fstab: Informações sobre os sistemas de arquivo a serem disponibilizados pelo sistema;
- /etc/inittab: Configuração do sistema de inicialização `init`;
- /etc/motd: *message of the day* — mensagem do dia: mensagem a ser exibida após o *login* do usuário via texto;
- /etc/profile: configurações padrão para *shell* bourne (`bash` incluído);
- /etc/csh.login: igual a /etc/profile, mas para *shell* C-Shell;
- /etc/HOSTNAME: contém o nome exclusivo da máquina local

A estrutura do /etc ficaria então assim:

Na realidade, existirão MUITO mais arquivos: em geral, *qualquer programa* no GNU/Linux deveria jogar suas configurações padrão aqui.

3.2.7 /var

O diretório **/var** contém arquivos que mudam constantemente de conteúdo, como *spools* (filas) de correio eletrônico e documentos a serem impressos, *logs*, *locks* de sistema, *cache* de navegação e *proxies* (como o **squid**⁵) e afins.

Anteriormente, esses conteúdos eram colocados em **/usr**, mas isso impedia que o sistema **/usr** fosse montado apenas para escrita. Atualmente, utiliza-se um *link* simbólico de **/var** para o diretório **/usr/var**. Dentro de **/var**, podem-se esperar os seguintes diretórios:

- **/var/cache**: informação de *cache* de aplicativos como **squid** e MySQL;
- **/var/lock**: *locks* de aplicações do sistema;
- **/var/log**: arquivos e diretório de *log* tanto do sistema quanto de programas;
- **/var/run**: informação dos processos em execução;
- **/var/spool**: *spool* de dados das aplicações como email e impressão;
- **/var/tmp**: informação variável que deve ser mantida entre reinicializações do sistema;

A estrutura do **/etc** ficaria então assim:

3.2.8 /mnt

O diretório **/mnt** contém em geral diretórios vazios conhecidos como *pontos de montagem*. No caso, não falaremos sobre eles aqui, pois esse é um assunto para a Seção 4.2, Página 4.2. A título de curiosidade, um diagrama de exemplo de um diretório **/mnt**:

⁵servidor *proxy* de Web capaz de filtro de conteúdo

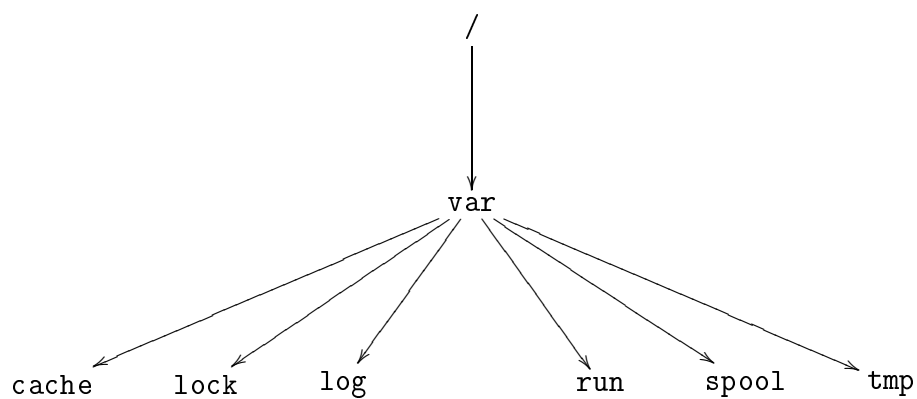


Figura 3.5: Estrutura do diretório `/var`

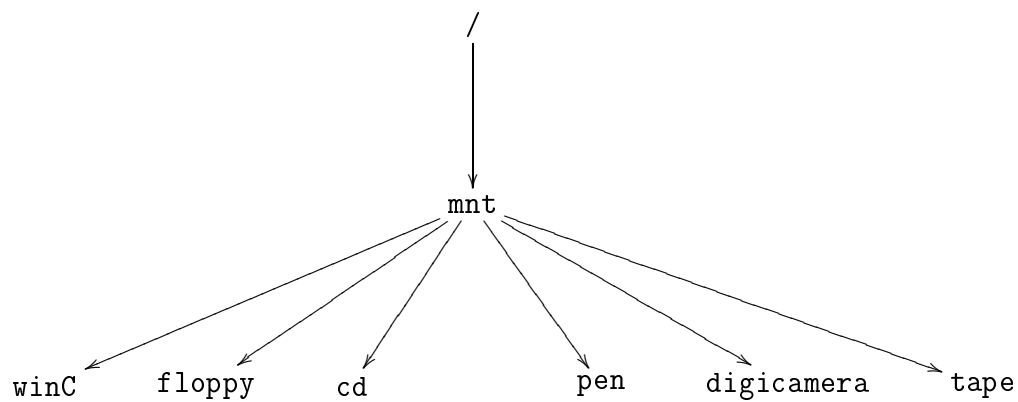


Figura 3.6: Exemplo de uma estrutura de pontos de montagem em `/mnt`

3.2.9 /opt

O diretório `/opt` contém em geral pacotes adicionais ao sistema. Embora não seja a função definida pela FHS, muitas instalações GNU/Linux aproveitam esse diretório para guardarem programas proprietários ou programas binários cuja instalação é diferente da padrão do sistema.

3.2.10 /dev

O diretório `/dev` é um diretório especial, pois ele contém os arquivos de dispositivo. Para entender, vamos explicar o porquê disso.

Para os Unix em geral (GNU/Linux entre eles), qualquer coisa dentro do sistema é um arquivo. Periféricos são arquivos, memória é arquivo, processos são arquivos, etc... No caso, para acessar um dispositivo, o Unix cria *arquivos de dispositivos* que são referências aos dispositivos. Por exemplo, para mandar uma listagem de arquivos para a impressora, você pode usar o comando `ls -la > /dev/lp0`, aonde `/dev/lp0` é a primeira impressora do sistema.

No `/dev` encontram-se todos os diretórios de dispositivo do sistema GNU/Linux. Em geral, os nomes dos dispositivos são dados por um tipo de barramento, mais o número do dispositivo no sistema. Por exemplo `/dev/lp0` (primeira impressora) `/dev/tty2` (terceiro terminal virtual)⁶... Não entraremos em detalhes nesse assunto, primeiro porque as partes relevantes serão discutidas no decorrer da apostila, em segundo lugar pelo fato de ser um assunto *extremamente complexo* e fora do escopo dessa apostila.

3.2.11 /proc

O diretório `/proc`, na verdade, é um pseudo-sistema de arquivos (ou seja, ele não existe em dispositivos de armazenamento), representando o *kernel* e os processos em execução. Em geral, não se manipula o diretório `/proc`, embora algumas vezes seja necessário manipular tais arquivos para ativar ou desativar opções do *kernel on-the-fly* (com o sistema funcionando).

⁶O começo por 0 é uma herança Unix da linguagem de programação C, aonde em geral a primeira posição dentro de um vetor é apontada por 0

Com isso, encerramos esse tópico. No próximo, veremos mais sobre o acesso a dispositivos externos e o agregamento de novos dispositivos ao sistema.

Capítulo 4

Acessando dispositivos no GNU/Linux

Com seu sistema de arquivos de árvore única, o acesso à informação no GNU/Linux (como no Unix em geral) sempre foi voltado à possibilidade de adicionar-se espaço de armazenamento *on-the-fly*, com o mínimo de *downtime* possível. Para isso, o GNU/Linux recorre a procedimentos de montagem e desmontagem de dispositivos e a pontos de montagens estruturados.

Nesse capítulo, nosso objetivo é que o leitor compreenda esses dois conceitos, assim como os comandos necessários para montar e desmontar dispositivos e um arquivo muito importante de configuração dos pontos de montagem do sistema, o `/etc/fstab`.

4.1 Conceito de “montagem de dispositivos”

Um conceito errôneo formado pelos sistemas automatizados de leitura de discos removíveis é de que os dispositivos de armazenamento ficam prontamente disponíveis uma vez inseridos no sistema. Na realidade, apenas um dele (o disco rígido) o fica, e ainda assim graças ao sistema de *bootstrapping*¹.

¹*bootstrapping* é o processo pelo qual o BIOS dispara a inicialização do sistema operacional, em combinação com os programas de *bootstrap*

Na realidade, todo dispositivo de armazenamento deve ser *montado* antes de ser usado e *desmontado* após o uso. O termo “*montar*” um dispositivo quer dizer informar ao sistema que o dispositivo está pronto para ser usado. O Unix, para aumentar sua eficiência, *não* procura entre os dispositivos de armazenamento quais deles estão prontos para o uso (embora algumas distros de GNU/Linux possuam um sistema atualizado, o **automount**, que faz esse serviço): o administrador (ou os usuários, com as corretas configurações) deve informar ao sistema que o dispositivo está preparado e pode ser usado.

Em alguns casos, após o uso, o sistema deve ser informado que o dispositivo em questão não será mais utilizado e será retirado do sistema. A isso, chamamos de “desmontagem”. No Unix (e no GNU/Linux), esse processo é muito importante, pois, novamente para questão de eficiência, o sistema *nem sempre* grava os dados no dispositivo de armazenamento no momento em que eles são “salvos”, algumas vezes mantendo os dados em um *cache* na memória principal até que os dados são gravados em definitivo no dispositivo de armazenamento (o que é chamado de *sincronização* ou *syncing*). Remover um dispositivo sem o “desmontar” pode acarretar na corrupção de dados ou até mesmo em danos ao dispositivo. De qualquer modo, fica o recado. No caso de uso de sistemas de automontagem, como o **automount**, essa preocupação é desnecessária, pois o dispositivo é sincronizado de imediato.

A vantagem disso é que é possível, através de sistemas remotos de arquivos, como o NFS (*Network File System*), montar-se dispositivos em servidores remotos, com coletâneas de programas muito grandes, como por exemplo o sistema \TeX – \LaTeX .

Esclarecido esse conceito, vamos ao conceito dos “pontos de montagem”.

4.2 Pontos de montagem

Em Unix (portanto, no GNU/Linux), “pontos de montagem” são diretórios especialmente designados aonde o sistema irá montar a estrutura de arquivos de dispositivos que estejam “montados”.

Na prática, qualquer diretório pode ser um “ponto de montagem”, mas sugere-se que sejam diretórios vazios (pois a montagem faz com que a estrutura de arquivos no dispositivo sobreponha a existente no diretório, embora

os dados em questão não desapareçam, podendo serem acessados após a desmontagem do dispositivo) e estruturados em relação ao sistema de arquivos principal (/).

Uma característica interessante do GNU/Linux permitida graças árvore de diretório única é que partes do sistema possam ser dispostas em partições (ou até mesmo dispositivos) diferenciados. Isso porque pode-se usar *qualquer* diretório como ponto de montagem, portanto pode-se, por exemplo, montar-se um dispositivo (ou uma partição — para o GNU/Linux, partições dentro de um mesmo dispositivo são considerados dispositivos diferentes) como, por exemplo, o diretório `/home`. Isso ajuda em algumas coisas:

- Limita o consumo de espaço em disco para determinados tipos de informação dentro do sistema: se você quer que os dados dos usuários do sistema ocupem no máximo 20 Gigabytes, você pode criar uma partição de 20 Gigas em seu disco e montá-lo como o diretório `/home`;
- Facilita o *backup*, o *upgrade* e o *disaster recovery* do sistema: exceto pela eventualidade de um `rm -rf /`, uma boa estrutura de diretórios permite que você possa, por exemplo, reinstalar um sistema que perca seu *kernel* (do diretório `/boot`) sem afetar os dados dos usuários (`/home`), desde que ambos estejam, em dispositivos/partições diferentes. Isso também afeta estratégias de *backup* (você não precisará de *backups* de `/usr`, ao menos não frequentemente) e assim por diante;
- Permite o uso de algumas estratégias de QoS (*Quality of Service*): você pode montar em um disco rígido rápido o diretório aonde ficam os dados do servidor de banco de dados, e colocar em discos mais lentos os dados dos usuários no servidor de arquivos, por exemplo;
- Montagem remota de grandes volumes de informações repetidas permitem poupar o disco do usuário, embora exija uma boa infraestrutura de rede: programas como L^AT_EX, GNU `emacs` e X-Windows podem ser compartilhados via NFS e montados no sistema local, sendo que suas configurações seriam localizadas (apenas os binários dos programas seriam compartilhados).

Uma boa estratégia para montagem de dispositivos pode oferecer grandes ganhos em tempo, performance e menos dores de cabeça ao administrador do sistema.

4.2.1 Dispositivos removíveis e pontos de montagem

Em geral, os dispositivos removíveis em um sistema são montados em pontos de montagem dentro de `/mnt` ou, o padrão da FHS, `/media`(FREE STANDARDS GROUP, 2004). Em ambos os casos, sugere-se que sejam criados diretórios específicos para cada um dos dispositivos removíveis com nomes característicos(`cdrom`, `cdrecorder`, `dvd`, `pen`, `digicamera`, `olympus`, ...). Isso ajuda (e muito) a localizar os dispositivos removíveis dentro da estrutura de árvore única do GNU/Linux.

Agora que vimos o suficiente sobre os conceitos de montagem, desmontagem e ponto de montagem, vamos colocá-los na prática para acessarmos dispositivos.

4.3 Montando e desmontando dispositivos: `mount` e `umount`

O Unix possui comandos para montagem e desmontagem de dispositivo. Como esse processo é considerado *muito perigoso* (desmontar um dispositivo *on-the-fly* ou montar um dispositivo sobre um diretório em uso pode ser algo catastrófico quando feito de maneira errada), em geral apenas o `root` consegue realizar essas operações.

O comando `mount` tem como sintaxe `mount <opções> <dispositivo> <ponto_montagem>`, aonde ele pega o dispositivo em `dispositivo` (em geral indicado como um arquivo em `/dev`) e o monta em `<ponto_montagem>`. Em geral os dispositivos são nomeados como demonstrados na Tabela 4.1, página 83. Em todos os casos, você precisará indicar qual partição do dispositivo deverá ser acessada. Para isso, pegue o número da partição e o coloque a frente do nome do dispositivo. Por exemplo, a primeira partição do primeiro disco IDE é referenciada usando `/dev/hda1`, enquanto que a quinta partição do segundo disco SCSI dentro do sistema é referenciada com `/dev/sdb5`, por exemplo. O ponto de montagens é simplesmente um diretório do sistema (preferencialmente vazio) aonde o dispositivo será montado.

Em geral, o GNU/Linux consegue detectar o tipo de conteúdo que está no dispositivo, mas algumas vezes pode ser útil determinar o tipo de sistema

NOME DO DISPOSITIVO	DISPOSITIVO
<code>/dev/hda</code>	Primeiro disco IDE
<code>/dev/hdb</code>	Segundo disco IDE
<code>/dev/hdc</code>	Terceiro disco IDE
<code>/dev/hdd</code>	Quarto disco IDE
<code>/dev/fd</code>	Unidades de disquete
<code>/dev/sda</code>	Primeiro dispositivo SCSI ou unidade USB
<code>/dev/sdb</code>	Primeiro dispositivo SCSI ou unidade USB
<code>/dev/sdc</code>	Primeiro dispositivo SCSI ou unidade USB

Tabela 4.1: Alguns exemplos de nomes de dispositivos

de arquivo do dispositivo. Para isso, utilize a opção `-t`(*type* — tipo), informando o tipo de sistema de arquivos a ser usado. No caso, algumas opções muito comuns a serem usadas no GNU/Linux estão listadas na Tabela 4.3, na Página 84 (inspirada em Casha (2001)). Para saber quais os sistemas de arquivo que o *kernel* pode montar, use o comando `cat /proc/filesystem`, pois ele irá listar todos os módulos de sistema de arquivos compilados no sistema

Algumas vezes, pode ser útil montar um dispositivo apenas para leitura, mesmo que ele permita escrita (como um HD que acabou de ser invadido por um invasor de sistemas). Para isso, usamos a opção `-r`(*read-only* — apenas). Por exemplo, para montar o dispositivo `/dev/sda1`, que tem como sistema de arquivos o `vfat`, no diretório `/mnt/pen`, utilize o comando `mount -t vfat /dev/sda1 /mnt/pen`, como `root`. Ele não deverá retornar nenhuma mensagem se estiver tudo OK. Nesse caso, você poderá fazer qualquer uso do dispositivo montado, como, por exemplo, listar os arquivos dentro do dispositivo, como mostrado no Trecho de Código 4.3.1, na Página 85.

O *mount* possui uma grande quantidade de opções úteis específicas para cada tipo de sistema de arquivos. Para maiores informações, consulte a *manpage mount(1)*

¹Experimental, nas versões mais atuais do *kernel* do GNU/Linux o sistema pode gravar em arquivos já existentes, mas não criar novos arquivos/diretórios

¹Imagens de CD prontas para serem gravadas, muito usadas nas trocas de distros GNU/Linux

²Sistema que permite rápida recuperação do sistema em caso de falha

NOME DO DISPOSITIVO	COMENTÁRIOS
<i>Sistemas padrão</i>	
ext2	Sistema de arquivos original do GNU/Linux
swap	Partição de memória virtual do GNU/Linux
iso9660	Sistema de arquivos de CD-ROMs Permite acessar ISOs ² de CDs
<i>Sistemas externos</i>	
vfat	FAT-16 ou FAT-32 (DOS/Windows)
ntfs ³	WindowsNT/2000/XP (Apenas-Leitura)
hpfs	OS/2
hfs	Apple System 7/MacOS
<i>Sistemas com journaling⁴</i>	
ext3	Evolução do ext2
reiser	ReiserFS, sistema <i>journaled</i> alternativo
xfs	Padrão do Irix, porte nativo no Linux
jfs	Desenvolvido pela IBM para sistemas de alta <i>performance</i>
<i>Sistemas de Rede</i>	
nfs	Volume remoto via <i>Network File System</i>
smb	Pastas compartilhadas Windows/SaMBa

Tabela 4.2: Alguns sistemas de arquivos que podem ser acessados pelo GNU/Linux


```
[root@hufflepuff ~]# ls -la /mnt/pen
total 107116
drwxrwxrwx  8 root root    22016 Dez 31  1969 .
drwxr-xr-x 27 root root    4096 Abr 20 18:22 ..
-rwxrwxrwx  1 root root   36352 Abr 14 13:14 00%20-revis%E3o%20para%20a%20prova.doc
-rwxrwxrwx  1 root root   697788 Abr 17 16:12 1145055685803.pdf
-rwxrwxrwx  1 root root   849465 Abr 20 13:38 2005-02-17 Aula 1.pdf
-rwxrwxrwx  1 root root   10036 Abr 17 15:13 4321.jpg
-rwxrwxrwx  1 root root   10478 Abr 17 16:11 4347.jpg
-rwxrwxrwx  1 root root   22670 Abr 14 16:20 aaai-named.bst
-rwxrwxrwx  1 root root   132195 Abr 17 16:36 abntex-0.9-beta.noarch.rpm
-rwxrwxrwx  1 root root   54108 Abr 17 14:03 AUGIE.ttf
-rwxrwxrwx  1 root root   659720 Abr 20 13:41 Aula_2_????_HTTP_e_FTP.pdf
-rwxrwxrwx  1 root root   179022 Abr 20 13:41 Aula_3_DNS.pdf
-rwxrwxrwx  1 root root   526827 Abr 20 13:42 Aula_4_-_Sistema_Completo.pdf
-rwxrwxrwx  1 root root    610 Abr 13 19:20 .log
-rwxrwxrwx  1 root root    8588 Abr 18 17:53 mandamentosrpm.html
drwxrwxrwx  2 root root    8192 Abr  6 10:15 NerdTV
-rwxrwxrwx  1 root root   36046 Abr 17 13:58 00o Professional.otp
-rwxrwxrwx  1 root root   297176 Abr 17 14:11 squares.otp
drwxrwxrwx  2 root root    2048 Abr 18 00:40 Upgrading to 2.2 from 2.0 - Apache HTTP Server_arquivos
-rwxrwxrwx  1 root root   12598 Abr 18 00:40 Upgrading to 2.2 from 2.0 - Apache HTTP Server.html
-rwxrwxrwx  1 root root 35140553 Abr 18 16:31 UTILS.zip
```

Trecho de Código 4.3.1: Exemplo do comando `ls` em um dispositivo “montado”

Uma vez que você tenha feito todas as operações desejadas, você pode (na verdade deveria) desmontar o dispositivo, usando o comando `umount`. Para desmontar o dispositivo, você pode usar tanto fazer referência ao dispositivo quanto ao ponto de montagem. No caso anterior, tanto `umount /dev/sda1` quanto `umount /mnt/pen` devem funcionar para desmontar o dispositivo.

Perceba que, por segurança, o GNU/Linux não irá autorizar a desmontagem de um dispositivo que tenha:

1. arquivos abertos;
2. usuários acessando arquivos;
3. usuários dentro de sua árvore de diretórios;

Nesse casos, pode-se usar comandos como `lsof` para detectar os usuários que estão utilizando o sistema, procurando assim formas de como fazer com que estes usuários terminem seus usos.

Após todos os usuários terem saído do ponto de montagem, o comando `umount` pode ser disparado sem problemas. Ele força o *sync* do dispositivo a ser desmontado e devolve o *prompt* do `root`, indicando operação bem sucedida.

O maior inconveniente do uso do `mount` está naqueles dispositivos que devem ser montados a cada inicialização. Isso fica ainda pior quando, por falta de espaço em um disco, você o troca e precisa desmontar o sistema antigo para ativar o sistema novo. Nesse caso, você pode “forçar” a inicialização dos dispositivos necessários. Isso é feito automaticamente pelo sistema, através de um arquivo de configurações, o `/etc/fstab`.

4.4 O `/etc/fstab`

O `/etc/fstab` (de *filesystem table* — tabela do sistema de arquivos) contém os registros de como é a estrutura do sistema de arquivos do GNU/Linux. Na prática isso quer dizer que o `/etc/fstab` contém quais são os dispositivos e partições que fazem parte do sistema de arquivos e como eles devem ser montados. O arquivo é um arquivo texto puro, sendo possível editá-lo por meio de programas simples como `vi` ou `emacs`. Um exemplo de `/etc/fstab`, retirado do Guia Foca GNU/Linux (SILVA, 2005a), pode ser encontrado no Trecho de Código 4.4.1, na Página 86.

<code>/dev/hda1</code>	<code>/</code>	<code>ext2</code>	<code>defaults</code>	<code>0</code>	<code>1</code>
<code>/dev/hda2</code>	<code>/boot</code>	<code>ext2</code>	<code>defaults</code>	<code>0</code>	<code>2</code>
<code>/dev/hda3</code>	<code>/dos</code>	<code>msdos</code>	<code>defaults,noauto,rw</code>	<code>0</code>	<code>0</code>
<code>/dev/hdg</code>	<code>/cdrom</code>	<code>iso9660</code>	<code>defaults,noauto</code>	<code>0</code>	<code>0</code>

Trecho de Código 4.4.1: Exemplo de `/etc/fstab`

A configuração do `/etc/fstab` é formada por seis campos, separados por espaços, e cada linha deve incluir um dispositivo a ser montado/desmontado. Os campos são:

1. **Dispositivo:** o dispositivo a ser montado;
2. **Ponto de montagem:** o local aonde o dispositivo será montado;
3. **Tipo de sistema de arquivos:** aqui, diferentemente do caso de um comando `mount`, é *obrigatório* indicar o tipo de sistema de arquivos para que o `/etc/fstab` possa realizar adequadamente sua função;

4. **Opções:** essas opções são em geral específicas a um determinado sistema de arquivos e podem ser encontradas na *manpage* `mount(1)`. Porém, existem algumas que são comuns a todas e extremamente úteis dentro do `/etc/fstab`:

- **defaults** — valores padrões de montagem para o tipo de sistema de arquivo;
- **noauto** — *não monta* o dispositivo automaticamente na inicialização. Essa opção é muito interessante, por exemplo, para mídias de *backup* e meio que oferece um atalho para a montagem do dispositivo *a posteriori* (para montar um dispositivo listado em `/etc/fstab`, basta o comando `mount <ponto_montagem>`);
- **ro** — monta o dispositivo como apenas leitura. Pode ser útil, por exemplo, para um sistema de arquivos contendo programas (o `root` poderia criar um outro ponto de montagem aonde a montagem permitisse escrita para ele usar quando precisasse instalar um programa);
- **user** — permite que usuários comuns montem/desmontem o dispositivo. Embora o Guia FOCA GNU/Linux (SILVA, 2005a) afirmar que não é bom por motivo de segurança (e o autor concordar), é interessante que alguns dispositivos, principalmente em sistemas *desktop* ou em *workstations*, possam ser montados pelo usuário para *backup* pessoal e afins. O ideal é que *apenas* dispositivos removíveis, como CD-ROMs e disquetes possam ser montados pelos usuários;
- **sync** — força a sincronização dos dados (gravação dos dados no meio físico) após cada operação de escrita. Muito útil para dispositivos removíveis (evitando corrupção de dados), mas com o inconveniente de degradar a *performance* do sistema. Caso não use-se essa opção, pode-se ocasionalmente forçar a sincronização com o uso do comando **sync** (na prática, o próprio sistema faz isso, em média, a cada minuto);
- **noexec** — impede que qualquer arquivo dentro do dispositivo sejam tratados como executáveis, inclusive *scripts*. Embora isso possa ser inconveniente por tolher a possibilidade de uso de *shell scripts*, em certos sistemas de arquivos, como `/tmp` e `/etc`, aonde não são esperado executáveis, impede uma potencial e séria falha

de segurança, aonde esses diretórios são utilizados como depósito para *rootshells* e *backdoors*³

- **nosuid** — um pouco menos radical que **noexec**, impede que programas dentro do dispositivo possam ser acionados com *suid* ativo. Isso pode ainda dar chance para o acesso ilegal, mas, ao impedir a execução de programas daquele sistema de arquivos em modo *suid*, já impede **rootshells**;
5. **dump**: Configura a frequência dos *backups* do dispositivo por meio do utilitário **dump**. Em geral, mantém-se em 0 (desativado), pois o *dump* consome muito espaço em disco e é muito lento comparado com outras estratégias de *backup*;
 6. **Seqüência de fsck**: No caso de uma queda de energia ou falha de sistema, esse número define a prioridade no uso do utilitário **fsck** para verificação da integridade dos sistemas de arquivo. Se estiver zerado, o dispositivo em questão não é checado. O ideal é que o dispositivo com o sistema de arquivos (/) seja o primeiro a ser verificado sempre. A ordem é do menor para o maior;

Isso deve bastar sobre */etc/fstab*, o que encerra esse nosso tópico sobre dispositivos. No próximo tópico, vamos falar um pouco sobre a administração de um sistema GNU/Linux.

³Programas que liberam *shells* de super-usuário para um invasor em potencial

Capítulo 5

Administrando o sistema

O sistema GNU/Linux é um sistema operacional multi-usuário e multi-tarefa, o que torna muito importante e ocasionalmente complexa a sua administração. O objetivo desse tópico é que você conheça o básico da administração de um sistema GNU/Linux, basicamente operando em três frentes:

- Administração de usuários: criar, modificar as senhas e eliminar usuários e grupos;
- Administração de processos: entender o que são processos, identificar processos no sistema e eliminar processos estranhos;
- *Backup*: Entender a importância de um *backup*, conhecer os utilitários envolvidos no *backup* do sistema, criar e recuperar um *backup*;

Essas operações são todas feitas como super-usuário(**root**), portanto toda cautela é pouca: basta um erro para causar instabilidade no sistema ou colocar todo o seu sistema fora do ar. Por isso, estude esse capítulo com calma que tudo dará certo.

Esse capítulo deve ser enxergado como uma introdução à administração do GNU/Linux. Para aqueles que desejam se aprofundar no assunto, o conselho (após a leitura dessa apostila) é estudar o Guia FOCA GNU/Linux, Nível Avançado(SILVA, 2005b), ou o *The Linux System Administrators' Guide*(WIRZENIUS *et al.*, 2005b), além de poder-se estudar algum dos diversos livros disponíveis em boas livrarias técnicas.

5.1 Criando usuários: `adduser`

Como já cansamos de dizer nesse documento, o GNU/Linux é um sistema multi-usuário, o que quer dizer que você precisa de um nome de usuário para entrar nele, como dissemos no Capítulo 2, quando comentamos a filosofia do GNU/Linux (Seção 2.1.1, Página 18) e quando comentamos o processo de *login* no sistema (Seção 2.1.2, Página 19). Naquele caso, você provavelmente deve ter usado ou a conta `root` (tomando os devidos cuidados) ou uma conta de usuário especialmente criada pelo administrador do sistema¹. Agora que conhecemos todas as questões envolvidas com o sistema de permissões de acesso e afins e como os usuários são afetados por isso, podemos agora ver o processo envolvido com a criação de uma conta de um usuário.

Ao criar uma conta de usuário no GNU/Linux, basicamente faz-se quatro coisas:

1. Dá-se um nome de usuário ao mesmo, associando um UID (*User Identification*) numérico ao mesmo a ser usado pelo sistema internamente;
2. Estabelece-se uma senha para acessar-se o sistema com a conta de usuário em questão;
3. Associa-se o usuário a um grupo de usuários (podendo ser inclusive — e esse é o padrão do GNU/Linux — um grupo do “eu sozinho”, ao qual apenas o usuário faz parte), identificado através de um UID (*Group Identification*) numérico;
4. Cria-se e associa-se ao usuário um diretório de uso pessoal, o diretório *home*, já comentado Capítulo 3, na Seção 3.1.3, Página 67;

Esse processo pode ser feito de várias maneiras, inclusive pela edição manual de arquivos de configuração e cópias de arquivos necessários (demonstrada no Capítulo 11 do *The Linux System Administrators' Guide* (WIRZENIUS *et al.*, 2005a)), mas o GNU/Linux possui um comando que facilita a vida do administrador, o comando `adduser`. Ele realiza esse processo de maneira automatizada para o administrador, com as opções corretas selecionadas. O comando padrão é `adduser <usuario>`, mas esse comando em geral só cria a conta do usuário sem senha e sem grupo ou *home*

¹para um curso como esse, seria aconselhável o usuário `guest`, senha `guest`

associado (algumas distros utilizam *alias* aqui para que faça algumas dessas funções de maneira mais automática). Por isso, é importante saber quais as melhores opções a serem usadas.

No caso, vamos ver as opções mais úteis:

- **-disable-password:** não pede uma senha de imediato, pedindo no momento em que o usuário se logar pela primeira vez no sistema;
- **-uid:** Define um UID específico ao usuário, ao invés de simplesmente usar o próximo UID disponível no sistema. Isso pode ser muito interessante quando você quer que um determinado usuário reaproveite os arquivos de outro, mas deve-se ter cuidado com isso, pois no caso de UIDs iguais as permissões dos dois usuários serão iguais (o GNU/Linux usa os UID para as suas funções, e os nomes de usuário são apenas uma referência fácil que a pessoa tem para saber informações sobre o uso do sistema). Veremos mais sobre isso na Seção 5.1.1, Página 92;
- **-gid:** Especifica um GID arbitrário ao novo grupo do usuário. Como no caso do UID, deve-se tomar cuidado, pois dois GIDs iguais são considerados iguais no sistema, mesmo que os nomes dos grupos sejam diferentes;
- **-home:** Determina o diretório *home* do usuário. Muitas vezes, utiliza-se o padrão `/home/<usuario>` como diretório *home* do usuário, mas essa opção permite definir arbitrariamente o *home* do usuário. Isso pode ser muito útil em sistemas cujo sistema de arquivos² `/home` esteja lotado ou em níveis críticos, podendo permitir que a conta seja criada em um sistema de arquivos menos ocupado;
- **-p:** define uma senha padrão para o usuário. Perceba que essa opção não pode ser usada em conjunto com **-disable-password**;
- **-s:** define o *shell* do usuário. Uma coisa muito interessante é que você pode definir como *shell* do usuário *qualquer* programa em absoluto. Ou seja, se você quiser que um determinado usuário acesse apenas o programa **workprog**, você pode definir ele como o *shell* do mesmo. Ele

²dentro do mundo GNU/Linux, algumas vezes os termos *dispositivo* e *zemp* sistema de arquivos podem ser usados de maneira intercambiáveis

poderá acessar apenas aquele programa e, ao sair do mesmo, será levado de volta ao *login*. Você pode até mesmo definir que o usuário *não terá um shell*. Isso é útil para servidores, criando ilhas de segurança através de usuários sem *shell*. Para isso, basta apontar para `/dev/null` ou para os *null shells* (*shells* nulos) `/bin/true` e `/bin/false`. Na maioria dos sistemas GNU/Linux, o *shell* padrão é o `/bin/bash` (ou seu *link* simbólico `/bin/sh`);

Para mais opções, consulte a *manpage* `adduser(8)`.

Vamos a um exemplo:

Imaginemos que queremos criar um usuário `adumbledore`, com o *home* em `/hogwarts/principal` e com a senha `allflavorbeans`. O *shell* será o padrão do sistema, e aceitaremos o novo UID e GID. Para isso, utilizamos o comando `adduser -p allflavorbeans -home /hogwarts/principal adumbledore`. Toda vez que alguém digitar `adumbledore`, irá listar o conteúdo de `/hogwarts/principal` e para entrar como o usuário `adumbledore` será necessário a senha `allflavorbeans`.

5.1.1 O cuidado com os UID

Uma coisa muito importante a tomar-se cuidado é com a questão dos UID. O motivo desse é que, citando por Wirzenius *et al.* (2005a), “*Na realidade, o kernel do Linux trata os usuários como simples números*”: o uso de nomes de usuário em listagens de arquivos e no *login* são apenas meras conveniências do sistema para o acesso facilitado a informações pelo usuário. Por isso, o UID deve ser um identificador único.

O principal erro (e que pode ser *muito* grave) é usar o mesmo UID para dois usuários. Nesse caso, os dois usuários terão a mesma permissão de acesso ao sistema. Isso é uma brecha de segurança séria, e um dos principais modos de se criar um *rootshell* é esse: criar um usuário comum, “inofensivo” e, adulterando os arquivos de configuração, definir seu UID para 0 (UID do `root`).

Reaproveitar UIDs de usuários removidos do sistema pode não ser também uma boa idéia, pois o novo usuário passará a ser dono de todos os arquivos do antigo dono. Mas isso pode também ser interessante: imagine

que um determinado usuário saiu da companhia e outro entrou em seu lugar. Ao invés de ter um trabalho enorme mudando permissões e configurando grupos, você pode simplesmente definir o UID do usuário em questão para o do antigo usuário.

Por exemplo: imaginemos que nosso usuário `adumbledore` tenha sido removido, e que criaremos um usuário `mmcgonagall`, que terá acesso aos arquivos e dados manipulados por `adumbledore`. Imaginemos que o UID de `adumbledore` seja 1024 e que o usuário `mmcgonagall` terá como senha `animagus`. Podemos usar o comando `adduser -p animagus -home /hogwarts/principal -uid 1024 mmcgonagall`. Isso *automagicamente* tornaria `mmcgonagall` o dono dos arquivos, diretórios, programas e do diretório *home* do antigo usuário `adumbledore`. Esse, inclusive, pode ser um truque interessante para impedir que ex-funcionários insatisfeitos roubem as informações com que trabalhavam na empresa.

5.1.2 O diretório `/etc/skel`

Como dissemos na Seção 3.1.3, Página 67, quando falamos sobre o diretório *home* dos usuários, uma das principais funções do diretório *home* é guardar os arquivos de configuração pessoal dos programas. Isso é muito vantajoso para oferecer ao usuário uma configuração pessoal condizente com as suas necessidades sem afetar os demais usuários do sistema.

Porém, essa configuração é um pouco chata, e pode ser útil que certos padrões, como papéis de parede para ambientes gráficos, páginas iniciais e parâmetros de configuração do ambiente do *shell* possam ser definidos em certos padrões. Para isso, existe a possibilidade de copiar-se um diretório de referência como o *home* a ser definido para o usuário, depois trocando-se o dono e grupo do arquivo para o do usuário. Esse processo, porém, torna a coisa toda maçante para o administrador.

O ponto bom é que o GNU/Linux, como os Unix em geral, pensou nessa possibilidade e tornou possível criar-se um padrão para o diretório *home* dos usuários. Esse padrão é o diretório `/etc/skel` (*skel*, de *skeleton*, que pode ser traduzido como estrutura). No caso, tudo o que o administrador tem a fazer é, depois de ter um padrão para os novos usuários configurado, copiar todos os arquivos do *home* aonde o padrão foi criado para dentro de `/etc/skel` e

remover quaisquer arquivos de uso pessoal do usuário em questão de dentro desse diretório. O próprio `adduser` irá copiar para dentro do *home* do novo usuário todos os arquivos e diretórios dentro do `/etc/skel` e passar a posse dos mesmos para o novo usuário, sem muita dificuldade.

A posse dos arquivos e do diretório `/etc/skel` é do `root`. Embora o diretório permita que o usuário o liste, *não é* uma boa idéia a cópia direta pelo usuário do diretório `/etc/skel` ou de partes dele, primeiro pela possibilidade de perder-se todas as configurações do ambiente, em segundo pois isso irá complicar as coisas, pois o `root` terá que passar manualmente a posse dos arquivos copiados...

Mas mesmo assim, o administrador conseguirá sempre fazer bom uso de `/etc/skel`.

5.1.3 O arquivo `/etc/passwd` e o arquivo `/etc/shadow`

Como dissemos anteriormente, uma vez que você usa o comando `adduser` para criar uma conta de usuário, você envia informações para um arquivo de configuração. Esse arquivo é o arquivo `/etc/passwd`. Ele é um arquivo de texto simples, que pode (mas não deveria) ser editado manualmente com o uso de editores como `vi` ou `emacs`. Um arquivo `/etc/passwd` típico se parece com o mostrado no Trecho de Código 5.1.1, na página 95.

Cada linha é referente a um determinado usuário, e é composta por:

```
user:password:uid:gid:gecos:home:shell
```

O campo `user` é bem claro: é o nome do usuário. Perceba que o primeiro deles é o `root`, e assim por diante. Alguns serviços, como o MPD³ e o Subversion⁴ criam usuários especiais para si, como `mpd` e `svn`. Esses usuários são apenas para que os processos deles não ofereçam riscos no caso de exploração de falhas.

³*Music Player Daemon* — reprodutor de música que pode atuar como servidor *Shout-cast*

⁴sistema de SCM (*Source Control Management* — Controle de Código Fonte)

```

root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/sh
daemon:x:2:2:daemon:/sbin:/bin/sh
adm:x:3:4:adm:/var/adm:/bin/sh
...
fecosta:x:500:500:FÃbio Emilio Costa:/home/fecosta:/bin/bash
mpd:x:84:81:system user for mpd:/var/lib/mpd:/bin/false
tomcat:x:91:91:Tomcat:/usr/share/tomcat5:/bin/sh
svn:x:92:92:system user for subversion:/var/lib/svn:/bin/false
postgres:x:93:93:system user for postgresql:/var/lib/pgsql:/bin/bash
webdav:x:501:501:/home/webdav:/bin/true
adumbledore:x:502:502:/home/adumbledore:/bin/bash
mmcgonagall:x:503:503:/home/mmcgonagall:/bin/bash
ssnape:x:504:504:/home/ssnape:/bin/bash
sblack:x:505:505:/home/sblack:/bin/bash
hslughorn:x:506:506:/home/hslughorn:/bin/bash

```

Trecho de Código 5.1.1: Exemplo de /etc/passwd

CAMPO password	COMENTÁRIOS
Seqüência de caracteres	senha criptografada
Nada	Sem senha para essa conta
*	A conta em questão está desativada
x	A senha dessa conta está gravada em outro lugar

Tabela 5.1: Valores típicos do campo password do /etc/passwd

O campo `password` é aonde iria normalmente as senhas, mas atualmente são poucas (se alguma) as instalações aonde as senhas do usuário são colocadas aí, pois o arquivo `/etc/passwd` tem que ser de leitura aberta a todos, o que tornava muito fácil roubar-se o arquivo e tentar-se quebrar as senhas por força bruta. Atualmente, utiliza-se o arquivo `/etc/shadow`, que pode ser travado para leitura para outros usuários fora o `root`. Esse campo pode conter as opções mostradas na Tabela 5.1.3, na Página 95.

Os campos `uid` e `gid` contêm o UID e o GID principal do usuário (o principal grupo ao qual o usuário faz parte) do usuário. Esses valores são numéricos.

O campo seguinte **GECOS** servia como uma espécie de registro de informações sobre o usuário, como telefone, local de serviço e afins. Nos antigos ambientes Unix, essa informação era usada pelo utilitário **finger**, e acabou sendo mantida por motivos históricos. Atualmente, ela serve apenas para registros diversos, sem função no sistema.

O campo **home** indica o diretório de *home* do usuário em questão. A estrutura normal é `/home/<usuario>`, mas não é obrigatória (no exemplo, o *home* do usuário **tomcat** é `/usr/share/tomcat5`). Alguns programas utilizam o diretório *home* como uma forma simples de apontar para diretórios de configuração dele.

O último campo, **shell**, define o *shell* do usuário. Perceba que existem casos de *null shells*, como os dos usuários **webdav**(`/bin/true`) e **svn**(`/bin/false`).

5.1.3.1 O arquivo `/etc/shadow`

Perceba que é razoavelmente simples o arquivo `/etc/passwd`. O resto do controle é feito no arquivo `/etc/shadow`, que tem a seguinte estrutura (segundo o *Linux Shadow Password HOWTO*(JACKSON, 1996)):

```
user:password:last:may:must:warn:expire:disable:reserved
```

Aonde **username** e **password** são equivalentes aos do `/etc/passwd`. No caso do campo **password** ele pode conter as opções mostradas na Tabela 5.1.3, na Página 95, exceto, obviamente, por **x**.

O campo **last** contém o dia em que a senha foi trocada pela última vez, no formato “Unix *timestamp*”⁵.

O campo **may** contém o número de dias que o usuário deve esperar antes de trocar sua senha por uma nova. Para evitar problemas, esse valor pode ser mantido baixo.

O campo **last** contém o número de dias que o usuário poderá usar essa senha antes de ser obrigado a trocá-la.

⁵Um formato aonde o valor é representado no número de dias passados do dia 1º de Janeiro de 1970

O campo `warn` contém o número de dias que o usuário deverá ser informado de que sua senha está para expirar, de modo a trocá-la.

O campo `expire` contém o número de dias após a senha ter vencido que deve-se esperar até que a conta seja desabilitada.

O campo `disable` indica o dia em que a conta foi desabilitada no formato “Unix *timestamp*”. E o campo `reserved` é reservado para uso futuro.

Portanto, não é muito complexo o estudo dos arquivos `/etc/passwd` e `/etc/shadow`. Perceba que falamos apenas de *estudo*.

Atenção: A manipulação errônea dos arquivos `/etc/passwd` e `/etc/shadow` podem possuir conseqüências desastrosas. Evite o máximo que possível manipular tais arquivos manualmente. Caso tenha que o fazer, faça uma cópia de *backup* de ambos os arquivos. *Você foi avisado!!!*

Com isso, encerramos o assunto usuários. Vamos falar agora dos grupos de usuários.

5.2 Criando grupos: `addgroup`

Os *grupos* de usuários são muito úteis para estruturar de maneira mais granular o acesso aos arquivos. Por exemplo, você pode criar um diretório aonde o dono do arquivo seja você, mas você permita que outros escrevam no diretório também (um *sticky bit* aqui pode ser muito útil para impedir outros de apagarem seus arquivos). Para isso, é importante que o usuário esteja cadastrado no grupo em questão.

Em geral, quando o usuário é criado, o padrão é que ele seja inserido em um grupo especial só para ele, que poderíamos chamar de “grupo do eu sozinho”, o que permite o isolamento dos arquivos pessoais. Mas algumas vezes isso pode ser inconveniente. Imagine, por exemplo, um projeto aonde vários usuários precisem dividir os mesmos arquivos. A opção se todos eles estiverem em “grupos do eu sozinho” seria o `root` ficar copiando os arquivos e mudando a posse dos arquivos para cada um dos participantes periodicamente. Isso ocasionaria todo o tipo de problemas.

Para evitar isso, o GNU/Linux, como os Unix em geral, possui o sistema de permissão por grupos. Se configurado corretamente, um grupo pode permitir que usuários diversos utilizem-se dos arquivos de outros usuários, inclusive os editando, sem maiores inconvenientes.

Para adicionar um novo grupo, utiliza-se o comando `addgroup`. Usam-se as mesmas opções do caso do `adduser` (Seção 5.1, Página 90), mas na realidade, nenhuma delas é útil (com a possível exceção de `-p`). Portanto, vamos tratar normalmente dos comandos.

Vamos imaginar que o usuário `adumbledore` deseje criar um grupo para sair do “grupo do eu sozinho”, então usa o comando `addgroup` para criar o grupo `orderphoenix`. Para isso, ele usa o comando `addgroup orderphoenix`.

5.2.1 O cuidado com os GID

Muito cuidado se quiser reaproveitar GIDs para novos grupos. Da mesma forma que no caso dos UIDs, os GIDs são fixados nos arquivos, o que pode fazer que usuários de um grupo com o GID reaproveitado possam acessar arquivos que não deveriam.

5.2.2 O arquivo `/etc/group`

As informações sobre os grupos são armazenadas no arquivo `/etc/group`, que o sistema consulta toda vez que precisa pesquisar acessos por grupo. No caso, a estrutura desse arquivo é:

```
NomedoGrupo:senha:GID:usuários
```

Aonde `NomedoGrupo` é o nome do grupo criado, `senha` é a senha para realizar-se operações como esse grupo em especial (que pode também estar em outro arquivo, o `/etc/gshadow`), e `GID` mostra o número do GID do grupo em questão.

A parte importante do arquivo fica logo em seguida, com o campo `usuários`, aonde ficam os usuários que fazem parte desse grupo. O Trecho de Código 5.2.1, na página 99, mostra um exemplo de `/etc/group`.

```
root:x:0:
bin:x:1:
daemon:x:2:messagebus
sys:x:3:
adm:x:4:
(...)
audio:x:81:
video:x:82:
users:x:100:
nogroup:x:65534:
(...)
mysql:x:79:
gdm:x:422:
canna:x:423:
haclient:x:60:
squid:x:424:
sshd:x:83:
slocate:x:425:
fecosta:x:500:
(...)
adumbledore:x:502:
mmcgonagall:x:503:
ssnape:x:504:
sblack:x:505:
hslughorn:x:506:
hogwarts:x:507:adumbledore,mmcgonagall,ssnape,hagrid
orderphoenix:x:509:adumbledore
aurors:x:510:amooddy
```

Trecho de Código 5.2.1: Exemplo de /etc/group

Perceba que a maioria dos grupos são “grupos do eu sozinho” e não mostram outros usuários: isso porque o GNU/Linux, quando um grupo tem o mesmo nome do usuário, ele é considerado um “grupo do eu sozinho”. Perceba também que, da mesma forma que o `/etc/passwd`, ele *não possui ordem, exceto pela a de entradas no registro*.

Porém, perceba, o grupo `orderphoenix`: ele está com senha em `/etc/gshadow` e tem como participante o usuário `adumbledore`. No caso, seu “dono” (o usuário `adumbledore`) deseja incluir os usuários `mmcgonagall` (que tem como grupo principal `hogwarts`), `hhagrid` (que tem como grupo principal `hogwarts`) e `amoody` (que tem como grupo principal `aurors`). Para isso, ele pede ao `root` que edite o `/etc/group` para incluir esses três usuários. O `/etc/group` resultante fica como o do Trecho de Código 5.2.2, na página 100.

```
root:x:0:
bin:x:1:
daemon:x:2:messagebus
sys:x:3:
adm:x:4:
(...)
adumbledore:x:502:
mmcgonagall:x:503:
ssnape:x:504:
sblack:x:505:
hslughorn:x:506:
hogwarts:x:507:adumbledore,mmcgonagall,ssnape,hagrid
orderphoenix:x:509:adumbledore,mmcgonagall,hagrid,amoody
aurors:x:510:amoody
```

Trecho de Código 5.2.2: Exemplo de alteração em `/etc/group`

5.2.3 Usando `adduser` para adicionar um usuário a um grupo extra

Existe uma outra forma de adicionar-se usuários a um grupo, que é usando o comando `adduser`. Para isso, usa-se `adduser <grupo> <usuario>`, de modo que nesse caso não será criado um novo usuário com o mesmo nome, mas sim

o sistema irá adicionar o usuário ao novo grupo (um usuário para um grupo por vez).

Retornando ao nosso exemplo: a seguir, ele decide incluir os usuários **sblack** (que tem como grupo principal **marauders**), **rlupin** (que tem como grupo principal **marauders**) e **ssnape** (que tem como grupo principal **deatheaters**). Para isso, ele (na verdade, o **root**, mas se o **root** quiser — *não é recomendável* — ele pode deixar o comando **adduser suid**) utiliza a combinação de comandos mostradas no Trecho de Código 5.2.3, na página 101, tendo o resultado listado na mesma listagem.

```
[root@hufflepuff CursoGNULinux]# adduser orderphoenix sblack
[root@hufflepuff CursoGNULinux]# adduser orderphoenix rlupin
[root@hufflepuff CursoGNULinux]# adduser orderphoenix ssnape
[root@hufflepuff CursoGNULinux]# cat /etc/group
root:x:0:
bin:x:1:
daemon:x:2:messagebus
sys:x:3:
adm:x:4:
(...)
deatheaters:x:3000:ssnape,lmalfoy,wmcnair,ppetigrew
marauders:x:3000:jpotter,sblack,ppetigrew,rlupin
adumbledore:x:502:
mmcgonagall:x:503:
ssnape:x:504:
sblack:x:505:
hslughorn:x:506:
hogwarts:x:507:adumbledore,mmcgonagall,ssnape,hhagrid
orderphoenix:x:509:adumbledore,mmcgonagall,hhagrid,amoody
aurors:x:510:amoody
```

Trecho de Código 5.2.3: Exemplo do uso do comando **adduser** para adicionar usuários a outros grupos, assim como o resultado desse uso

Isso deve bastar sobre adicionar usuários nos grupos. Vejamos agora sobre como usar a questão de grupos a seu favor.

5.2.4 Acessando documentos com outra identificação de grupo: `sg`, `newgrp` e `id`

Em geral, uma vez que você tenha entrado em um outro grupo, você pode acessar todos os arquivos daquele grupo normalmente. Em alguns casos, porém, você precisa que a *identificação de grupo* sua seja alterada para aquele procedimento. Existem duas formas de se fazer isso:

1. usando `setgid` no arquivo/programa a ser manipulado;
2. usando o comando `sg` (*substitute group*);

O `sg` lembra um pouco o `su`, mas ele vale para apenas um comando. A forma de uso é `sg <grupo> '<comando>'`. Ele irá pedir a senha do grupo: se esta estiver incorreta ou não tiver sido definida, você não poderá realizar a operação.

Qual a vantagem do uso `sg`? Normalmente, você está sempre como se você fosse do seu “grupo do eu sozinho”. É possível alterar permanentemente o grupo principal, manipulando o arquivo `/etc/passwd` e modificando manualmente o GID para o do grupo desejado. Porém, isso não é recomendável (e muitas vezes nem possível, uma vez que apenas o `root` pode escrever no `/etc/passwd`). Nesse caso, utiliza-se o `sg` para “alternar-se” para outro grupo e poder manipular o arquivo em questão.

Por exemplo: imaginemos que o usuário `adumbledore` esteja querendo ler e fazer comentários no arquivo `voldieplans` do usuário `ssnape`. Esse arquivo tem como dono `ssnape`, grupo designado `orderphoenix` (ao qual `adumbledore`) também pertence) e permissões `rw-rw---` (o que quer dizer que apenas o dono e o grupo sabe da existência do arquivo). Como `adumbledore` sabe da existência do arquivo, ele apenas digita o comando `sg 'emacs voldieplans'`⁶. Ele passa então a editar o arquivo normalmente, como se tivesse como grupo principal `orderphoenix`. Assim que ele terminar, ele volta ao normal, ao seu “grupo do eu sozinho”.

É possível alterar-se durante uma sessão o grupo efetivo de uso, através do comando `newgrp <grupo>`. Nesse caso, o grupo torna-se efetivo até que

⁶Alvo Dumbledore usa EMACS, pessoal!! E o autor também! :P

o usuário digite `exit` ou `newgrp -`. De outra forma ele opera exatamente da mesma forma que o comando `sg`.

Como final, o comando `id` mostra as informações do usuário atual (pode-se usar `id <usuario>` para ver as de outro usuário). Nesse caso, mostra-se tanto o UID/GID real quanto o efetivo (que pode ser mudado por `su`, `sg` ou `newgrp`).

5.2.5 Alterando senhas: o comando `passwd`

Algumas vezes, precisamos mudar a senha de um usuário (ou o próprio usuário mudar sua senha). Para isso, usa-se o comando `passwd`. Esse comando abre uma janela simples aonde pede-se a senha antiga, a nova senha e uma confirmação da nova senha. Em versões mais atuais, o `passwd` utiliza o utilitário `crack` para checar a força da senha. Caso não seja forte o suficiente, alerta o usuário para que ele forneça uma senha mais forte⁷. Apenas o dono de uma conta ou o `root` pode alterar a senha da conta. Para alterar-se uma senha, usa-se `passwd <usuario>`. Se `<usuario>` não for fornecido, admite-se o usuário atual como o que terá a senha trocada.

Também é possível usar-se `passwd` para trocar a senha de um grupo. Essa operação só pode ser feita pelo `root` ou pelo administrador do grupo (o primeiro usuário listado na lista de usuários do grupo, ou o próprio usuário, no caso de grupos “do eu sozinho”). Para isso, usa-se a sintaxe `passwd -g <grupo>`.

O comando `passwd` possui mais uma grande quantidade de opções de administração que não veremos aqui, pois são específicas para o `root`, podendo causar bloqueio de contas e configurar parâmetros de validade e alternância de senhas. Para isso, consulte a *manpage* `passwd(1)`. Se quiser saber mais sobre administração de grupos, inclusive bloqueio de grupos, consulte a *manpage* do utilitário `gpasswd(1)`.

⁷ou que se conforme se algo acontecer de errado no futuro :P

5.2.6 Removendo usuários e grupos: `userdel` e `groupdel`

Algumas vezes, precisamos remover usuários e grupos por vários motivos, seja pela demissão do usuário ou apenas pelo cancelamento de seus projetos. O ideal é que você não remova contas, a não ser que os usuários em questão tenham deixado a instituição ou perdido o direito de uso ao sistema. Nesse caso, a sugestão é:

1. Realize *backup* dos arquivos pessoais do usuário para qualquer eventualidade. Veremos mais sobre isso na Seção 5.4, 120;
2. Desabilite a conta do usuário (ela ainda estará lá, mas na prática o usuário não conseguirá mais se logar;
3. Apague qualquer arquivo ou diretório desnecessário e troque a posse dos demais arquivos e diretórios. Isso atua tanto para salvar espaço em disco quanto para impedir falhas de segurança;
4. Elimine a conta do usuário;

A primeira parte (*backup*) é simples e envolve pouco esforço. O segundo passo pode ser feito de duas formas:

1. Editando manualmente o arquivo `/etc/passwd` (ou `/etc/shadow`) e no campo `password` deles e colocando o símbolo `*` na frente da senha criptografada (esse procedimento *não é recomendável*, uma vez que um erro na edição manual desses arquivos pode por todo o sistema a pique);
2. Usando o comando `passwd -l <usuário>` para desabilitar a conta (recomendável);

Esse último passo pode ser feito como uma segurança, por exemplo, caso o usuário em questão saia de férias.

A terceira parte envolve os comandos que já estudamos no Capítulo 2, na Página 17, como o `cp`, `rm` e `chown`. O *backup* dos arquivos antes é para permitir que arquivos acidentalmente deletados possam ser recuperados em caso de necessidade.

A última etapa é a remoção propriamente dita do usuário, com o uso do comando `userdel`, que remove todas as entradas do usuário em arquivos de configuração (`/etc/passwd`, `/etc/shadow`, `/etc/group`, ...), sendo que sua sintaxe normal é `userdel <usuario>`. Uma opção útil é a opção `-r`, que remove os arquivos do usuário (incluindo seu *home*), o que permite poupar algumas etapas (basta copiar os arquivos importantes daquele usuário para outro diretório).

Para remover um grupo, você irá utilizar o comando `groupdel`. Esse comando remove a entrada do grupo em `/etc/group`. No caso, ele tem uma “pegadinha”: você nunca poderá remover um grupo primário de um usuário (em geral, os “grupos do eu sozinho”) sem antes remover o usuário em questão. O uso de `groupdel` é `groupdel <usuario>`.

Todos esses comandos foram oferecidos apenas como uma introdução à administração do usuário. Existem muitos outros comandos úteis, como `usermod(8)`, `groupmod(8)`, `chfn(1)` e `chsh(1)`, mas não trataremos deles aqui: consulte as *manpages* deles para maiores informações.

Agora que terminamos o tópico usuários, vamos passar para o tópico de administração dos programas em execução. Ou como chamamos em Unix (e GNU/Linux), de *administração de processos*.

5.3 Entendendo processos e *jobs*

No GNU/Linux, como no Unix, todos os programas do usuário são chamados de *processos*. Na prática, todo programa que está carregado é um *processo*. O objetivo dessa seção é que você compreenda o funcionamento dos processos e entenda como eles funcionam, consiga identificar e solucionar problemas com processos.

5.3.1 O que é um processo

No GNU/Linux, quando uma cópia de um programa é carregada para memória, certas alocações de recursos como memória, discos, portas TCP, ... são realizadas pelo sistema operacional, através de programas conhecidos como

*loaders*⁸. Em geral, o *loader* é embutido no programa, portanto chamar o programa já automaticamente dispara o *loader* para que o mesmo seja carregado na memória. Uma vez que o *loader* carregue todas as informações do programa para que ele possa ser executado, ele envia informações sobre o programa carregado na memória para um trecho de código do *kernel* chamado *escalonador de processos*, que é responsável pelo controle do tempo de execução dos processos. Isso porque, embora falemos que o GNU/Linux é multi-tarefa, na prática, exceto em sistemas multiprocessados, o sistema executa *uma tarefa por vez*. O escalonador de processos passa então a controlar quanto tempo o processo está trabalhando, para que, após algum tempo, o controle do sistema seja passado para outro programa.

Quando o *loader* passa as informações do sistema para o escalonador de processos, o escalonador “cria” um processo para o programa carregado⁹, sendo que o mesmo recebe um PID (*process ID* — identificador de processo), dado pelo escalonador, que também passa a manter certos registros de contabilidade do acesso do processo aos recursos do sistema, assim como informações sobre o usuário e grupo *efetivos* que disparou o programa¹⁰ e *grau de prioridade* do processo no sistema. A partir de todas essas informações, principalmente no grau de prioridade (ou *prioridade*, para resumir), o escalonador vai determinando como os processos usarão os recursos do sistema.

5.3.1.1 Processos de primeiro e segundo plano

Normalmente, em todo sistema multitarefa, existem milhares de processos em execução. O sistema precisa decidir, portanto, como ele irá proceder com cada um. Mas isso em geral é transparente ao usuário, para o qual apenas importa em qual programa ele está trabalhando no momento.

O GNU/Linux utiliza, como outros sistemas operacionais multitarefa, o conceito de *primeiro plano*(*foreground*) e *segundo plano*(*background*).

⁸Isso apenas no primeiro momento. Depois dele, a carga de recursos em tempo real é responsabilidade do próprio programa

⁹Na prática, quem cria o processo é o *loader*, mas essa pequena alteração pode ser útil para a compreensão da idéia de processo

¹⁰lembrando que essas informações podem ser alteradas por permissões e comandos especiais, como *suid* e *sg*

Um processo em geral é considerado de *primeiro plano* quando o usuário está o manipulando *interativamente* e espera receber a saída do mesmo. Por exemplo: um comando `ls -la` é um processo de primeiro plano, pois o usuário precisará receber a saída do comando, assim como quando um usuário está usando o `emacs`, por exemplo. Só pode haver *um* processo em primeiro plano por usuário.

Processos em *segundo plano* são processos cujo resultados o usuário não precisa de imediato. Em geral, servidores no GNU/Linux são executados em *segundo plano*, pois os resultados dos acessos a ele podem ser acompanhados via *logs* do sistema. Um programa é carregado em segundo-plano no GNU/Linux quando usa-se diante dele o símbolo `&`(e comercial).

Atenção: O fato de que um processo foi enviado para segundo plano *não quer dizer que ele não irá devolver nenhuma saída*. Para evitar tornar o seu *shell* uma bagunça, é interessante que a saída de um programa em segundo plano seja redirecionada para um arquivo;

Atenção: No GNU/Linux, programas gráficos estão *todos* em segundo plano. Apenas o servidor X-Windows fica em primeiro plano. É importante essa distinção, pois ao invocar um programa gráfico você pode travar o seu *shell*, não mais o podendo utilizar. Veremos mais sobre isso quando falarmos de Interfaces Gráficas no Capítulo 8 (Página 214).

A principal vantagem do uso de processos de segundo plano é o fato de que você pode automatizar comandos e não precisar ficar esperando ou operando interativamente. Combinando redirecionamento e *pipes* (que falamos no Capítulo 2, na Seção 2.8.2.1, Página 59) com o uso de programas em segundo plano, você pode automatizar uma série de tarefas administrativas.

5.3.2 Diferença entre programas multi-thread e programas de múltiplos processos

O GNU/Linux, como muitos servidores, além de ser multitarefa, aceita aplicações *multithreaded*. Os programas dotados de *multithread* executam partes internas dele em paralelo, da mesma forma que o SO multitarefa executa

muitas tarefa em paralelo. Para isso, em alguns casos, utiliza-se tanto de um escalonador interno ao processo quanto, no caso do GNU/Linux, bibliotecas que fazem com que o escalonador de processos considere as *threads*¹¹ dentro do sistema.

Muitos servidores antigos, como o Apache até a versão 2.0 e o PostgreSQL são servidores multi-processos: o servidor que é carregado pelo sistema tem como função apenas gerar outros processos que realizem a requisição efetuada. Isso causa um certo *overhead* pois o sistema tem que ser interrompido para que o novo processo seja gerado (*process spawn*) e para que recursos sejam alocados ao mesmo. Além disso, em alguns casos, o processo gerador fica interrompido, impedindo que ele gere novos processos para atender novas requisições. Em geral isso ocorre quando um determinado número de processos-filhos foram gerados. Uma terceira dificuldade é que, como cada processo-filho possui seu próprio espaço de memória alocada e recursos, isso pode aumentar *rapidamente* o consumo da memória e dos recursos computacionais do sistema, além de acarretar mais processamento por parte do escalonador de processos.

Por sua vez, processo *multithread* atendem suas requisições gerando novas *threads* dentro deles. Isso tem um menor *overhead* do que o *process spawn*, além de impedir que o processo fique travado ou que o escalonador de processos tenha mais o que processar (mesmo quando o processamento das *threads* é feito pelo escalonador do sistema, isso permite que várias *threads* possam ser atendidas quando uma ou mais delas necessitem ou tenham tempo de processamento disponível). Em compensação, é muito fácil para um processo *multi-thread* mal-implementado gerar uma *race condition*¹² ou *deadlocks*¹³ do que em programas multi-processos, pois em geral os recursos entre processos não são disputados (tem ou não tem disponível).

Em que isso afeta a administração do sistema? Quando você tem um servidor *multithread* você possui apenas um processo desse servidor, enquanto no caso de servidores multiprocessso você tem vários processos. Veremos

¹¹uma *thread* é parte do processo pai, mas embora rode em paralelo em relação a outras *threads*, compartilha o acesso a memória e outros recursos com as demais *threads*.

¹²condição de disputa na qual dois processos/*threads* necessitam de determinado recurso

¹³situação em que um programa A exige um recurso que um programa B está alocando, que por sua vez exige um recurso que um programa C está alocando, que por sua vez exige um recurso que um programa A está alocando, e todos os três estão esperando que os recursos desejados estejam disponíveis para liberar os recursos que estão usando

melhor isso quando usarmos o comando `ps` para vermos os acontecimentos dentro do sistema (Seção 5.3.4, Página 110).

5.3.3 Estado de processos

Por causa do escalonamento, existem vários processos ao mesmo tempo “em execução” no sistema (lembrando que o escalonador de processo libera apenas um processo por vez a ser executado por alguns instantes, passando a execução para outro logo em seguida). Portanto, cada um dos vários processos no sistema possui um estado diferente. Basicamente existem quatro estados no qual um processo pode estar:

- **Processo em primeiro plano (*foreground*):** são processos que estão bloqueando o terminal. Normalmente são processos que exigem interatividade, como ler *email* ou digitar um texto no `vi`;
- **Processo em segundo plano (*background*):** são processo que, embora estejam recebendo e enviando dados, ele não exige interatividade de nenhum tipo. Em geral, os servidores (ou *daemons*) do sistema contam como exemplo de processos em segundo plano, assim como os programas rodando sobre uma interface gráfica;
- **Processos parados:** são processos que estão paralizados por algum motivo, mas que voltarão a ser executado. Um exemplo é o processo `init`: ele inicializa todo o sistema, realizando uma série de configurações, que por sua vez culminam com o disparo de processos `login`, quando então ele vai para o estado parado. Por sua vez, `login` também é normalmente um processo parado: uma vez que um *login* tenha sido feito de maneira bem sucedida, o processo `login` gera um processo do *shell* apropriado ao usuário e fica parado, voltando a ser executado quando o usuário abandona aquele *shell*, quando `login` espera um novo *login* bem sucedido, que irá gerar um novo processo *shell* e assim *ad infinitum*. Ao processo que fica parado após gerar um processo é chamado *processo pai* e ao que foi gerado, *processo filho*;
- **Processos zumbis:** Algumas vezes, um processo pai “morre” (ou seja, é encerrado) antes de um processo filho. O normal é que o processo

pai, ao “morrer”, “mate” também os processos filhos. Também é responsabilidade do processo pai desalocar os recursos usados pelo processo filho. Pode acontecer, porém, de o processo filho não “morrer” por algum motivo. Nesse caso, pode acontecer do processo filho, ao “morrer”, tentar devolver recursos ao processo pai (“morto”) e não o conseguir. Nesse caso, o processo filho simplesmente “congela”, virando um processo zumbi. O problema desses processos é que eles amarram consigo recursos preciosos para o sistema. É possível forçar um processo a ficar ativo mesmo depois do seu pai ter “morrido”, mas sem correr o risco de torná-lo zumbi, por meio do comando `nohup` (Seção 5.3.9, Página 119). Outro motivo para um processo virá zumbi pode acontecer se *“um programa cria um processo e demora para consultar seu resultado após seu término, o processo permanece como ‘zumbi’. Geralmente, é um bug do programa pai do processo. Se existirem muitos processos ‘zumbi’, pode ser necessário terminar o programa pai para desocupar a tabela de processos do kernel.”*(ZAGO, 2006);

É muito importante sabermos distinguir como os processos estão situados no sistema, principalmente por causa dos zumbis. Para isso, precisamos saber como vistoriar o sistema atrás de processos. O comando para isso é o `ps`.

5.3.4 O comando `ps`

O comando `ps` (*process scan* — verificação de processos) permite ao usuário investigar quais processos estão em que situação. Para isso, basta usar o comando `ps`. É muito interessante que a saída do `ps` seja redirecionada para o `less` via *pipe*.

O normal é apenas listar-se os processos do usuário, mas o `ps` possui algumas opções interessantes, retirada de Silva (2005a). Perceba que as opções no `ps` não levam `-`:

- `a`: mostra os processos em execução de todos os usuários;
- `x`: mostra processos que não são controlados pelo terminal;
- `u`: mostra quem disparou o processo (UID efetivo) e quando ele foi iniciado;

- **m**: mostra o consumo de memória no processo;
- **f**: mostra uma árvore mostrando quem é pai de quem (útil para detectar, junto com outras opções, para detectar processos zumbis e seus pais);
- **w**: mostra o que não couber na linha na(s) linha(s) seguinte(s). Normalmente, cada processo ocupa apenas uma linha;

Existem muitas outras opções no **ps**, e para saber mais sobre elas a consulta à *manpage ps(1)* é *altamente* recomendável.

O **ps** mostra vários campos úteis, contendo o PID, nome e grupo real e efetivos do usuário que disparou o processo, tempo de execução, memória consumida e outros. Mas a coluna mais interessante para nós é a coluna **s**, ou **stat**, que apresenta para nós qual o estado de cada processo dentro do sistema:

- **D**: processo dormente em estado ininterrupto. Normalmente fica nesse estado processos que dependem de uma operação de I/O, como servidores de arquivo;
- **R**: processo que está ou pode vir a ser executado (*ready*, pronto);
- **S**: dormindo, pode ser chamado por um processo ou *system call*. Normalmente fica nesse estado processos que estejam esperando outros processos serem processados para agirem em cima de seus resultados (um servidor Web durante a execução de um CGI, por exemplo);
- **T**: paralizado, normalmente aparece quando um programa está sofrendo depuração;
- **W**: paginado, mandado para a memória virtual (caiu na versão 2.6 do *kernel* Linux);
- **Z**: processo zumbi.

Outra coluna útil é a **%CPU**, que retorna a quantidade de CPU consumida pelo processo em questão. A soma de todas as entradas da coluna **%CPU** *não*

costumam totalizar 100% (sempre é necessário um certo processamento para o *kernel*, em especial pelos sistemas de memória e escalonamento de processo, que *não aparece no ps*).

Outra coluna útil é a **%MEM**, que retorna o percentual de memória consumido pelo processo. Esse pode totalizar 100%, pois considera apenas o *user space*¹⁴ e não o *kernel space*¹⁵.

Como muitas vezes, ao usar-se **ps**, recebe-se *muitos* resultados, um utilitário muito útil para ser usado com o **ps** (e com muitos outros comandos do GNU/Linux) é o **grep**. Ele é um poderoso sistema de *expressões regulares* que pode comparar a entrada padrão ou um arquivo a um padrão determinado. Na prática isso quer dizer que ele pode encontrar trechos de texto dentro do arquivo ou saída em questão.

Por exemplo, imagine que você queira achar todas as instâncias do **bash**, use o comando **ps** ligado por um *pipe* ao **grep** e forneça ao **grep** o comando a ser investigado, no caso **bash**. O comando final ficará assim: **ps aux | grep bash**.

O Trecho de Código 5.3.1, na Página 5.3.1, mostra um exemplo de uma saída do **ps aux**. Procure atentar aos processos dos diversos tipos.

Se você quiser monitorar continuamente a *performance* do sistema, você pode utilizar-se do comando **top**, um programa interativo que é uma solução mais interessante do que usar um **watch 'ps'**. Como não é nosso foco aqui falar sobre *todas* as opções para fazer-se as coisas no GNU/Linux, você pode consultar bons guias de referência, como o Guia Foca GNU/Linux(SILVA, 2005a) ou, claro, a *manpage* do comando **top(1)**.

5.3.5 Mandando um processo para segundo plano e o trazendo de volta: **bg** e **fg**

Algumas vezes, você pode desejar interromper temporariamente um processo interativo (que são chamados na terminologia Unix de *trabalhos*, ou *jobs*) de primeiro plano para operar no *shell* por alguns instantes ou passar para

¹⁴**espaço do usuário**, aonde as aplicações do usuário são alocadas

¹⁵**espaço do kernel**, espaço reservado exclusivamente para o *kernel* e seus processos internos

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	1564	536	?	S	Apr22	0:00	init [5]
root	2	0.0	0.0	0	0	?	SN	Apr22	0:00	[ksoftirqd/0]
root	3	0.3	0.0	0	0	?	S<	Apr22	0:03	[events/0]
root	4	0.0	0.0	0	0	?	S<	Apr22	0:00	[khelper]
root	5	0.0	0.0	0	0	?	S<	Apr22	0:00	[kthread]
root	7	0.0	0.0	0	0	?	S<	Apr22	0:00	[kacpid]
root	79	0.0	0.0	0	0	?	S<	Apr22	0:00	[kblockd/0]
root	111	0.0	0.0	0	0	?	S	Apr22	0:00	[pdflush]
root	112	0.0	0.0	0	0	?	S	Apr22	0:00	[pdflush]
root	114	0.0	0.0	0	0	?	S<	Apr22	0:00	[aio/0]
root	113	0.0	0.0	0	0	?	S	Apr22	0:00	[kswapd0]
root	703	0.0	0.0	0	0	?	S	Apr22	0:00	[kseriod]
root	780	0.0	0.0	0	0	?	S	Apr22	0:00	[kjournald]
root	922	0.0	0.0	1560	500	?	S<s	Apr22	0:00	udev -d
root	1077	0.0	0.0	0	0	?	S	Apr22	0:00	[khud]
root	1415	0.0	0.0	0	0	?	S	Apr22	0:00	[scsi_eh_0]
root	1416	0.0	0.0	0	0	?	S	Apr22	0:00	[usb-storage]
root	2552	0.0	0.0	828	376	?	S	Apr22	0:00	/sbin/zcip -s -i eth0:9
rpc	2702	0.0	0.0	1684	588	?	Ss	Apr22	0:00	portmap
root	4264	0.0	0.0	1608	640	?	Ss	Apr22	0:00	syslogd -m 0 -a /var/spool/postfix/dev/log
root	4272	0.0	0.1	2340	1348	?	Ss	Apr22	0:00	klogd -2
root	4324	0.0	0.0	1548	596	?	Ss	Apr22	0:00	/usr/sbin/acpid
root	4370	0.0	0.0	1688	724	?	Ss	Apr22	0:00	rpc.statd
root	4426	0.0	0.2	5496	2456	?	Ss	Apr22	0:00	cupsd
xfs	4815	0.0	0.2	4216	2592	?	Ss	Apr22	0:00	xfs -port
71	4840	0.0	0.1	2224	1192	?	Ss	Apr22	0:00	dbus-daemon-1 --system
root	4865	0.0	0.2	4080	2696	?	Ss	Apr22	0:00	hald
root	4929	0.0	0.0	1828	704	?	Ss	Apr22	0:00	/usr/sbin/mandi -d
root	5088	0.0	0.0	2612	748	?	S	Apr22	0:00	/usr/bin/kdm -nodaemon
root	5095	0.0	0.0	2800	672	?	Ss	Apr22	0:00	nifd -n
root	5115	2.9	0.9	15600	9224	tty7	Ss	Apr22	0:25	/etc/X11/X
root	5167	0.0	0.1	3552	1884	?	S	Apr22	0:00	-:0
daemon	5259	0.0	0.0	1684	612	?	Ss	Apr22	0:00	/usr/sbin/atd
root	5284	0.0	0.1	4276	1648	?	Ss	Apr22	0:00	/usr/sbin/sshd
...										
root	5970	0.0	0.1	2660	1108	?	Ss	Apr22	0:00	/usr/bin/lisa -c /etc/lisarc
root	6062	0.0	0.0	1544	440	tty1	Ss	Apr22	0:00	/sbin/mingetty tty1
root	6063	0.0	0.0	1544	440	tty2	Ss	Apr22	0:00	/sbin/mingetty tty2
root	6064	0.0	0.0	1548	440	tty3	Ss	Apr22	0:00	/sbin/mingetty tty3
root	6065	0.0	0.0	1548	444	tty4	Ss	Apr22	0:00	/sbin/mingetty tty4
root	6066	0.0	0.0	1548	440	tty5	Ss	Apr22	0:00	/sbin/mingetty tty5
root	6067	0.0	0.0	1544	436	tty6	Ss	Apr22	0:00	/sbin/mingetty tty6
...										
fecosta	6331	0.0	1.4	27840	14076	?	S	Apr22	0:00	korgac --miniicon korganizer
fecosta	6333	0.1	0.0	0	0	?	Z	Apr22	0:00	[kdesu] <defunct>
fecosta	6337	0.0	0.5	15856	5712	?	S	Apr22	0:00	/usr/bin/kdesud
...										
root	6354	0.0	0.0	0	0	?	Z	Apr22	0:00	[ifup-post] <defunct>
fecosta	6451	0.2	0.0	0	0	?	Z	Apr22	0:02	[net_monitor] <defunct>
fecosta	6457	0.1	0.0	0	0	?	Z	Apr22	0:00	[kdesu] <defunct>
fecosta	6674	0.5	1.5	28424	15392	?	S	00:00	0:03	konsole [kdeinit]
fecosta	6675	0.0	0.1	3160	1832	pts/1	Ss	00:00	0:00	/bin/bash
fecosta	6717	0.1	0.0	0	0	?	Z	00:00	0:00	[kdesu] <defunct>
root	7135	0.0	0.0	1580	524	?	Ss	00:00	0:00	/sbin/ifplugd -b -i wlan0
postfix	7214	0.0	0.1	4416	1564	?	S	00:00	0:00	qmgr -l -t fifo -u -c
root	7238	0.0	0.1	2548	1212	pts/1	S	00:01	0:00	su -c ./smarturpmi.sh
root	7241	0.0	0.1	2480	1236	pts/1	S	00:01	0:00	/bin/sh ./smarturpmi.sh
...										
fecosta	7266	0.0	0.1	3152	1808	pts/2	Ss	00:01	0:00	/bin/bash
root	7362	0.0	0.1	2544	1212	pts/2	S	00:02	0:00	su
root	7365	0.0	0.1	2652	1572	pts/2	S	00:02	0:00	bash
fecosta	7416	2.4	2.0	41160	20948	?	S1	00:03	0:11	/usr/bin/gaim
fecosta	7472	0.0	0.1	3160	1836	pts/3	Ss	00:08	0:00	/bin/bash
fecosta	7530	1.4	0.7	10572	7480	?	S	00:10	0:00	emacs
fecosta	7531	0.0	2.0	41144	20944	?	S	00:10	0:00	/usr/bin/gaim
fecosta	7532	11.6	1.7	31276	18020	pts/3	S	00:10	0:01	kwrite
fecosta	7537	0.0	0.0	2572	836	pts/3	R	00:11	0:00	ps aux

Trecho de Código 5.3.1: Exemplo do comando `ps aux`

outro processo. Quase todos os programas GNU/Linux possuem uma forma de passar para o *shell*, seja disparando um *shell* dentro do processo, quanto permitindo que o usuário paralise temporariamente o *job* (através de `Ctrl` + `C`) para voltar ao *shell*. Esse atalho é padrão para a maioria dos programas interativos do GNU/Linux, e manda temporariamente o *job* para segundo plano.

Quando você sai, ele apresenta uma mensagem como a do Trecho de Código 5.3.2, Página 114. Nesse caso, pode-se ver que o texto mostra um [1] ou outro número qualquer. Esse é o *número do job em execução* ou *job id* (não confundir com o PID) do programa executado pelo usuário. Quase todos os próximos comandos usam esse *job id* para realizar suas funções. Uma opção mais complexa é fornecer o PID da aplicação, que você sempre pode obter usando `ps`.

[2] Stopped	cat >teste
-------------	------------

Trecho de Código 5.3.2: Exemplo de resultado de `Ctrl` + `Z` e do seu número do trabalho (*job id*)

A primeira coisa que você precisa saber é que é possível ter vários *jobs* para vários usuários rodando ao mesmo tempo em um sistema GNU/Linux, mas *apenas um pode estar em primeiro plano por usuário*¹⁶. Os demais (se possível) continuarão processando suas informações. Para trazer de volta um determinado *job* para o primeiro plano, utilize o comando `fg` (*foreground*) com o *job id* do processo, precedido por um `%` (por exemplo: `fg %1`).

Da mesma forma, você pode mandar um determinado *job* para segundo plano. O normal ao usar-se `Ctrl` + `C` é que o *job* passe para o estado de dormência (S ou D na lista de estado do processo no `ps`). Ou seja, ele irá parar de ser executado. Isso pode não ser interessante algumas vezes (por exemplo, se você deixou um cálculo grande ser processado enquanto editava um texto no `emacs`). Nesse caso, você pode mandar o *job* para o segundo plano com o comando `bg` com o *job id* do processo, precedido por um `%` (por exemplo: `bg %1`).

¹⁶Na verdade, por sessão aberta, sendo que cada sessão equivale a um *shell* ou terminal no qual o usuário está logado. Como não estamos vendo muito sobre *sessões* nesse documento, a “regra de casa” de um *job* em primeiro plano por usuário continua válida. Vamos falar mais sobre *sessões* quando falarmos mais sobre interfaces gráficas

O normal é que apareça uma linha similar ao do Trecho de Código 5.3.2, Página 114, mas com o comando sendo seguido de um `&`, o que reflete o fato do comando agora estar em segundo plano.

5.3.6 Vendo os processos de segundo plano: *jobs*

Algumas vezes você pode querer saber quais *jobs* estão em segundo plano naquele momento e como eles estão. A opção de usar um `ps` é possível, mas pouco viável, pois você teria que usar muitos outros utilitários ligados por *pipes* para filtrar o conteúdo desejado. Para isso, o GNU/Linux, como todos os Unix oferece um comando para mostrar os *jobs* em execução. Surpreendentemente, o nome do comando é *jobs*.

O normal do comando *jobs* é que ele retorne uma listagem com os seguintes dados, uma linha para cada *job* no sistema, como no exemplo do Trecho de Código 5.3.3, na Página 115:

jobId jobAtual Estado Comando

[1]	Running	ogg123 -zq /mnt/winE/Músicas\ Especiais/ &
[2]-	Stopped	cat >teste
[3]	Stopped	/usr/bin/mc -P "\$MC_PWD_FILE" "\$@"

Trecho de Código 5.3.3: Exemplo da saída de um comando *jobs*

Aonde *jobId* é o *job id* do *job* listado em questão. *jobAtual* normalmente será um espaço em branco, exceto em dois *jobs*, um marcado com o sinal de + e o outro com o sinal de -. O + indica o *job* atual, enquanto o - indica o *job* ao qual o sistema devolverá o usuário assim que ele encerrar o *job* marcado pelo sinal de +.

Estado mostra uma mensagem informando o estado atual do *job* em questão, sendo eles: **Running**(em execução), **Done**(encerrado com sucesso)¹⁷, **Done(status)**(encerrado com sucesso, mas que retornou um valor de **status**

¹⁷O POSIX, padrão do Unix ao qual o GNU/Linux segue, exige que todo processo ao encerrar-se envie um valor de *status* de volta ao sistema — normalmente um número inteiro sem sinal. Um processo terminado com sucesso retorna valor 0, enquanto processos

diferente de 0, indicado por `status`¹⁸) e **Stopped** (paralizado) ou **Suspended** (suspensão). O último campo, `comando`, indica o comando do *job* em questão.

Como opções, `jobs` oferece as opções `-l`, que retorna uma listagem mais completa que a anterior, trazendo informações que normalmente seriam incluídas no `ps`, o que pode ser muito útil para levantar a *performance* dos *jobs* sem recorrer ao `ps`, e `-p`, que apenas lista o PID dos *jobs* atuais.

Com isso acabamos com o comando `jobs`. Vamos seguir adiante, explicando como eliminamos processos com problemas.

5.3.7 “Matando” processos: `kill` e `killall`

Algumas vezes, processos ficam congelados ou travados por motivos bobos. Outras, precisamos mesmo é eliminar o processo, principalmente no caso de processos mal-executados ou maliciosos, garantindo assim que o sistema não irá travar. Nesse caso, a solução é “matar” o problema¹⁹. Para isso, utilizaremos o comando `kill`.

Esse comando normalmente recebe como parâmetro o PID ou o *job id* do sistema (precedido de %). Ele tentará “matar” o processo normalmente. Mas algumas vezes o processo pode querer “engrossar”. Nesse caso, você pode entrar junto com o PID ou *job id* um nível de sinal²⁰ entre os vários que o sistema disponibiliza, mostrados na Tabela 5.3.7, Página 117, retirada do “Guia Foca GNU/Linux”(SILVA, 2005a).

Desse vários sinais, o normal é usar os sinais `TERM` (15), `KILL` (9), ou `HUP`

encerrados com falhas ou abruptamente retornam valores de status diferentes de 0. As *manpages* e *infopages* dos comandos e utilitários do GNU/Linux trazem explicações sobre que tipo de erro provocou cada valor de status. Independente de qualquer outra coisa, o valor de retorno 0 é *obrigatório* e esperado de todos os utilitários, comandos e programas

¹⁸Algumas vezes isso quer dizer que o programa foi encerrado “elegantemente” — devolvendo todos os recursos alocados ao sistema — e que houve algum tipo de falha na execução do programa

¹⁹Por favor, não levem essa idéia para a vida real! Isso vale apenas na administração dos processos em sistemas GNU/Linux. Não tentar resolver seus problemas dessa forma. *Você foi avisado!!!!* :P

²⁰*Sinais* são utilizados para que processos dos sistemas Unix (GNU/Linux incluído) saibam quais comportamentos deverão assumir ao se encerrar

NOME DO SINAL	VALOR DO SINAL	AÇÃO	COMENTÁRIOS
HUP	1	A	Travamento detectado no terminal de controle ou finalização do processo controlado
INT	2	A	Interrupção através do teclado
QUIT	3	C	Sair através do teclado
ILL	4	C	Instrução Ilegal
ABRT	6	C	Sinal de abortar enviado pela função <code>abort</code>
FPE	8	C	Exceção de ponto Flutuante
KILL	9	AEF	Sinal de destruição do processo
SEGV	11	C	Referência Inválida de memória
PIPE	13	A	<i>Pipe</i> Quebrado: escreveu para o <i>pipe</i> sem leitores
ALRM	14	A	Sinal do Temporizador da chamada do sistema <code>alarm</code>
TERM	15	A	Sinal de Término
USR1	30,10,16	A	Sinal definido pelo usuário 1
USR2	31,12,17	A	Sinal definido pelo usuário 2
CHLD	20,17,18	B	Processo filho parado ou terminado
CONT	19,18,25		Continuar a execução, se interrompido
STOP	17,19,23	DEF	Interromper processo
TSTP	18,20,24	D	Interromper digitação no terminal
TTIN	21,21,26	D	Entrada do terminal para o processo em segundo plano
TTOU	22,22,27	D	Saída do terminal para o processo em segundo plano

Tabela 5.2: Sinais do sistema segundo POSIX

(1). Esses sinais querem dizer *encerrar o programa normalmente*, “matar” (*eliminar*) o processo e *reiniciar o processo*. O padrão do `kill` é usar o sinal `TERM` (15).

Nesse caso, você mata um processo por vez. Mas e se você quiser matar vários processo de uma vez, principalmente baseando-se no nome do aplicativo (por exemplo, para matar um processo malicioso que dispara milhares de si próprio para destruir sua rede ou redes alheias). Nesse caso, apenas um “tiro” não funciona, você precisará de uma “metralhadora”. E essa “metralhadora” é o comando `killall`.

Na prática, `killall` é uma versão turbinada do comando `kill`, que caça dentro dos processos ativos um determinado nome passado como parâmetro. Na prática, o comando `killall` tem como sintaxe `killall <sinal> <nome>`. Por exemplo, imaginemos que você queira matar todos os processo cujo comando seja `horcrux` com um sinal `KILL`. Nesse caso, o comando a ser disparado é `killall -KILL horcrux` ou `killall -9 horcrux`.

Existem algumas opções para `killall` que podem ser úteis, mas não iremos comentar elas aqui. Uma boa consulta à *manpage* `killall(1)` deve resolver. Existe também um comando que seria a “bomba atômica” dessa história, que é o comando `killall5`, que manda um sinal de encerramento para *todos os processos*. Esse comando *não deveria* ser usado para desligar ou reiniciar o sistema (prefira comandos simples, como `halt` e `shutdown`).

Isso deve encerrar esse tópico sobre o `kill`. Vamos seguir em frente.

5.3.8 Aumentando a prioridade do processo: `nice` e `renice`

Em geral, o escalonador de processos do GNU/Linux define certas prioridades aos processos, e essas prioridades são suficientes na grande maioria dos casos. Porém, algumas vezes precisamos que um determinado processo rode *realmente* rápido ou pode ser que precisemos diminuir a prioridade de um processo que está consumindo muito processamento, além do devido. Nesse caso, utilizamos os comandos `nice` e `renice`.

Esses dois comandos tem o mesmo objetivo, que é reconfigurar a prioridade dos processos no sistema. A diferença é que o `nice` é usado quando o

processo é criado (*leia-se*: quando o comando é dado) e o **renice** pode ser usado para alterar as prioridades de processo já carregados.

No caso, a sintaxe de ambos é parecida: **nice** <prioridade> '<comando>' e **renice** <prioridade> <pid>. No caso, **nice** não possui nenhuma opção interessante, enquanto **renice** possui uma muito interessante, que é a opção **-u** (*user* — usuário), que permite modificar as prioridades do sistema para *todos* os processos de um determinado usuário. Isso pode ser útil tanto para aumentar o processamento de um determinado usuário que precisa de resultados rápidos quanto para diminuir a prioridade dos processos de um espertinho que procurou aumentar as prioridades de um programa de DVD...

Agora, a pergunta que não quer calar é: como é que são definidas as prioridades.

O escalonador de processos do GNU/Linux utiliza certos valores para definir o grau de prioridade do usuário no sistema. Para isso, além de certos valores internos, ele utiliza o *niceness* (gentileza) do processo. Em geral, todos os processos, sem exceção, são lançados com *niceness* 0, que é o limite que um usuário típico pode lançar. É possível aumentar-se o *niceness* pelos comandos **nice**/**renice**, o que de certa forma quer dizer que o processo estará mais propenso a ceder sua vez na execução de comando (daí o *niceness*), até um valor 19 (menor prioridade possível). É possível também definir valores negativos de *niceness*, que querem dizer que o processo estará mais exigente quanto à sua posição no escalonamento de processos. Porém, para impedir que usuários inescrupulosos aproveitassem-se e transformassem o escalonamento de processos em uma verdadeira luta-livre para ver quem conseguiria mais recursos e tempo de processamento, o GNU/Linux impede que um usuário normal utilize um *niceness* negativo (apenas o **root** pode utilizar *niceness* negativo). A maior prioridade é -19.

Para um artigo introdutório bem interessante sobre o assunto escalonamento de processo no GNU/Linux, consulte Coelho (2003).

5.3.9 “Congelando” um processo: **nohup**

Como dissemos anteriormente, em geral quando um processo pai “morre”, os processos filho “morrem” com ele. Isso, porém, é muito ruim. O exemplo

clássico disso é o aplicativo gráfico que, aberto via um *shell* é derrubado quando o *shell* é fechado.

Para evitar esse problema, o Unix (e o GNU/Linux com ele) possui um comando chamado **nohup** (*no hangup* — sem queda), que impede o processo a ser aberto “morra” quando o seu processo pai “morra”. A sintaxe desse comando é simples: **nohup** '<comando>'

Com esse comando, terminamos a parte da administração de processos. Seguiremos agora em frente com a parte de *backup*.

5.4 *Backup*

Há um fato inevitável na informática: cedo ou tarde você *perderá* dados com os quais estava trabalhando. É fato: basta um comando mal-dado ou salvar um arquivo em uma localidade errada e lá se vai todo o trabalho de anos que você tinha, ou aquela apostila especial que você estava produzindo. Em um ambiente multi-usuário como o GNU/Linux isso é ainda pior: embora as permissões de acesso restrinjam de certa forma “acidentes”, nada impede que uma combinação de erro (ou procedimento malicioso) e permissões mal-configuradas podem colocar todo o trabalho de uma equipe a perder, ou até mesmo aquele projeto crítico em que a equipe de Pesquisa & Desenvolvimento estava trabalhando.

Para impedir (ou ao menos minimizar) impactos por causa da perda de dados provocado por comandos mal-dados ou arquivos removidos de maneira errônea, o GNU/Linux oferece uma série de utilitários de *backup* de dados de todos os tipos, desde ferramentas simples como o **cpio** até ferramentas proprietárias altamente sofisticadas com rotação automática de fitas, sistema de *storage* e muito mais.

Mas a principal ferramenta de *backup* no GNU/Linux é composto pelo conjunto de ferramentas **tar** + GNU **gzip** ou GNU **bzip2**.

5.4.1 Os utilitários tar, gzip e bzip2

O utilitário **tar** (*tape archiver* — arquivador para fitas) é um dos mais antigos utilitários do ambiente Unix em geral, e é a principal ferramenta de *backup* do mundo Unix, por sua simplicidade e versatilidade. Na prática, tudo o que o **tar** faz é pegar uma massa de arquivos e os transformar em uma saída formatada de bits (*stream*) que pode ser transformada em um arquivo ou enviada para um dispositivo físico de armazenamento²¹. Isso gera um arquivo com transporte facilitado para ser enviado por vários meios.

O problema é que um arquivo **tar** costuma ter o mesmo tamanho do arquivo/diretório, senão mais. Isso porque o **tar** *não oferece* compressão. A idéia é que a compressão dos dados fosse feita pelo *hardware* dos dispositivos de fita, portanto poupando o sistema de uma tarefa ingrata e custosa em processamento (para os minicomputadores nos quais as primeiras versões do Unix rodavam). Quando a coisa ficava para transmissões por meios digitais como redes UUCP²² ou via Internet, o problema da falta de compressão continuava-se presente.

O primeiro utilitário que foi construído para compactar os dados de um arquivo **tar** (ou de qualquer tipo) foi o **compress**, que sofria de um sério problema: uma patente sobre o algoritmo LZW de compressão que era de posse da Unisys e que impedia a criação de uma versão livre do **compress**. Para evitar isso, o projeto GNU construiu sua própria ferramenta de compressão, o GNU Zip, ou **gzip**. Usando ele, poderia-se comprimir um pacote **tar** e enviá-lo com compressão similar (em alguns casos melhor) que a oferecida pelo **compress** e livre de algoritmos patenteados. Além disso, o formato **gzip** foi descrito na forma de RFCs²³ (no caso, as RFCs 1950(DEUTSCH; GAILLY, 1996), 1951(DEUTSCH, 1996b) e 1952(DEUTSCH, 1996a))(PROJETO GZIP, 2003).

Com o tempo e o processamento dos sistemas melhorando, foi desenvolvido um protocolo ainda melhor de compressão de dados para Unix, o

²¹Demonstrando o poder de um dos principais princípios do Unix em geral, portado para o GNU/Linux, o KISS (*Keep It Simple and Safe* — mantenha tudo simples e seguro), ou seja, de que um utilitário deve fazer apenas *uma* tarefa, mas deve fazê-la MUITO bem

²²*Unix to Unix Copy* — um sistema rudimentar de transferência de informações entre ambientes Unix

²³*Request For Comments*, documentos que, de certa forma, são considerados os padrões de protocolos da Internet

bzip2, desenvolvido por Julian Seward (PROJETO BZIP2, 2005). Esse formato chega a conseguir 10 a 15% de compressão a mais em cima do **gzip**, além de, assim como **gzip** ser um utilitário *free software*.

É essa trinca que é usada para nossos *backups*. Vamos nesse caso ser um pouco diretos quanto aos comandos a serem usados, pois esses comandos são razoavelmente complexos. Como dica, fica a sugestão para ler-se as *manpages* **tar(1)**, **gzip(1)**, **gunzip(1)**, **bzip2(1)**, **bunzip2(1)** e suas *infopages*.

5.4.2 Criando um *backup* com o utilitário **tar**

Para criar um *backup*, em geral usamos o **tar** associado ao **gzip** ou **bzip2** via *pipe*. Normalmente usa-se algumas combinações padrão de opções no **tar**. No caso, o padrão é usar-se (CINEIROS, 2005):

```
tar cvf <arquivo> <dir> | gzip -
```

ou

```
tar cvf <arquivo> <dir> | bzip2 -
```

Essas opções padrão do **tar** querem dizer.

- **c** — indica que irá se criar um novo arquivo **tar**.
- **v** — cria um padrão de identificação de “pedaços” do arquivo **ar**. Esse é muito útil quando você vai trabalhar com várias fitas e usa o **tar** para gerar o *backup*
- **f** — Fornece um nome a ser usado pelo arquivo **tar**

No caso, a idéia é que o **tar** monte o arquivo. Depois esse arquivo é passado ao **gzip** ou ao **bzip2** pela saída padrão. Por sua vez, eles vão compactar o arquivo em questão, adicionando suas próprias extensões ao final do arquivo (normalmente **.gz** e **.bz2** respectivamente), devolvendo ao sistema esse arquivo final (conhecido no mundo Unix algumas vezes como *tarball*, principalmente para os arquivos **.tar.gz**).

Esse passo do pipe pode ser facilmente pulado usando-se os comando `tar cvzf <arquivo>` ou `tar cvjf <arquivo>`. Os símbolos `z` e `j` fazem respectivamente com que o `tar` chame por conta própria a compressão por `gzip` ou `bzip2`. Essas opções, porém, são próprias da versão GNU do `tar`. Portanto, se você for utilizar essas dicas em outros ambientes, pode acontecer dessas dicas não funcionar com você. De qualquer modo.

Existe uma série enorme (enorme *mesmo*) de opções que oferecem, entre outras coisas, possibilidades de *backup* incremental (ou seja, apenas fazer cópia de arquivos que tenham sido alterados depois de algum tempo), filtragem dos arquivos a serem copiados, e por aí afora. É muito recomendável a consulta das *manpages* e *infopages* do comando `tar`, principalmente pela grande variedade de comandos possíveis e opções que podem ser úteis. O que demos aqui foi uma introdução na questão do *backup*.

Atenção: Lembre-se de salvar os *backups* em mídias confiáveis e, caso seja para uma empresa, possuir uma estratégia que permita a rápida recuperação do *backup*.

5.4.3 Recuperando um *backup* com o utilitário `tar`

Como dissemos no começo dessa seção, o fato é que, cedo ou tarde, *você irá perder dados!* Essa é uma das maiores certezas da informática, que não importa o quão você seriamente você proteja seus dados ou que você seja cuidadoso ao salvar arquivos, basta uma tecla errada, uma opção mal-selecionada e os dados vão-se embora, e sua paz com eles.

Mas para isso é que fazemos *Backup*, de qualquer modo.

Mas precisamos restaurá-los. Para isso, vamos fazer a operação reversa da criação de um arquivo, a *extração* de dados do arquivo. Para isso usamos:

```
gunzip <arquivo> | tar xf -
```

ou

```
bunzip2 <arquivo> | tar xf -
```

No caso, o que fazemos aqui é o oposto exato do processo de criação do arquivos: primeiro os utilitários `gunzip` ou `bunzip2` (conforme que compactou o arquivo originalmente) descompactam o arquivo `.tar` que é enviado pelo *pipe* para o comando `tar`, sendo que esse realiza a extração dos dados dentro do arquivo.

Atenção: É *extremamente* aconselhável que essa extração seja feita em um diretório à parte, de modo que o usuário possa selecionar os arquivos a serem restaurados sem que perca dados e acabe piorando a situação (por exemplo, sobrescrevendo um arquivo com uma versão mais antiga do mesmo).

Como no caso da compactação, é possível evitar-se a necessidade de usar-se o *pipe* com o a adição da chave `z` ou `j`, conforme o compactador usado ao gerar-se o arquivo.

Antes de terminar-mos essa seção, queremos reiterar que o que falamos aqui foi o básico do básico sobre *backup*. Existem muitas opções e formas de usar-se a combinação do `tar` com `gzip` ou `bzip2`, além de haver outros utilitários para *backup* de dados. No caso, esse processo é um pouco rudimentar, mas muito efetivo. Na Internet pode-se encontrar com facilidade métodos mais modernos de *backup* para GNU/Linux.

Com isso terminamos esse Capítulo, aonde falamos do básico da administração de um sistema GNU/Linux. No próximo capítulo, falaremos sobre a conectividade em rede, um outro elemento muito importante na administração de um sistema GNU/Linux. Para se aprofundar no tópico da Administração do sistema GNU/Linux o conselho já foi dado no começo desse capítulo e que reiteramos, que é a consulta ao Guia FOCA GNU/Linux, Nível Avançado(SILVA, 2005b), ou ao *The Linux System Administrators' Guide*(WIRZENIUS *et al.*, 2005b), além de qualquer um dos variados livros sobre o assunto que podem ser encontrados nas prateleiras de lojas especializadas.

Capítulo 6

Rede e Internet

Nos nossos tempos modernos de globalização e troca de informações, ou “Sociedade da Informação”, como os sociólogos vêm nomeando essa nossa nova era (CASTELLS, 2000), um computador isolado, sem conectividade em rede é pouco mais útil que uma máquina de escrever. Na realidade, é impossível pensar em uma sociedade aonde sejamos cada vez mais dependentes do computador e das TIC¹. Mesmo com a baixa quantidade de usuários relativos a outras tecnologias, a conectividade em rede vem provocando muito mais impactos (e porque não dizer, estragos) na sociedade mundial do que outras tecnologias de informação.

Essa introdução visa dar ao leitor uma idéia da importância que tem o tópico administração de redes em um sistema atual. Na prática, quase sempre estamos administrando uma rede de alguma forma, nem que seja realizando algumas configurações para conectarmos-nos à Internet.

De qualquer forma, o que veremos aqui, assim como no Capítulo anterior, é apenas a introdução à administração de redes e Internet. Na prática, como o GNU/Linux, assim como a maioria dos sistemas baseados em Unix tem seus serviços de rede baseados em TCP/IP, tanto a conectividade local quanto a de/para a Internet é oferecido pelos mesmos comandos. Para maiores informações, um documento muito interessante a ser consultado (apesar de um tanto antigo) é o *The Linux Network Administrators' Guide, Second Edition*, de Kirch e Dawson (2000). Embora antigo e não falando de alguns

¹Tecnologias da Informação e Comunicação

assuntos interessantes, como a configuração de certos servidores, ele oferece bases reforçadas aqueles que quiserem aprender de maneira mais aprofundada o assunto de redes em GNU/Linux.

Além disso, o projeto *Linux HOWTO Project* (LINUX DOCUMENTATION PROJECT, 1999) tem vários documentos interessantes na questão de redes. Entre eles os principais são o *Linux Network HOWTO* (DRAKE, 1999), que, embora antigo e sem manutenção ainda pode conter várias informações úteis; e o par *Linux Ethernet HOWTO* (GORTMAKER, 2003), e o *Linux Wireless HOWTO* (ARCOMANO, 20002), que mostram informações sobre as principais tecnologias de rede da atualidade.

Atenção: Não falaremos nesse documento sobre a parte de detecção de *hardware*, pois a maioria das distros contam com bons sistemas de configuração e até mesmo com tecnologias de conexão à quente, como *hotplug* e *kudzu*. Vamos nos concentrar na questão da base lógica. Os documentos já citados poderão ajudar na questão de configuração de *kernel*.

6.1 Bases do TCP/IP

Nessa seção falaremos um pouco de teorias importantes no uso do TCP/IP. No caso, ainda não entraremos nos detalhes específicos do GNU/Linux em relação ao TCP/IP, sendo uma introdução rápida ao funcionamento do TCP/IP.

6.1.1 História da Internet e do TCP/IP

O TCP/IP é um protocolo de rede criado pela DARPA (*Defense Advanced Research Projects Agency* — **Agência de Projetos de Pesquisa de Defesa Avançada**) no ano de 1958. No auge da Guerra fria, o projeto da DARPA era desenvolver uma rede *não-hierárquica*, que, em caso de guerra, pudesse permanecer operacional mesmo com a destruição de um ou mais nós da rede. Na prática, não existiam na época redes *em malha*, aonde cada nó se comunicava de maneira não-hierárquica com os demais, existindo apenas redes hierárquicas. Em redes desse porte, o elo mais fraco é o servidor:

elimine-o da rede e todo o resto da mesma para de funcionar. Foi então desenvolvida uma estrutura de redes conhecida como ARPAnet, aonde normalmente cada nó se comunicava com pelo menos outros dois.

Com o tempo, o protocolo original da rede ARPAnet, o NCP (*Network Control Protocol* — Protocolo de Controle de Rede) foi demonstrando suas fraquezas, sendo que em 1979 começou a ser planejado um novo protocolo para a ARPAnet. Em 1980, parte da arquitetura foi alterada com o surgimento da RFC 761, “*Transmission Control Protocol*” (IETF, 1980). Em 1981, foi completada a pilha de protocolos TCP/IP, no caso com a publicação da RFC 791, “*Internet Protocol*” (IETF, 1981). Sendo um protocolo simples e rapidamente implementado em vários SOs. A principal implementação é a BSD, que pode ser encontrada em vários tipos de ambientes tanto livres quanto proprietários.

Com o surgimento da *World-Wide Web*, a Internet se popularizou rapidamente. Mas para os ambientes baseados em Unix (GNU/Linux entre eles), o TCP/IP é a fonte fundamental de conectividade e transferência de informações, por meio de serviços como Telnet², FTP³, NIS⁴ e NFS⁵. Portanto, mais do que apenas um sistema de conexão com outros ambientes de rede, o TCP/IP no GNU/Linux é a *própria conectividade em rede*. Diferentemente, por exemplo, do WindowsTM, que originalmente usava (e ainda usa) NetBIOS, ou do Novell NetwareTM, que usava originalmente o IPX/SPX, no GNU/Linux o TCP/IP é presente desde as raízes como protocolo de rede.

6.1.2 Definindo um IP

Na prática, normalmente o IP é definido pelo seu Provedor de acesso à Internet que determina o valor do IP que você deverá utilizar em sua máquina (ao menos, naquela(s) que tiver(em) comunicação com a Internet). Isso principalmente pelo fato de que não há IPs suficientes para todas as máquinas conectadas à Internet no mundo, então os provedores utilizam-se de combinações de elementos para "fingir" que existem IPs suficientes para todos os

²Terminal Remoto

³*File Transfer Protocol* — Protocolo de Transferência de Arquivos

⁴*Network Information Services* — Serviços de Informações em Rede

⁵*Network Filesystem* — Sistema de Arquivos em Rede

usuários da Internet no mundo, principalmente por meio do DHCP⁶ e do NAT⁷. No caso, esses mecanismos não serão estudados nessa apostila, mas farta documentação sobre eles podem ser encontrados na Internet.

Um IP é um endereço numérico de 32 bits (2^{32} combinações existentes) que costuma ser representado em quatro grupos de 8 bits, representados por 4 números decimais separados por pontos (notação essa chamada de *quad-dotted octets* — quatro octetos separados por pontos). Esses números são sequencialmente definidos a partir de um valor colocado em cada um dos seus quatro octetos (variando de 0 a 255) que são colocados, e uma rede é composta, em geral, por IPs cuja numeração seja contínua (por exemplo: todos os IPs de 10.0.0.1 a 10.0.0.255) são considerados uma rede. Na verdade, a Internet não possui o conceito de redes isoladas: embora na prática seja o conjunto de redes, para a Internet não existem redes isoladas. Com o tempo isso provocou lentidão e falhas sérias de segurança nas trocas de informações dentro da Internet (por exemplo, obrigando um pacote de informação a sair de dentro da rede à qual ia se destinar). Isso veio a provocar a criação de um mecanismo de “isolamento”, as máscaras de sub-rede.

6.1.3 Definindo máscara de sub-rede

Conceitualmente, uma máscara de sub-rede (chamada também de *subnetwork mask* ou *netmask*) é um valor que é usado para separar o endereço de uma rede do endereço do *host* dentro daquela rede. Esse mecanismo permite dividir a rede em estruturas lógicas chamadas *sub-redes*.

Uma máscara de subrede, da mesma forma que um endereço IP, é formada por um número binário de 32 bits. Porém, sua construção é diferenciada: para criar-se a máscara de sub-rede, você deverá definir em 1 todos os bits que irão indicar o endereço da rede, do dígito de maior ordem até o dígito de menor ordem. Esses bits definem dentro da máscara qual é o endereço da

⁶*Dynamic Host Configuration Protocol* — Protocolo de Configuração Dinâmica do Host, aonde o servidor envia à máquina que deseja estabelecer a conexão com a Internet um IP de uma pilha existente no servidor

⁷*Network Address Translation* — Tradução do Endereço de Rede, também conhecido como *IP Masquarading* — mascaramento de IP, técnica onde o servidor “traduz” os endereços internos da rede em endereços externos via Internet, pela combinação de endereços IP e portas TCP

rede. Os endereços do *host* na rede são determinados pelos bits restantes.

Na prática isso funciona basicamente assim: imagine que você utilize um IP 192.168.10.4, com uma máscara de rede de 16 bits (16 bits setados em 1). Seu endereço pode ser representado por 192.168.10.4/16 (ao que chamamos de endereçamento CIDR⁸), sendo que a subrede também pode ser representada por 255.255.0.0.

Perceba que a máscara ficaria assim em binário o endereço de subrede.
11111111111111111000000000000000

Utilizando então uma operação AND (E) bit a bit, temos o endereço da rede:

```

11000000101010000000101000000010
AND 11111111111111111000000000000000
-----
11000000101010000000000000000000

```

No caso, se dividirmos o bloco em quatro octetos e fizermos a conversão decimal, obteremos 192.168.0.0.

Utilizando um AND bit-a-bit contra a máscara *invertida*, podemos obter o *endereço do host na rede*:

```

11000000101010000000101000000010
AND 00000000000000000111111111111111
-----
00000000000000000000101000000010

```

Se convertermos o bloco para a notação de quatro octetos separados por pontos, teremos o endereço 0.0.10.4

Nesse exemplo, foi muito fácil separar a rede do *host* através da *netmask*, embora na prática a complexidade pode ser alta, pois você pode dividir sua rede fora dos limites dos octetos. Nada impede, por exemplo, que você utilize 20 bits de *netmask*. Na realidade, esse exemplo foi construído para ser facilmente compreensível em relação ao seu funcionamento.

⁸ *Classless Inter-Domain Routing* — Roteamento entre domínios sem uso de Classes

6.1.4 Endereços especiais

A máscara de subrede é importante de ser entendida pois, a partir dela, definem-se dois endereços importantes, o endereço da rede e o endereço de *broadcast*. O endereço da rede é usado como identificação da rede como um todo e é definido por um endereço especial aonde todos os bits da seção do *host* (os bits que ficam em 0 no *netmask*) são zerados. No exemplo citado anteriormente, podemos dizer que o endereço da rede em questão é 192.168.0.0.

O endereço de *broadcast* é um endereço que qualquer máquina deve escutar. Normalmente é um endereço aonde todos os bits da seção do *host* (os bits que ficam em 0 no *netmask*) estão definidos em 1. Por exemplo, se você definir como 1 todos os 16 bits equivalentes ao endereço do *host* na rede, você obterá como endereço 192.168.255.255.

Perceba que esses endereços *não podem* (ou ao menos não deveriam) ser usados como endereços de *hosts* na rede, para evitar problemas de funcionamento incorreto da rede.

Além disso, existe um endereço fixo chamado de *loopback* (retorno), que referencia permanentemente a mesma máquina. Esse endereço é usado principalmente em testes de sistemas de rede e operações que poderiam ser feitas em rede, mas aonde o cliente e o servidor das operações estão na mesma máquina. Esse endereço é chamado normalmente de *localhost* e é referenciado pelo IP 127.0.0.1.

Além disso, todos os IPs que com o primeiro octeto 0 são *reservados* como referência rápida a IPs dentro da mesma subrede do *host*. O mesmo vale para os IPs com primeiro octeto 127 e alguns IPs mais altos.

6.1.5 Classes de rede

A importância das máscaras de rede surge quando você passa a entender as *classes de rede*.

No começo da Internet, uma organização chamada ICANN (*Internet Corporation for Assigned Names and Numbers* — Corporação para a determinação de nomes e números da Internet) era a responsável pela administração

dos endereços de IP que eram atribuídos a cada corporação ou entidade que desejava se conectar à Internet. Para isso, usando as máscaras de subrede e determinadas faixas de endereço, ela criou as Classes de Rede. Essas Classes permitiam uma melhor distribuição dos endereços de IP conforme as necessidades das entidades envolvidas.

Por esse método, as Classes de Redes são divididas em:

- **Classe A:** São todos os endereços cujo bit mais significativo dentro do endereço de rede é 0 (cobrindo todos os IPs de 1.0.0.0 a 127.255.255.255). Na prática, existem 126 redes de Classe A, com $2^{24} - 2$ endereços de host por rede (lembrando de remover os endereços de rede e de *broadcast* que são reservados), usando *netmask* de 8 bits (255.0.0.0). Normalmente grandes corporações como GE e Citibank eram as principais favorecidas ao receber IPs da Classe A, além dos provedores de *backbone tier 1* (provedores da conectividade internacional pesada) ou universidades muito grandes, como a Universidade da Califórnia;
- **Classe B:** São endereços cujo bit mais significativo é 1 e o bit seguinte é 0 (cobrindo todos os IPs de 128.0.0.0 a 191.255.255.255). A *netmask* padrão nesse caso é 255.255.0.0 (16 bits). A classe B possui 2^{15} redes, cada uma contendo $2^{16} - 2$ endereços de *hosts* para as redes. Normalmente, os endereços B são usados em corporações médias ou em governos. Além disso, os provedores de *backbone tier 2*. Algumas faculdades de porte médio a nível mundial (como a UniCamp) também costumam usar endereços de Classe B;
- **Classe C:** é a classe mais usada de endereços. Seus IPs são aquelas aonde os dois bits de maior ordem são 1 e o bit de terceira maior ordem é 0 (cobrindo todos os IPs de 192.0.0.0 a 223.255.255.255). Tem como *netmask* padrão 255.255.255.0 (24 bits) e possui 2^{22} redes disponíveis, com 254 *hosts* por rede. É o mais comum e é usado em geral por todo tipo de provedor de acesso à Internet e pequenas empresas e instituições. Algumas vezes, usuários caseiros podem ter eles próprios um ou mais IPs válidos dentro dessa faixa;

Além dessas, existem duas outras classes de IP que não são usadas normalmente, sendo reservadas para usos especiais:

- **Classe D:** Os endereços de Classe D começam com os três primeiros bits de maior ordem com 1 e o quarto bit de maior ordem em 0. Eles são utilizados para *multicast*, sendo que o endereço indica um *grupo de hosts* que irão receber a mensagem enviada. Perceba que isso é diferente de *broadcast*, pois no *multicast* não são todas as máquinas que irão receber a informação em questão. Os IPs da Classe D são de 224.0.0.0 a 239.255.255.255;
- **Classe E:** Reservada para fins experimentais, tem os quatro primeiros bits de maior ordem em 1. No caso, os IPs dessa faixa são de 240.0.0.0 a 255.255.255.254. O IP 255.255.255.255 é reservado para *broadcast* na rede do *host*;

6.1.6 IPs privativos ou “inválidos”

Aparentemente existem muitos IPs em cada classe. Porém, existem muito mais usuários na Internet. Para isso, criou-se várias técnicas, como o uso de *netmasks* que não fixam-se aos limites dos octetos do IP (CIDR — *Classless Inter-Domain Routing* — Roteamento Interdomínio sem classes) e o uso de NAT (*Network Address Translation* — Tradução de Endereços de Rede), de maneira a aproveitar ao máximo os IPs disponíveis. Para facilitar isso, tomou-se uma decisão aparentemente contrária ao objetivo delineado, que foi a separação de faixas de IPs como de *uso privativo* (também chamados de IPs “inválidos”).

Esses IPs são utilizados principalmente em combinação ao NAT. A idéia é que a Internet enxergue apenas uma pequena quantidade de *hosts* de uma instituição (podendo ser até mesmo apenas um IP por instituição), e dentro dela várias máquinas usando IPs privativos façam a comunicação. No caso, fica a cargo de um computador chamado de *gateway* o uso do NAT, ou seja, a tradução da chamada de IP interno para a de um IP externo. Esse repassa as requisições para a Internet e gerencia o recebimento da informação externa de modo a determinar quem solicitou o quê e para quem deve ir qual informação recebida⁹. O diagrama da Figura 6.1, na Página 133, dá uma idéia de como funciona o NAT.

⁹Na prática a idéia é um pouco mais complexa, mas a parte básica pode ser compreendida como demonstrada. O NAT utiliza, na verdade, uma combinação das portas TCP com os endereços de IP para criar a conectividade desejada

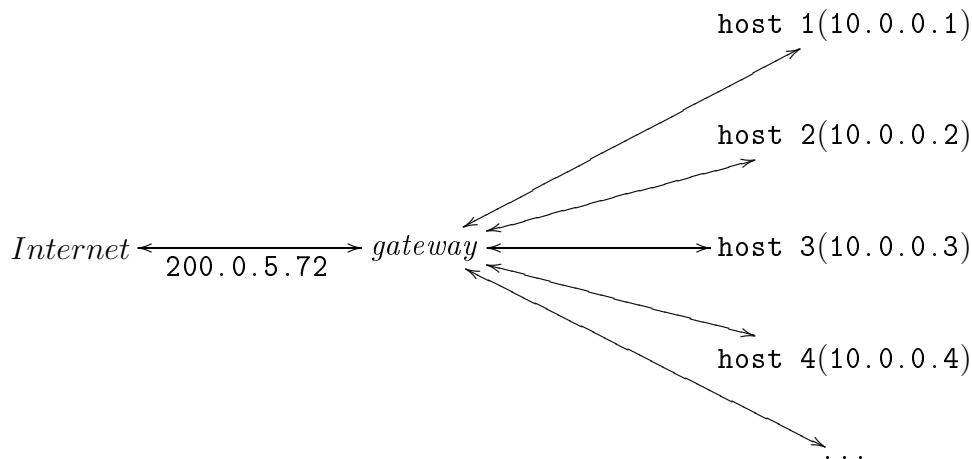


Figura 6.1: NAT e IPs privados

CLASSE	FAIXA DE IP	NÚMERO DE REDES
A	10.0.0.0 — 10.255.255.255	1
B	172.16.0.0 — 172.31.255.255	16
C	192.168.0.0 — 192.168.255.255	256

Tabela 6.1: Tabela de IPs privados

Para permitir isso, a *Internet Engineering Task Force* criou a idéia dos IPs privados, normalizada na RFC 1918, “*Address Allocation for Private Internets*”(IETF, 1996), aonde ela definiu as faixas de IPs reservados como IP privados, mostrado na Tabela 6.1, na Página 133. Na prática, espera-se que roteadores da Internet não roteem tráfego de IPs privados, exceto se forem da rede à qual o roteador pertencem. Também é importante isso como política de *firewall* não permitir que pacotes que proveiam da Internet mas que, por qualquer motivo, tenha sido endereçado com IP privados sejam roteados para dentro da rede.

Perceba que existe apenas uma rede privada de Classe A, enquanto existem 256 redes privadas de Classe C. Porém, perceba que *na realidade*, podemos ter tantas redes privadas de um determinado tipo quanto desejarmos, desde que a isolemos atrás de um IP válido para a Internet. Por

exemplo, duas empresas podem usar redes privadas de Classe A (10.0.0.0) normalmente, mas para se comunicar, devem fazê-lo por meio de IPs válidos via Internet. Não é possível comunicar-se diretamente com redes privadas isoladas diferentes da originária do pacote, *mesmo que o IP das duas sejam idênticas*.

Isso deve ser teoria suficiente para podermos seguir adiante em nossos estudos. Se desejar, procure mais informações sobre TCP/IP na própria Internet, ou consulte livros especializados, como “TCP/IP Illustrated, Volume 1 - The Protocols”(STEVENS, 1999), ou o “Interligação em rede com TCP/IP volume 1”(COMER, 2001).

Agora que já temos a teoria para formar base, vamos ao assunto em questão, que é a configuração da rede TCP/IP em redes GNU/Linux.

Atenção: Como já dissemos anteriormente, vamos apenas falar aqui da parte relacionada à estrutura lógica e da conectividade em rede, e não da parte relacionada à parte física, como a carga de módulos de *kernel*, principalmente porque a maior parte das distros GNU/Linux possuem recursos para auto-detecção de *hardware* em geral e de rede em particular. No caso, no início desse capítulo foi colocado determinados recursos que você pode consultar caso precise de mais informações nesse assunto em especial.

Atenção: Os comandos que serão vistos a seguir devem ser usados como **root**, pois eles lidam com o funcionamento do *kernel* Linux. O mesmo vale para os arquivos de configuração que serão mencionados.

6.2 Conectando uma máquina a uma rede IP: os comandos `ifconfig` e `route`

Tudo tendo corrido bem na instalação da sua distro GNU/Linux, sua placa de rede estará com seu *driver* (ou, como é mais correto dizer, *módulo do kernel*) carregado. Portanto, sua *interface* de rede estará *levantada* (carregada), mas

ela poderá *não estar* habilitada para trabalhar em rede. Para poder trabalhar em rede, será necessário definir um endereço de rede dentro da rede em questão. Para isso, utilizamos o comando `ifconfig` (*interface configuration* — configuração da Interface). Ele permite configurar parâmetros diversos da interface de rede, entre eles o endereçamento IP da sua placa.

Observação: Para certos tipos de *hardware*, certos parâmetros devem ser configurados com outros comandos. Por exemplo, para configurar informações como o ESSID, o canal de acesso ou a senha WEP¹⁰ de uma placa de rede Wi-Fi (802.11) você irá utilizar o comando `iwconfig`. Não trataremos desse assunto aqui, mas documentação suficiente sobre esse assunto pode ser encontrada na Internet, principalmente em documentos como o o *Linux Wireless HOWTO* (ARCOMANO, 20002).

Como de costume, aconselhamos que se consulte a *manpage* do comando em caso de dúvidas. No caso, usamos a *manpage* `ifconfig(8)`.

6.2.1 Explicando o comando `ifconfig`

A sintaxe mais normal do `ifconfig` é:

```
ifconfig <interface> [add|del] <address> hw <classe>  
netmask <mascara> broadcast <broad> [up|down]
```

A interface segue um nomeamento padronizado no GNU/Linux, no qual cada interface começa com um prefixo específico do tipo de dispositivo em questão e é completada por um número que indica sua ordem (começando por 0. Por exemplo, a primeira interface de rede Ethernet (a mais tradicional) é indicado por `eth0`. A Tabela 6.2, na Página 136 mostra alguns prefixos comuns de dispositivos de rede.

As opções em seguida, `add` e `del`, permitem adicionar ou remover um determinado endereço que o *hardware* em questão irá “escutar”. Isso permite servidores com vários IPs em uma única interface de rede, através de *alias*

¹⁰ *Wireless Encryption Password* — Senha de Criptografia Sem-Fio

PREFIXO	DISPOSITIVO
eth	dispositivos de rede Ethernet
wlan	dispositivos Wi-Fi
ppp	dispositivos com conexão PPP (<i>Point-to-Point Protocol</i>) Normalmente dispositivos de <i>modems dial-up</i> e ADSL

Tabela 6.2: Prefixos de Dispositivos de Rede

(apelidos) para as interfaces em questão. Não falaremos mais sobre esse assunto nesse documento, mas uma pesquisa na Internet poderá revelar mais sobre o assunto aos interessados.

A opção `hw` indica o tipo de *hardware* que será usado. Pode ser usada para alguns tipos de *hardware* que exijam um endereçamento físico fixo determinado pelo usuário, como redes Token Ring ou redes em sistemas Dec ou AppleTalk. Para as redes comuns Ethernet, isso não é necessário uma vez que:

1. O comando `ifconfig` consegue determinar “automagicamente” o tipo de *hardware* em questão pelo nome do dispositivo e;
2. Nas redes Ethernet, em geral o endereçamento físico é gravado na própria placa, não sendo exigida intervenção do usuário.

Atenção: Um dos motivos pelos quais o comando `ifconfig` só pode ser utilizado pelo `root` é o fato de que é possível utilizar-se ele para adulterar o endereçamento MAC¹¹ de uma placa de rede comum baseada em Ethernet ou em redes que utilizem esse esquema de endereçamento (como redes WiFi). Se esse endereço for adulterado, essa placa passará a receber *todos* os dados direcionados à máquina cuja qual os dados foram enviados. Na prática, chama-se isso de *MAC hijacking* (seqüestro de MAC) e é utilizado para casos aonde o atacante precisa clonar uma determinada máquina, como em ataques do tipo *man-in-the-middle* (homem do meio), principalmente para o farejamento (*sniffing*)

¹¹*Media Access Control* — Controle de Acesso ao Meio

ou para a enganação (*spoofing*) de redes, principalmente aquelas com esquemas de segurança baseados no endereçamento MAC.

netmask é uma opção que permite configurar a máscara de rede no esquema *quad-dotted octets*. Se você utilizar CIDR ou então trabalhar com as máscaras padrão da faixa do IP utilizado, essa opção não é necessária.

broadcast permite estabelecer um endereço de *broadcast* que o sistema utilizará para mandar pacotes para a rede como um todo. Essa opção é desnecessária se você fornecer um IP CIDR ou oferecer a *netmask* correta de sua rede, “automagicamente” gerando o endereço para você.

As opções **up** e **down** ativam (“levantam”) ou desativam (“derrubam”) a interface de rede em questão.

Mas configurar o IP pode não ser o suficiente, principalmente se for necessário conectar o equipamento em questão à Internet. Nesse caso, é necessário definir ao sistema como buscar uma conexão. Mais exatamente, definir o *gateway*¹² por onde o sistema deve buscar sua conexão com a Internet. Para isso, utiliza-se o comando **route**.

6.2.2 Explicando o comando route

O comando **route(8)** é um comando que permite que você manipule a *tabela de roteamento IP* do sistema. Essa tabela permite que uma máquina, chamada *gateway*, retransmita os dados de máquinas de uma rede para as de outra rede (desde que essa máquina pertença às duas *ou* saiba como repassar os dados adiante). Sua sintaxe é basicamente a seguinte:

```
route [add|del] [-net|-host|default] <target> [gw GW]
[[dev] <if>]
```

add e **del** funcionam como no caso de **ifconfig**.

-net e **-host** permite indicar uma rota para uma determinada rede ou *host*. Para indicar a rota para qualquer pacote que não seja local (ou seja, seja de outra rede), utilize **default**. Nesse caso, *não indique* o alvo (<target>).

¹²uma máquina ou equipamento especial de rede que conecta uma determinada rede IP a outra

`gw` faz com que você indique qual a máquina que irá atuar como *gateway*. Pode receber tanto um IP quanto um nome de máquina.

`dev` permite que você indique qual interface de rede que deverá ser usada para rotear os dados. Caso não seja indicado, o sistema irá tentar determinar qual a interface a ser usada para rotear os dados (na maioria dos casos com sucesso). Pode ser interessante definir essa interface para algum ganho em desempenho, mas isso não chega a ser obrigatório.

6.2.3 Um exemplo de configuração de rede

Vamos imaginar a seguinte rede de exemplo: no caso, o nome da rede será *hogwarts*. As máquinas são *gryffindor*, *hufflepuff*, *ravenclaw* e *slytherin*. A rede *hogwarts* usa o endereço de rede 10.0.0.0/8, com *netmask* padrão Classe A (255.0.0.0). Elas se conectam à Internet por meio da máquina *dumbledore*, que possui duas interfaces de rede: uma com um IP “normal” para a rede em questão e outra com um IP válido para a Internet, 200.126.15.2/24 (ou seja, IP 200.126.15.2 e *netmask* 255.255.255.0). A estrutura da rede em questão ficaria similar à do diagram da Figura 6.2, na Página 139. Não consideraremos (ainda) coisas como NAT ou *firewalls*, deixando isso para mais adiante.

No caso, imaginemos que você está configurando inicialmente a máquina *hufflepuff*. Como `root`, você deverá utilizar o comando:

```
ifconfig eth0 10.0.0.2 netmask 255.0.0.0 broadcast 10.0.0.255 up
```

Perceba que falamos que ele utiliza o *netmask* padrão para a sua Classe IP. Além disso, em geral o *broadcast* pode ser definido conforme a necessidade pelo `ifconfig`. Nesse caso, se desejar, você pode definir a interface com o seguinte comando.

```
ifconfig eth0 10.0.0.2 up
```

Além disso normalmente o `ifconfig` já conseguiria automaticamente estabelecer um roteamento de rede normal. Se você preferir, sempre pode-se usar `route` para estabelecer um roteamento manual, sendo que isso é útil

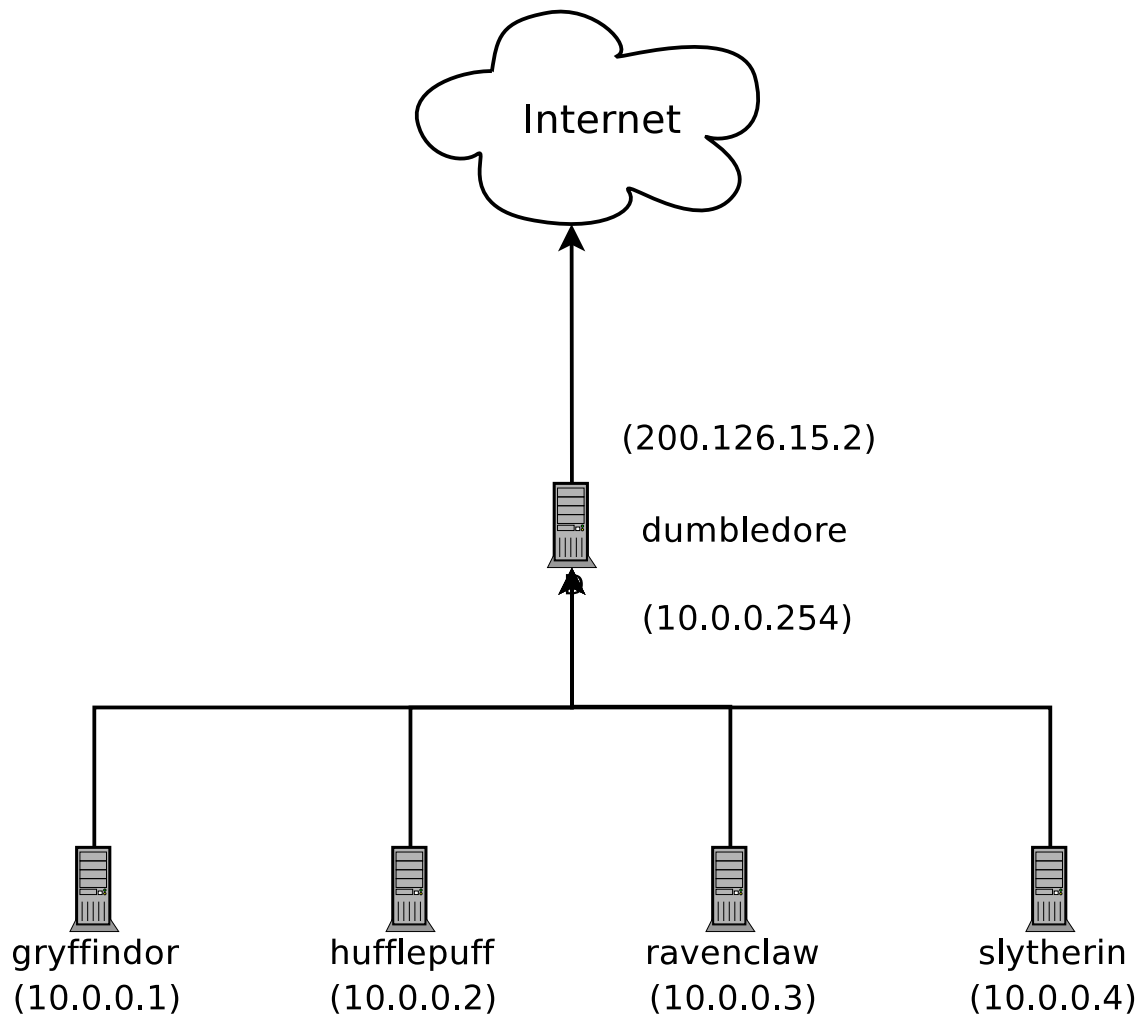


Figura 6.2: Rede exemplo hogwarts

principalmente para pacotes IPs para *fora* da rede interna (por exemplo, para a Internet ou para uma VPN¹³). Por exemplo, imaginando que queremos estabelecer uma rota padrão por *dumbledore* para *todos* os pacotes da rede, usamos o comando:

```
route add default gw 10.0.0.254
```

O próprio **route** se configura: o *The Linux Network Administrators' Guide, Second Edition*, de Kirch e Dawson (2000) explica que o *kernel* do Linux checa a tabela de interfaces de rede configuradas, de modo a rotear para uma interface configurada para mandar os pacotes de um IP parte da mesma rede do *gateway* e define essa interface como a interface de saída dos dados.

6.3 Arquivos de informações sobre redes

Como você deve ter notado, trabalhamos muito com números de IPs. Porém, embora IPs sejam fundamentais para o funcionamento de uma rede, é muito difícil para uma pessoa comum decorar os IPs de uma rede, exceto por redes pequenas e triviais, o que *difficilmente* é a realidade de um ambiente de redes tradicional, onde normalmente temos uma grande quantidade de IPs para os mais diversos tipos de recursos e máquinas, tanto *hosts* quanto terminais. Por isso, diversos esquemas de criar-se nomes simples de máquina foram desenvolvidos com o tempo.

Embora existam esquemas como DNS¹⁴ e DDNS¹⁵ (também conhecido como *Bonjour*), em redes não muito grandes, principalmente redes internas, pode-se adotar o esquema tradicional por arquivos do Linux. Esse esquema envolve basicamente quatro arquivos: `/etc/hosts`, `/etc/networks`, `/etc/HOSTNAME` e `/etc/resolv.conf`.

¹³*Virtual Private Network* – Rede Privada Virtual

¹⁴*Domain Name Service* – Serviço de Nomes de Domínio

¹⁵*Dynamic DNS* – DNS dinâmico


```
hufflepuff          hufflepuff.hogwarts
```

Trecho de Código 6.3.1: Exemplo de `/etc/HOSTNAME`

6.3.1 `/etc/HOSTNAME`

O arquivo `/etc/HOSTNAME` permite configurar-se o nome interno da máquina. Isso é muito útil em alguns casos, mas nas versões mais atuais do Linux o uso do `/etc/HOSTNAME` vem caindo por terra, na realidade o `/etc/HOSTNAME` contem apenas uma linha, como no caso do Trecho de Código 6.3.1, na Página 141. O primeiro nome é o nome simples da máquina, acessável a todos os usuários da rede em questão, enquanto o segundo é um FQDN¹⁶ da máquina em questão. Normalmente ele seria alguma coisa do tipo `www.google.com`, mas, principalmente em máquinas em intranets, nada impede que se adote um FQDN como o descrito anteriormente, ou seja, `hufflepuff.hogwarts`.

Nas versões mais atuais do *kernel* do Linux, o arquivo `/etc/HOSTNAME` acabou caindo por terra, sendo substituído por uma combinação do comando `hostname` (para configuração em *runtime*) e do arquivo `/etc/hosts` (para configuração persistente).

6.3.2 `/etc/hosts`

Na verdade, o `/etc/hosts` é o arquivo que contem as definições dos nomes de *hosts* de uma rede. Sua origem remonta a origem da Internet, quando um arquivo *hosts* era mantido pela IANA¹⁷ e redistribuído conforme era atualizado na Internet. Com o tempo, esse esquema deu lugar a esquemas como o DNS, mas sua utilidade para redes de porte pequeno a médio continuou bastante válida.

O arquivo `/etc/hosts` é similar em funcionamento ao arquivo `/etc/HOSTNAME`, mas com uma linha por IP. Além disso, cada IP pode ter, além dos nomes de máquina e dos FQDN, um ou mais *alias* (apelidos) para as mesmas estabelecidos. Por exemplo, na Trecho de Código 6.3.2, Página

¹⁶*Fully Qualified Domain Name* – Nome de Domínio Totalmente Descrito

¹⁷*Internet Assigned Names and Addresses* – Nomes e Endereços distribuídos pela Internet: organização que mantém a distribuição dos IPs de uma rede

```
127.0.0.1    localhost localhost.localdomain
10.0.0.1     gryffindor gryffindor.hogwarts security security.hogwarts
10.0.0.2     hufflepuff hufflepuff.hogwarts supercomp supercomp.hogwarts
10.0.0.3     ravenclaw ravenclaw.hogwarts data data.hogwarts
10.0.0.4     slytherin slytherin.hogwarts cert cert.hogwarts
10.0.0.254   dumbledore dumbledore.hogwarts proxy proxy.hogwarts
```

Trecho de Código 6.3.2: Exemplo de `/etc/hosts`

142, vemos a lista de *hosts* da Figura 6.2, Página 139. Além disso, cada domínio possui um ou mais *alias*, como o caso, por exemplo, de `supercomp`, que é um *alias* para `hufflepuff`.

Algumas instalações ou administradores preferem separar cada *alias* e FQDN por linha, repetindo o IP na primeira coluna do arquivo. O Trecho de Código 6.3.3, Página 143, apresenta um exemplo de `/etc/hosts` dividido por linha como o exemplo do Trecho de Código 6.3.2, Página 142. Essa disposição é herança dos padrões antigos do UnixTM, e continua sendo usado no Microsoft WindowsTM. A adoção desse padrão nos Unix mais modernos no Linux fica a critério do administrador do sistema, conforme suas necessidade. Pode-se inclusive usar uma versão separando os nomes de máquinas e os de *alias* Trecho de Código 6.3.4, Página 143. O esquema a ser adotado fica a critério do próprio desenvolvedor. De qualquer forma, uma sugestão é colocar comentários (que começam com `#`) explicando seu padrão.

Perceba, por fim, que em *todos* os exemplos, existe uma definição para o *localhost*. Isso é importante pois alguns serviços, como a interface gráfica X, dependem de uma entrada *localhost* para localizar corretamente os servidores e dados na máquina local. Nunca, *jámais*, apague a definição do *localhost*, ou crie um `/etc/hosts` sem a mesma, pois isso pode resultar em problemas de comportamento do sistema.

6.3.3 `/etc/networks`

`/etc/networks` auxilia a administração das redes às quais o sistema pode acessar diretamente, assim como ao nomeamento de redes e equipamentos. Seu principal uso é facilitar o uso do comando `route`, ao desobrigar o admi-

```
127.0.0.1 localhost localhost.localdomain
10.0.0.1 gryffindor
10.0.0.1 gryffindor.hogwarts
10.0.0.1 security
10.0.0.1 security.hogwarts
:
10.0.0.254 dumbledore
10.0.0.254 dumbledore.hogwarts
10.0.0.254 proxy
10.0.0.254 proxy.hogwarts
```

Trecho de Código 6.3.3: Exemplo de `/etc/hosts` separado por linhas

```
127.0.0.1      localhost localhost.localdomain
10.0.0.1      gryffindor gryffindor.hogwarts
10.0.0.1      security security.hogwarts
10.0.0.2      hufflepuff hufflepuff.hogwarts
10.0.0.2      supercomp supercomp.hogwarts
10.0.0.3      ravenclaw ravenclaw.hogwarts
10.0.0.3      data data.hogwarts
10.0.0.4      slytherin slytherin.hogwarts
10.0.0.4      cert cert.hogwarts
10.0.0.254    dumbledore dumbledore.hogwarts
10.0.0.254    proxy proxy.hogwarts
```

Trecho de Código 6.3.4: Exemplo de `/etc/hosts` separando nomes de máquina e *aliases*

```
hogwarts          10.0.0.0
beauxbatons       192.168.0.0
durmstrang        192.168.10.0
```

Trecho de Código 6.3.5: Exemplo de `/etc/network` para a rede da Figura 6.3

nistrador de decorar IPs de rede.

Vejamos agora um exemplo que nos auxilie a entender: imagine-mos a rede da Figura 6.2 (Página 139). Agora, imaginemos que a rede `hogwarts` comece a prover serviços para duas outras redes, `beauxbatons` e `durmstrang`, conforme apresentado na Figura 6.3 (Página 145). Como você pode perceber, as máquinas `durmstrang` e `beauxbatons` não possuem acesso via Internet direto, precisando passar pela rede `hogwarts`. Para isso, é necessário fazer o roteamento dos pacotes IPs de suas redes através da máquina `dumbledore.hogwarts`. Para isso, tem que editar o arquivo `/etc/networks` e configurar corretamente as redes. Em todos as estações com roteamento (no caso `dumbledore.hogwarts`, `maxime.beauxbatons`, `karkaroff.durmstrang`) o arquivo `/etc/network` deverá ser definido como no Trecho de Código 6.3.5, Página 144

Bem, você deve estar perguntando “*qual a diferença nisso tudo??*”. Na verdade, da mesma forma que o arquivo `/etc/hosts`, o `/etc/network` auxilia na configuração de rede por comandos. No caso, ele facilita a configuração via `route`. No caso, ambas as redes podem ser configuradas por meio do comando `route add default gateway dumbledore.hogwarts` e `route add -net beauxbatons`(por exemplo, na máquina `fleur.beauxbatons`).

Na verdade, existem sempre os mecanismos de DNS que podem valer muito mais a pena, mas esse sistema, para redes pequenas e médias deve ser mais do que o suficiente.

Nota: como conselho, cada máquina conectada a um ambiente como o da Figura 6.3, deve ter um arquivo `/etc/hosts` como o do Trecho de Código 6.3.6 (Página 146), para que o mesmo consiga localizar corretamente todas as máquinas.

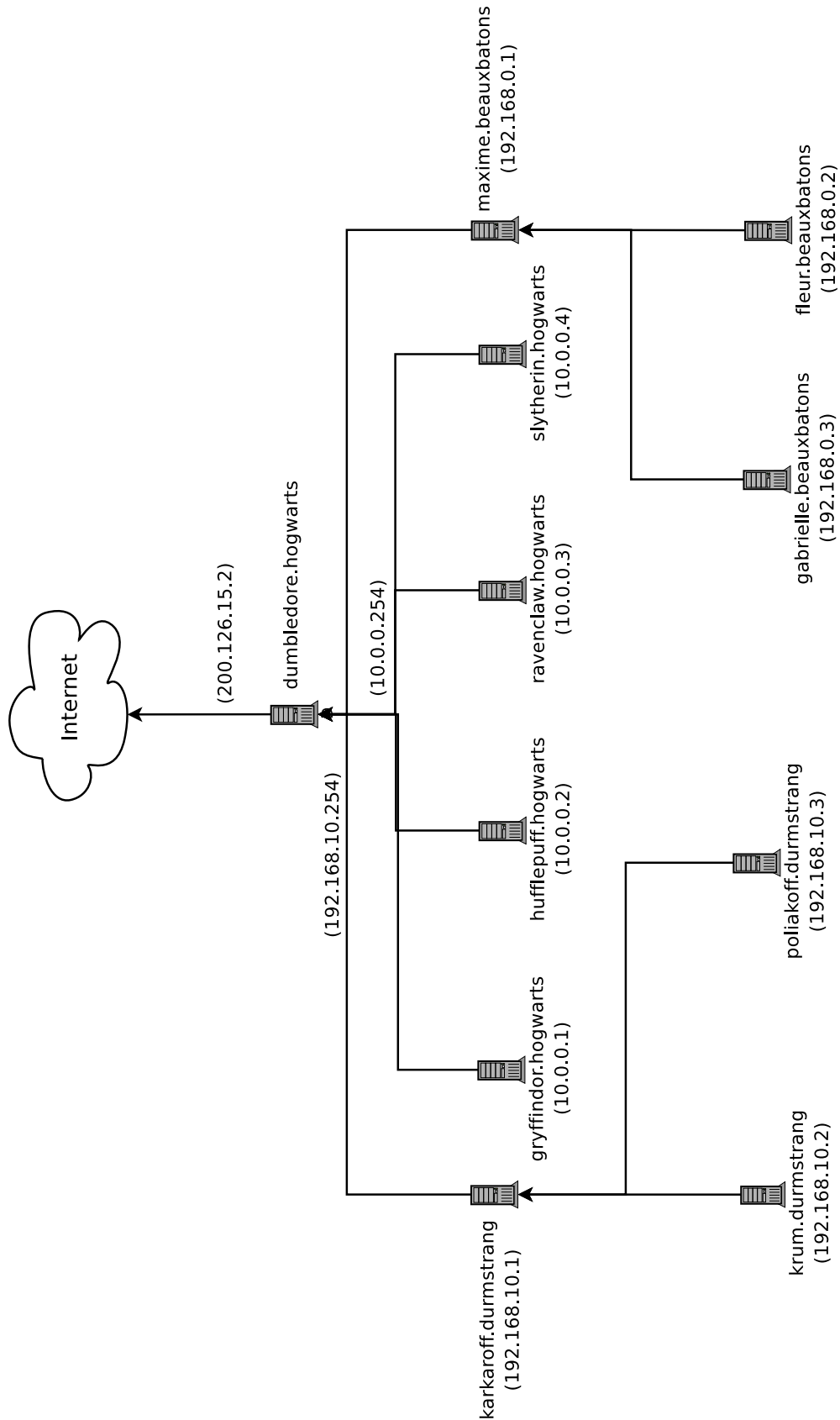


Figura 6.3: Redes exemplo hogwarts, durmstrang e beauxbatons

```
127.0.0.1      localhost localhost.localdomain
10.0.0.1       gryffindor gryffindor.hogwarts
10.0.0.2       hufflepuff hufflepuff.hogwarts
10.0.0.3       ravenclaw ravenclaw.hogwarts
10.0.0.4       slytherin slytherin.hogwarts
10.0.0.254     dumbledore dumbledore.hogwarts
192.168.10.1   karkaroff karkaroff.durmstrang
192.168.10.2   krum krum.durmstrang
192.168.10.3   poliakoff poliakoff.durmstrang
192.168.0.1    maxime maxime.beauxbatons
192.168.0.2    fleur fleur.beauxbatons
192.168.0.3    gabrielle gabrielle.beauxbatons
```

Trecho de Código 6.3.6: Exemplo de `/etc/hosts` para a rede da Figura 6.3

6.4 Arquivos de *resolver*

Uma parte importante das funcionalidades de rede do Linux envolve o *resolver*. Esse componente interno ao *kernel* do Linux permite que o sistema resolva os nomes de máquina, ou seja, faça a conversão do nome de máquina ou do FQDN para IPs.

Como foi dito, os nomes de máquina/*alias*/FQDNs são apenas formas de tornar ao usuário mais simples o acesso a máquinas ou recursos da mesma. Para o computador, ele *continua usando o IP*. Esse entendimento ajuda a compreender a importância do *resolver*. Na prática, ele utiliza todos os mecanismos que o *kernel* conhecer para *resolver o IP*, ou seja, pegar o nome fornecido pela aplicação do usuário, “traduzir” esse nome para um valor de IP que o sistema possa usar na transmissão via rede, e usá-lo para conectar a máquina cliente ao *host* com o serviço desejado.

Para configurar como o *resolver* do ambiente Linux irá se comportar, são usados dois arquivos, o `/etc/host.conf` e o `/etc/resolv.conf`.

6.4.1 O arquivo `/etc/host.conf`

O arquivo `/etc/host.conf` configura o comportamento do *resolver*. Em geral ele vai ter apenas algumas configurações, sempre linha por linha.

A primeira opção é **order**, que configura a seqüência na qual o *resolver* tentará resolver o nome de máquina passado a ele. Os valores dele são:

- **bind** — esse parâmetro indica que o sistema irá recorrer ao sistema de DNS para resolver o nome de máquina;
- **hosts** — esse parâmetro faz com que o sistema utilize o esquema tradicional, que é através dos arquivos citados anteriormente, como `/etc/hosts` e `/etc/network`;
- **nis** — esse parâmetro permite que o sistema resolva o nome de máquina pelo sistema de NIS (*Network Information Service* – Serviço de Informação de Rede), um sistema originalmente da Sun Microsystems que é muito usado em ambiente exclusivamente Unix, como no caso de redes *cluster*, por sua simplicidade e leveza comparada com outros sistemas;

Essa opção não possui valor único, podendo ser inseridas todas essas opções sem problemas. No caso, cada opção deve ser separada das demais com uma vírgula, como no caso de **order bind, hosts**, e as opções devem ser colocadas na seqüência de prioridade à qual o sistema irá recorrer na resolução do nome. Nesse exemplo, o sistema irá recorrer primeiro ao DNS e em seguida ao sistema tradicional de arquivos do Unix.

Outra opção importante é **multi**, aonde o *resolver* responde à requisição do sistema recuperando *todos* os valores IP que ele puder resolver para o nome de máquina oferecido. Embora isso ocasionalmente possa provocar conflitos, é muito interessante manter em **on** permitindo, por exemplo, que no caso de mudança de IP não aja problemas na resolução de um *site*. Agora, no caso de ambientes internos simples, pode ser interessante desativar essa opção com **off**.

Uma opção de segurança na resolução de IP é **nospoof**. Ativando essa opção em **on**, o sistema irá, após uma resolução (principalmente para *remote*

```
order hosts,bind
multi on
```

Trecho de Código 6.4.1: Exemplo de `/etc/hosts.conf`

shell, *telnet*, e *secure shell*) executá-la novamente. Se os valores de *ambas* as resoluções não casarem, isso significa uma tentativa de *spoof* por técnicas como *DNS poisoning*¹⁸ e similares, o que o sistema irá automaticamente “cortar”. Porém, saiba que isso com certeza irá reduzir a performance do sistema, pois ele deverá realizar *duas* resoluções de nome **por vez**. Portanto, use com cautela essa opção.

A opção anterior pode ter seu nível de segurança ampliado ainda mais por meio de outra opção, que é a opção **spoofalert**, também com valores **on** e **off**. No caso, se *ambas* as opções **nospoof** e **spoofalert** estiverem ativadas (**on**), uma mensagem de erro do *resolver* será enviada ao *syslog*, o registro de informações (*log*) do sistema. Para mais informações sobre o *syslog*, consulte a Seção 11.5, na Página 220.

Isso deve ser o suficiente sobre `/etc/host.conf`. Um exemplo desse arquivo pode ser encontrado no Trecho de Código 6.4.1, Página 148.

6.4.2 O arquivo `/etc/resolv.conf`

O segundo arquivo que podemos dizer que é de configuração do *resolver* é o `/etc/resolv.conf`. Esse arquivo permite configurar corretamente o *resolver* em como ele irá se comportar no caso das opções estabelecidas no `/etc/hosts.conf`. Ele armazena informações sobre quais servidores de nome deverão ser procurados e em que sequência. Perceba que quando é usado o termo servidor de nome (*nameserver*) nesse arquivo, *não diferenciaremos* servidores de nome dos diversos modelos, como NIS ou DNS. A responsabilidade quanto a isso deve ser do *resolver*, baseando-se no esquema definido no arquivo `/etc/hosts.conf`.

É importante deixar claro que *não é necessário* configurar o *resolver* em

¹⁸*DNS poisoning* é uma técnica que envolve a substituição de valores DNS legítimos por valores falsificados, por meio de substituições na sequência em que os servidores são chamados, literalmente “envenenando” o *cache* de DNS do sistema

uma máquina que tenha um servidor DNS ou NIS próprio rodando, ou que ainda necessite apenas de informações de nome de *hosts* em rede vindas do `/etc/hosts`. A configuração do *resolver* só é necessária se o sistema precisar buscar servidores de nome remotos. Se um servidor de nome estiver configurado, o *resolver* poderá ser desativado, deixando por conta do servidor de nomes local da máquina a responsabilidade de resolver o nome de máquina ao servidor local. Isso só é necessário quando um sistema precisar de um servidor de nomes remoto, como em um *desktop* conectada à Internet sem nenhum servidor de nomes local ativo.

O `/etc/resolv.conf` é construído de forma similar ao `/etc/hosts.conf`, ou seja, com uma opção por linha de texto no arquivo. Suas principais opções são:

- **nameserver**: essa opção permite configurar o IP de uma máquina remota que servirá a resolução de nomes de máquina. Perceba que *não é necessário determinar qual o tipo de sistema de nome adotado*, NIS, DNS ou qualquer outro. Isso fica a cargo do próprio *resolver* determinar;
- **domain**: essa opção ajuda a identificar o domínio ao qual a máquina configurada pertence, de maneira que o sistema de consulta de nome de domínio possa trabalhar adequadamente na resolução de nomes curtos ou *aliases* passados. Isso é feito por meio da expansão do nome curto/*alias* para o FQDN apropriado. Por exemplo, imaginemos que a máquina **fleur** do arquivo `/etc/hosts` do Trecho de Código 6.3.6 (Página 146) esteja buscando o servidor **hufflepuff.hogwarts**. Ela pode utilizar o nome curto **hufflepuff**, mas o *resolver* em sua máquina está configurado para “expandir” os nomes curtos de máquina usando **beauxbatons** como nome curto. No caso, o primeiro servidor que ela irá procurar será o “expandido”, que será determinado como **hufflepuff.beauxbatons**. Como essa máquina não existe, ele tentará localizar **hufflepuff**, sem nenhum nome de domínio. Essa máquina também não existe, o que irá provocar a falha na pesquisa.

Essa opção pode receber até seis domínios diferentes para “expansão” dos nomes curtos e *aliases* oferecidos, sendo que o número de caracteres nessas seis opções *juntas* não pode superar 256 caracteres.

```
nameserver 200.167.20.5
nameserver 200.176.2.10
# nameserver 192.168.20.2

# ppp temp entry
```

Trecho de Código 6.4.2: Exemplo de `/etc/resolv.conf`

Existem outras tantas opções, que podem ser consultadas na *manpage resolv.conf(8)*. O Trecho de Código 6.4.2, na Página 150, apresenta um exemplo de configuração de `/etc/resolv.conf`.

6.5 O arquivo `/etc/rc.d/init.d/network` e o arquivo `/etc/sysconfig/network`

Uma pergunta que deve estar passando por sua cabeça é “tudo bem, entendi tudo isso. Mas como a máquina sabe qual é a configuração de rede que ele deve ter?”

Na realidade, como na maioria dos sistemas operacionais, as configurações no Linux são *transientes*, ou seja, elas estão ativas enquanto o sistema estiver no ar. Uma vez que o sistema seja retirado do ar, as configurações desaparecem.

Isso parece uma tolice, mas a realidade é que o funcionamento da maioria dos sistemas é esse. Porém, assim como outros sistemas com essa característica, o Linux oferece um mecanismo que permite a permanência dessas configurações. No caso, a maioria das distros GNU/Linux possuem algum tipo de *shell script* que permite carregar e configurar as informações das interfaces de rede no *boot* da máquina. Nos sistemas baseados em System V Unix (a grande maioria), esse arquivo é o `/etc/rc.d/init.d/network`. Em outras distros, baseadas no modelo BSD (como o Slackware), o arquivo é o `/etc/rc.d/rc.inet1`. Veremos mais sobre os vários modelos de inicialização na Seção 10.2.2, Página 217.

No caso, não há muito a se dizer sobre o arquivo em questão: ele é um *shell script* pré-construído de fábrica, que permite sua configuração por

outros caminhos. A título de configuração, o Trecho de Código 6.5.1 a 6.5.6, Páginas 152 a 157, apresenta o `/etc/rc.d/init.d/network` da distribuição Mandriva Linux 2006.0 Free.

Como podemos ver, trata-se de um *script* extremamente complicado, sendo que nenhuma de suas vertentes são simples, não importa a distro.

Por isso mesmo, as distros “isolam” a parte de configuração da parte das funcionalidades do *script* da parte de configuração. Nas distros baseadas em System V, o arquivo é `/etc/sysconfig/network`, enquanto o arquivo para as distros baseadas em BSD normalmente é `/etc/rc.d/rc.inet1.conf`. No caso, vamos falar do arquivo em sua vertente System V. As versões BSD costumam se comportar da mesma forma, com pequenas e perceptíveis diferenças.

Da mesma forma que os outros arquivos de configuração apresentados anteriormente, esse arquivo é composto de linhas com parâmetros de configuração do sistema. O Trecho de Código 6.5.7, da Página 158, mostram o arquivo na versão do Mandriva Linux 2006.0 Free. Como podemos perceber, é um arquivo extremamente simples. Cada linha é de um parâmetro, sendo que os valores são indicados adiante. Não vou comentar os parâmetros, pois creio que são extremamente auto-explicativos. Então, não há necessidade de muitas explicações. Procure apenas perceber como o arquivo funciona. Os parâmetros podem variar de distro para distro, mas em geral são sempre auto-explicativos. Dê uma boa olhada no seu sistema e verifique como você entende eles. Isso deve ser o suficiente.

6.6 Alguns sistemas de troca de arquivos

Existem muitas funções importantes no uso de servidores, inclusive como banco de dados e servidores de Web, mas em um ambiente “comum” de empresa ou de escola, uma das maiores, senão a maior, utilidade de um servidor é como um servidor de arquivos. Portanto, falaremos mais sobre isso nesse capítulo. Para aqueles interessados em mais sobre servidores em ambientes Linux, o livro “Linux: Redes e servidores, Guia Prático — 2ª Edição”, de Morimoto (2006) possui MUITA informação sobre o assunto servidores, e tudo de maneira bastante prática, o que torna muito mais simples para você aprender.

```

#!/bin/bash
#
# network          Bring up/down networking
#
# chkconfig: 2345 10 90
# description: Activates/Deactivates all network interfaces configured to \
#              start at boot time.
# probe: false
### BEGIN INIT INFO
# Provides: $network
### END INIT INFO

# Source function library.
. /etc/init.d/functions

if [ ! -f /etc/sysconfig/network ]; then
    echo "NETWORKING=no" > /etc/sysconfig/network
    exit 0
fi

. /etc/sysconfig/network

if [ -f /etc/sysconfig/pcmcia ]; then
    . /etc/sysconfig/pcmcia
fi

# Check that networking is up.
[ "${NETWORKING}" = "no" ] && exit 0

# if the ip configuration utility isn't around we can't function.
[ -x /sbin/ip ] || exit 1

# Even if IPX is configured, without the utilities we can't do much
[ ! -x /sbin/ipx_internal_net -o ! -x /sbin/ipx_configure ] && IPX=

# Even if VLAN is configured, without the utility we can't do much
[ ! -x /sbin/vconfig ] && VLAN=

# If IPv6 is explicitly configured, make sure it's available.
if [ -n "$NETWORKING_IPV6" ]; then
    alias='modprobe -c | awk '/^alias net-pf-10 / { print $3; exit }''
    if [ "$NETWORKING_IPV6" = "yes" ]; then
        new_alias=ipv6
    fi
    if [ "$NETWORKING_IPV6" = "no" ]; then
        new_alias=off
    fi
    if [ -n "$new_alias" ]; then
        if [ "$alias" != "$new_alias" -a ! -f /proc/net/if_inet6 ]; then
            case "$(modprobe -V 2>/dev/null)" in
                modprobe* )
                    echo "alias net-pf-10 $new_alias" >> /etc/modules.conf
                    ;;
                module-init-tools* )
                    echo "alias net-pf-10 $new_alias" >> /etc/modprobe.conf
                    ;;
            esac
        fi
    fi
fi

```

Trecho de Código 6.5.1: Exemplo de `/etc/rc.d/init.d/network`

```

CWD='pwd'
cd /etc/sysconfig/network-scripts

. network-functions

# find all the interfaces besides loopback.
# ignore aliases, alternative configurations, and editor backup files
interfaces='ls ifcfg* | LANG=C egrep -v '(ifcfg-lo|:|rpmsave|rpmorig|rpmnew)' | \
    LANG=C egrep -v '(~|\.bak)$' | \
    LANG=C egrep 'ifcfg-[A-Za-z0-9\._-]+$' | \
    sed 's/^ifcfg-//g' |
    sed 's/[0-9]/ &/' | LANG=C sort -k 1,1 -k 2n | sed 's/ //'

boot=boot

# See how we were called.
case "$1" in
    start)
        # IPv6 hook (pre IPv4 start)
        if [ "$NETWORKING_IPV6" = "yes" ]; then
            if [ -x /etc/sysconfig/network-scripts/init.ipv6-global ]; then
                /etc/sysconfig/network-scripts/init.ipv6-global start pre
            fi

            action "Setting network parameters: " sysctl -e -p /etc/sysctl.conf

            if [ -r /etc/ethers -a -x /sbin/arp ]; then
                action "Storing ARP mapping" /sbin/arp -f /etc/ethers
            fi

            # bring up loopback interface
            action "Bringing up loopback interface: " ./ifup ifcfg-lo

            case "$IPX" in
                yes|true)
                    /sbin/ipx_configure --auto_primary=$IPXAUTOPRIMARY \
                        --auto_interface=$IPXAUTOFRAME
                    if [ "$IPXINTERNALNETNUM" != "0" ]; then
                        /sbin/ipx_internal_net add $IPXINTERNALNETNUM $IPXINTERNALNODENUM
                    fi
                    ;;
                esac
            # deprecated but we still use it.
            if [ -f /proc/sys/net/ipv4/ip_forward ] && [[ "$FORWARD_IPV4" = "yes" || "$FORWARD_IPV4" = "true" ]];
            then
                action "Enabling IPv4 packet forwarding" sysctl -n -w net.ipv4.ip_forward=1
            fi

            case "$VLAN" in
                yes)
                    if [ -d /proc/net/vlan ] || modprobe 8021q >/dev/null 2>&1 ; then
                        action "Setting 802.1Q VLAN parameters: " /sbin/vconfig set_name_type DEV_PLUS_VID_NO_PAD
                    else
                        gprintf "No 802.1Q VLAN support available in kernel.\n"
                    fi
                    ;;
                esac

            vlaninterfaces=""
            cipeinterfaces=""
            xdslinterfaces=""
            bridgeinterfaces=""

```

Trecho de Código 6.5.2: Exemplo de
/etc/rc.d/init.d/network(Continuação)

```

# bring up all other interfaces configured to come up at boot time
for i in $interfaces; do
eval $(LANG=C fgrep "DEVICE=" ifcfg-$i)
eval $(LANG=C fgrep "TYPE=" ifcfg-$i)
eval $(LANG=C fgrep "SLAVE=" ifcfg-$i)
eval $(LANG=C fgrep "BRIDGE=" ifcfg-$i)

if [ -z "$DEVICE" ] ; then DEVICE="$i"; fi

if [ "${DEVICE##cipcb}" != "$DEVICE" ] ; then
cipeinterfaces="$cipeinterfaces $i"
unset DEVICE TYPE SLAVE BRIDGE
continue
fi
if [ "$TYPE" = "xDSL" -o "$TYPE" = "ADSL" ]; then
xdslinterfaces="$xdslinterfaces $i"
unset DEVICE TYPE SLAVE BRIDGE
continue
fi

if [ -n "$BRIDGE" ]; then
is_available $i
bridgeinterfaces="$bridgeinterfaces $i"
unset DEVICE TYPE SLAVE BRIDGE
continue
fi

if [ "${DEVICE%%.*}" != "$DEVICE" ] ; then
vlaninterfaces="$vlaninterfaces $i"
unset DEVICE TYPE SLAVE BRIDGE
continue
fi

if [ "$SLAVE" = "yes" ]; then
unset DEVICE TYPE SLAVE BRIDGE
continue
fi

if LANG=C egrep -q "^ONBOOT=['\"]?[Nn][Oo]['\"]?" ifcfg-$i; then
continue
fi
# If we're in confirmation mode, get user confirmation.
[ -f /var/run/confirm ] &&
{
    confirm $i
    case $? in
0)
:
;;
2)
CONFIRM=
;;
*)
continue
;;
esac
}
action "Bringing up interface %s: " $i ./ifup $DEVICE $boot
done

```

Trecho de Código 6.5.3: Exemplo de
/etc/rc.d/init.d/network(Continuação)

```

# Bring up xDSL and CIPE interfaces
for i in $vlaninterfaces $bridgeinterfaces $xdslinterfaces $cipeinterfaces ; do
    if ! LANG=C egrep -q "ONBOOT=['\"]?[Nn][Oo]['\"]?" ifcfg-$i; then
# If we're in confirmation mode, get user confirmation.
if [ -f /var/run/confirm ]; then
confirm $i
test $? = 1 && continue
fi
action "Bringing up interface %s: " $i ./ifup $i boot
    fi
done

# Add non interface-specific static-routes.
if [ -f /etc/sysconfig/static-routes ]; then
grep "^any" /etc/sysconfig/static-routes | while read ignore args ; do
    /sbin/route add -$args
done
fi

# IPv6 hook (post IPv4 start)
if [ "$NETWORKING_IPV6" = "yes" ]; then
if [ -x /etc/sysconfig/network-scripts/init.ipv6-global ]; then
/etc/sysconfig/network-scripts/init.ipv6-global start post
fi
fi

touch /var/lock/subsys/network
;;
stop)
# If this is a final shutdown/halt, check for network FS,
# and unmount them even if the user didn't turn on netfs

if [ "$RUNLEVEL" = "6" -o "$RUNLEVEL" = "0" -o "$RUNLEVEL" = "1" ]; then
NFSMTAB='LC_ALL=C awk '!/^#/ && $3 ~ /^nfs/ { print $2 }' /proc/mounts'
SMBMTAB='LC_ALL=C awk '!/^#/ && $3 == "smbfs" { print $2 }' /proc/mounts'
NCPMTAB='LC_ALL=C awk '!/^#/ && $3 == "ncpfs" { print $2 }' /proc/mounts'
if [ -n "$NFSMTAB" -o -n "$SMBMTAB" -o -n "$NCPMTAB" ]; then
/etc/init.d/netfs stop
fi
fi

# IPv6 hook (pre IPv4 stop)
if [ "$NETWORKING_IPV6" = "yes" ]; then
if [ -x /etc/sysconfig/network-scripts/init.ipv6-global ]; then
/etc/sysconfig/network-scripts/init.ipv6-global stop pre
fi
fi

vlaninterfaces=""
cipeinterfaces=""
xdslinterfaces=""
bridgeinterfaces=""
remaining=""

# get list of bonding, cipe, and xdsl interfaces
for i in $interfaces; do
eval $(LANG=C fgrep "DEVICE=" ifcfg-$i)
eval $(LANG=C fgrep "TYPE=" ifcfg-$i)
eval $(LANG=C fgrep "BRIDGE=" ifcfg-$i)

if [ -z "$DEVICE" ] ; then DEVICE="$i"; fi

```

Trecho de Código 6.5.4:
/etc/rc.d/init.d/network(Continuação)

Exemplo de

```

if [ "${DEVICE##cipcb}" != "$DEVICE" ] ; then
cipeinterfaces="$cipeinterfaces $i"
unset DEVICE TYPE BRIDGE
continue
fi
if [ -n "$BRIDGE" ] ; then
    bridgeinterfaces="$bridgeinterfaces $i"
unset DEVICE TYPE BRIDGE
continue
fi
if [ "$TYPE" = "xDSL" -o "$TYPE" = "ADSL" ] ; then
    xdslinterfaces="$xdslinterfaces $i"
unset DEVICE TYPE BRIDGE
continue
fi

if [ "${DEVICE%.*}" != "$DEVICE" ] ; then
vlaninterfaces="$vlaninterfaces $i"
unset DEVICE TYPE SLAVE BRIDGE
continue
fi
remaining="$remaining $i"
unset DEVICE TYPE BRIDGE
done

for i in $cipeinterfaces $xdslinterfaces $bridgeinterfaces $vlaninterfaces; do
eval $(fgrep "DEVICE=" ifcfg-$i)
if [ -z "$DEVICE" ] ; then DEVICE="$i"; fi

if ! check_device_down $DEVICE; then
    action "Shutting down interface %s: " $i ./ifdown $i boot
fi
done

# shut down all interfaces (other than loopback)
for i in $remaining ; do
eval $(fgrep "DEVICE=" ifcfg-$i)
if [ -z "$DEVICE" ] ; then DEVICE="$i"; fi

if ! check_device_down $DEVICE; then
    action "Shutting down interface %s: " $i ./ifdown $i boot
fi
done

case "$IPX" in
yes|true)
    if [ "$IPXINTERNALNETNUM" != "0" ] ; then
        /sbin/ipx_internal_net del
    fi
    ;;
esac

```

Trecho de Código 6.5.5:
/etc/rc.d/init.d/network(Continuação)

Exemplo de


```

action "Shutting down loopback interface: " ./ifdown ifcfg-lo

if [ -d /proc/sys/net/ipv4 ]; then
    if [ -f /proc/sys/net/ipv4/ip_forward ]; then
    if [ 'cat /proc/sys/net/ipv4/ip_forward' != 0 ]; then
    action "Disabling IPv4 packet forwarding: " sysctl -n -w net.ipv4.ip_forward=0
    fi
    fi
    if [ -f /proc/sys/net/ipv4/ip_always_defrag ]; then
    if [ 'cat /proc/sys/net/ipv4/ip_always_defrag' != 0 ]; then
    action "Disabling IPv4 automatic defragmentation: " sysctl -n -w net.ipv4.ip_always_defrag=0
    fi
    fi
fi
fi
if [ -f /proc/sys/net/ipv4/tcp_syncookies ];then
    if [ 'cat /proc/sys/net/ipv4/tcp_syncookies' != 0 ]; then
    sysctl -n -w net.ipv4.tcp_syncookies=0
fi
fi

# IPv6 hook (post IPv4 stop)
if [ "$NETWORKING_IPV6" = "yes" ]; then
if [ -x /etc/sysconfig/network-scripts/init.ipv6-global ]; then
/etc/sysconfig/network-scripts/init.ipv6-global stop post
fi
fi

    rm -f /var/lock/subsys/network
    ;;
    status)
gprintf "Configured devices:\n"
echo lo $interfaces

gprintf "Currently active devices:\n"
echo '/sbin/ip -o link show | awk -F " " '"/UP>/ { print $2 }''
;;
    restart|reload)
    cd "$CWD"
$0 stop
interfaces="$active"
boot=""
$0 start
;;
    *)
    gprintf "Usage: %s\n" "$(basename $0) {start|stop|restart|reload|status}"
    exit 1
esac

exit 0

```

Trecho de Código 6.5.6: Exemplo de
/etc/rc.d/init.d/network(Continuação)

```
HOSTNAME=hufflepuff
NETWORKING=yes
GATEWAY=192.168.20.1
```

Trecho de Código 6.5.7: Exemplo de `/etc/sysconfig/network`

A idéia principal por trás de um servidor de arquivos é que monta-se uma estrutura centralizada onde arquivos importantes dentro de uma empresa possam ser armazenados. Isso provê as seguintes vantagens:

1. **Backup centralizado:** com essa estrutura, todos os arquivos importantes são centralizados, de forma que é necessário fazer o *backup* de apenas *uma* ou algumas poucas unidades de disco, o que facilita muito tanto o planejamento quanto a execução de *backups* quanto restauração;
2. **Evitar redundância:** como todos os principais documentos estão dentro do servidor de arquivos, pode-se evitar redundâncias em arquivos sensíveis, como planilhas de custo ou folhas de pagamento. Combinando uma boa estrutura de segurança de dados, como a oferecida pelo GNU/Linux, e o uso de aplicações específicos, você terá uma boa infraestrutura de dados, segura e efetiva;
3. **Facilidade na colaboração:** no caso, o uso de um servidor centralizado de arquivos permite que a colaboração entre os funcionários seja muito fácil. Ao invés de transferir arquivos via *email* ou mesmo usando DPL/DPC¹⁹, você tem um local aonde os documentos poderão ser obtidos e editados quando necessário, aumentando a colaboração e a eficiência dos sistemas;

No caso, veremos três dos principais sistemas servidores de arquivos: o NFS (*Network FileSystem* — Sistema de arquivos em rede), o FTP (*File Transfer Protocol* — Protocolo de Transferência de Arquivos) e o SMB (*Server Message Block* — Servidor de Blocos de Mensagem), utilizado no WindowsTM e implementado no GNU/Linux por meio de protocolos do *kernel* e pelo SaMBa, servidor de arquivos para Unix/Linux.

¹⁹“Protocolo” usado quando não haviam redes de computadores, chamado de *Disquete pra lá/Disquete pra cá*

6.6.1 NFS

O NFS (*Network FileSystem* — Sistema de Arquivos em Rede) é um servidor que foi criado pela Sun Microsystems como parte de seu sistema NIS (*Network Information System* — Sistema de Informações de Rede). Embora *extremamente* inseguro (possui uma quantidade enorme de falhas de segurança), possui uma grande vantagem no compartilhamento de arquivos no ambiente Unix, que é ser muito rápido quando compartilha-se arquivos Unix/Unix. O uso de outros compartilhamentos, como o SaMBa, na mesma situação, causa uma redução violenta de *performance*, embora, por incrível que pareça, seja muito rápido em transferências WindowsTM/Linux e *vice-versa*.

A estrutura do NFS é baseada em um módulo de *kernel* para cliente/-servidor NFS, outro para montagem de volumes NFS, e um servidor real, o **portmap**, que faz a tradução de requisições RPC (*Remote Procedure Call* — Chamadas de Procedimentos Remotos) em informações transmitidas via Internet. Isso é utilizado pelo sistema para traduzir os pedidos de dados e transferir os resultados. Normalmente, são instalados por pacotes com nomes como **nfs** ou **portmap**, conforme a sua distribuição. Cheque a documentação de sua distribuição para maiores informações. Os serviços NFS são inicializados (ou “levantados”, como é chamado o processo no jargão do mundo Unix). Veremos um pouco mais sobre inicialização no Capítulo 10, na verdade na Seção 10.1, na Página 217.

6.6.1.1 Configurando um *share* NFS

A configuração de um compartilhamento, ou *share*, é feita de maneira muito simples, envolvendo apenas um arquivo, o **/etc/exports**.

Vamos nos pegar novamente da Figura 6.2, tendo sua estrutura descrita na forma de um **/etc/hosts** indicado na Figura 6.3.4. Como visto, o servidor **ravenclaw.hogwarts** também tem um *alias* chamado **data.hogwarts**. No caso, imaginemos que o administrador da rede libere uma pasta em **/home/arquivos** dentro de **ravenclaw.hogwarts** como um *share* NFS acessível por todos os *hosts* da rede **hogwarts**. Nesse caso, utilizamos uma linha simples no arquivo **/etc/exports**:

```
/home/arquivos 10.0.0.*(rw)
```

Essa linha também poderia ser escrita assim:

```
/home/arquivos *.hogwarts(rw)
```

Claro, considerando que o arquivo `/etc/networks` esteja corretamente configurado.

A entrada é descrita por **(1)** diretório compartilhado, **(2)** redes ou *hosts* que podem acessar o *share* em questão e **(3)** opções.

Por exemplo, imaginemos que a máquina `dumbledore.hogwarts` deseja compartilhar uma pasta `/etc/info`, mas de maneira que ninguém escreva no diretório em questão por meio do NFS. Para isso, ele utiliza a seguinte entrada em seu `/etc/exports`:

```
/etc/info *.hogwarts(ro)
```

Algumas opções úteis, sendo que você sempre poderá consultar a *man-page exports(5)*, são:

- **async**: essa opção faz com que os arquivos sejam transmitidos de maneira assíncrona, aproveitando melhor os momentos de rede ociosa, otimizando o uso de rede, mas incorrendo no risco de corrupção de arquivos. *Shares* de disco com apenas-leitura (**ro**) ou em redes de alta confiabilidade são bons lugares para o uso de **async**;
- **root_squash**: em ambientes com acesso remoto via rede, um dos maiores riscos do acesso via NFS é a característica do NFS de mapear os acessos de arquivo em relação ao usuário *local*, de modo que as permissões de acesso aos arquivos remotos são baseados no usuário e grupo *local* do usuário. Se você imaginar uma partição *root(/)* compartilhada e acessada por um usuário remoto logado em sua máquina local como **root** dá margem a todo tipo de problemas de segurança, como roubos de senhas, violações de acesso e afins. Para esses caso, o NFS prevê a opção **root_squash**, que “transforma” o *root* em um usuário comum chamado *anonymous*, do grupo *anonymous*, o que impede o acesso a arquivos com permissões restritas por um *root* remoto;

- **all_squash**: é similar a **root_squash**, mas com maior amplitude, onde *todos* os usuários remotos que acessam arquivos via NFS passam a ser considerados *anonymous*;
- **anonuid** e **anongid**: essas opções são interessantes para os casos de usar opções como **root_squash** e **all_squash**, pois permite configurar quais são os usuários *locais* que serão utilizados pelos usuários remotos que caíam em ambas as opções para o compartilhamento em questão.

Isso pode ser útil para facilitar a administração de ambientes, principalmente no caso de *shares* para compartilhamento conjunto de dados. Isso porque o NFS continua a obedecer as permissões definidas para os arquivos no acesso remoto. Ou seja, se o usuário remoto não tiver acesso aos arquivos compartilhados após a tradução de usuário, ele não irá acessar os arquivos. Definindo **anonuid** e **anongid** e configurando corretamente as permissões de arquivo, você garante que os arquivos serão acessados corretamente por qualquer um no caso dos compartilhamentos públicos.

Uma vez tudo configurado, utilize o comando **exportfs -av** para liberar os *share* definidos sem precisar reiniciar (fisicamente!!) o servidor de arquivos ou reiniciar o **portmap**. Outro comando útil é o **showmount**, que mostra quais são os *shares* que estão sendo acessados e por quem. Ele não mantém registros históricos, mas nada que um *script* não resolva.

6.6.1.2 Acessando um *share* NFS

Para acessar o *share* remoto, é muito simples: basta que o *kernel* tenha habilitado dentro dele o *filesystem* NFS, seja internamente ou na forma de módulo e usar o comando **mount**, indicando o IP do *host* que está oferecendo o *share* e o diretório do *share* em questão. Por exemplo, se quisermos acessar o *share* que foi configurado anteriormente em **ravenclaw.hogwarts**, podemos usar o comando:

```
mount -t nfs 10.0.0.3:/home/arquivos /mnt/share_nfs
```

ou

```
mount -t nfs ravenclaw.hogwarts:/home/arquivos /mnt/share_nfs
```

Considerando as mesmas regras para qualquer montagem de dispositivo, como ter o diretório do ponto de montagem criado e que os dados naquele arquivo estarão indisponíveis enquanto o dispositivo montado estiver montado, sendo liberado após a desmontagem do *share*. A desmontagem é simples, usando o comando `umount`.

Algumas opções na montagem, como `users`, `auto` e `exec` estão disponíveis, o que torna o NFS útil para compartilhamento de sistemas grandes, como `TeX`, `LATeX` e `EMACS`. Algumas opções úteis e específicas para o NFS são:

- **soft**: essa opção não “trava” o programa no caso de um acesso a um *share* cujo servidor esteja fora do ar ou com os serviços “derrubados” (não carregados). No caso de acontecer isso em um *share* montado com a opção **soft**, o sistema irá enviar mensagens como “arquivo não localizado” e afins.
- **rsiz**e: Aumenta o *buffer* de leitura do NFS. Essa opção causa problemas em *shares* com NFS versão 2 como servidor, mas nos *shares* com NFS versão 1 pode aumentar a *performance* do ambiente.
- **wsiz**e: Aumenta o *buffer* de escrita do NFS. Essa opção causa problemas em *shares* com NFS versão 2 como servidor, mas nos *shares* com NFS versão 1 pode aumentar a *performance* do ambiente.

Aqui é importante uma ressalva: da mesma forma como o de quaisquer dispositivos montados por meio do `mount`, as configurações dos *shares* pode ser gravado no `/etc/fstab`, conforme mostrado na Seção 4.4, Página 86. Por exemplo, se quisermos que o *share* de `ravenclaw.hogwarts` definido anteriormente, sendo que poderá ser montado automaticamente no *boot* e com acesso por usuários, evitando problemas de localização de arquivos no caso de queda do servidor NFS, basta inserir a seguinte linha de NFS no `/etc/fstab`:

```
10.0.0.3:/home/arquivos /mnt/share_nfs nfs users,auto,soft 0 0
```

Isso é o suficiente sobre NFS nesse documento. Uma boa pesquisa na Internet poderá fornecer muito mais explicações sobre o NFS.

6.6.2 FTP

O NFS é um bom sistema para ambientes Unix quando precisa-se compartilhar arquivos remotos para acesso imediato (ou seja, tem que estar disponível de maneira automática). Mas existe alguns problemas:

1. O NFS não é normalmente acessível via sistemas WindowsTM, sendo normalmente exigidos produtos de terceiros para habilitar essa funcionalidade no mesmo (ambientes MacOS também sofrem desse problemas nas versões anteriores ao MacOS X);
2. Como dito, o NFS é *muito* inseguro, além de ter algumas complexidades na questão de configuração dos ambientes local e remoto e o acesso às permissões de arquivo;
3. Algumas vezes, não é necessário manter-se um *share* montado permanentemente, principalmente no caso de arquivos que são copiados de/para a máquina local para serem trabalhados;

Para isso, existem sistemas como o FTP (*File Transfer Protocol* — Protocolo de Transferência de Arquivos), que permitem que arquivos sejam deslocados de/para máquinas remotas antes/depois de serem trabalhados na máquina local. A principal utilidade disso é manter um *backup* remoto que possa ser acessado quando necessário, ao mesmo tempo sem precisar de uma conexão permanente, aproveitando a rede o melhor possível.

Uma vantagem é a questão de que, como os comandos FTP são configuráveis, pode-se definir *scripts* que realizem *backup* de arquivos de uma máquina local sem a intervenção (ou mesmo sem conhecimento) do usuário, principalmente no ambiente Unix/Linux, com a combinação de *scripts* e do sistema **cron** para execução de tarefas agendadas.

Outra vantagem é que quase todos os sistemas operacionais possuem tanto clientes quanto servidores FTP em suas plataformas. No Windows, o próprio Windows Explorer pode ser usado como um cliente FTP rudimentar. Uma sugestão melhor são clientes especializados como o CuteFTP e *plugins* como o FireFTP, que transforma o Firefox em cliente FTP. Para servidor, pode-se usar o FileZilla, um projeto *free software* que oferece um servidor simples de configurar e usar para o ambiente WindowsTM.

No caso do Unix/Linux, você pode usar como cliente o comando `ftp`, que é parte dos comandos do padrão POSIX (ou seja, deve ser incluído em qualquer Unix “de respeito”). Como servidor, existem vários, sendo que no caso iremos falar do ProFTPD, aproveitando o material incluído em Morimoto (2006). No caso da maioria das distribuições, o servidor pode ser implementado usando os pacotes `proftpd` incluídos com elas. Cheque a documentação de sua distribuição para maiores informações, ou dê uma olhada no Capítulo 9, na Página 216.

A maioria das distros configura o ambiente do ProFTPD para rodar em modo *standalone*. Ele é considerado mais seguro, pois o servidor fica ativo o tempo todo. Outra opção é utilizar o “super-servidor” `inetd`, que permite que (1) o servidor seja carregado apenas quando necessário e que (2) no caso de suspeita de invasão, possa-se usar *wrappers* que chequem o tipo de dados trafegado por meio da conexão FTP.

A grande desvantagem do FTP, sem sombra de dúvidas, é o fato de ele ser um verdadeiro pesadelo para a configuração de um *firewall*, uma vez que, na verdade, o FTP mantém DUAS conexões abertas ao mesmo tempo, uma conexão chamada de *conexão de controle*, que acessa o servidor pela porta normal, e a outra, a *conexão de dados*, que é negociada e estabelecida no momento em que o servidor FTP confirma a conexão com o cliente FTP. Veremos mais sobre isso na Seção 6.7, na Página 185, portanto não iremos discutir isso aqui.

Outra desvantagem é que o FTP é bastante inseguro se mal configurado. A maioria dos servidores roda em um ambiente aberto, aonde a pessoa pode copiar e acessar arquivos em qualquer lugar dentro do servidor (desde que ele possua permissões normais). Isso não é lá muito seguro, principalmente considerando-se falhas de escalada de privilégios²⁰. Existem várias formas de configurar-se um bom ambiente FTP, usando um ambiente `chroot`²¹, mas eles exigem alguma preparação, principalmente quanto a arquivos locais e espelhamento de diretórios e não iremos tratar sobre isso aqui, sendo que

²⁰falhas de segurança que permitem que o usuário dispare comandos arbitrários ou consiga uma *shell* de um usuário privilegiado, normalmente o *root*

²¹O nome deriva do comando `chroot`, criando uma estrutura à parte, onde um determinado diretório é definido como o *root(/)* do ambiente, sendo que no caso do FTP apenas utilitários e arquivos abaixo do diretório definido como *root* podem ser acessados pelo usuário no FTP

existem muitos tutoriais na Internet sobre isso. Algumas informações básicas podem ser encontrado na *infopage* `coreutils`, no *node* `chroot` (`info coreutils chroot`).

Bem, esclarecido isso, vamos passar à configuração e acesso do servidor FTP.

6.6.2.1 Configurando um servidor FTP

A configuração do ProFTPD é acessado por meio do arquivo `/etc/proftpd.conf`. Esse arquivo de configuração é simples, sendo apresentado um exemplo no Trecho de Código 6.6.1, na Página 166, sendo o exemplo baseado em Morimoto (2006).

Vamos analisar o arquivo com calma. A primeira linha, `Port`, indica a porta que o FTP vai “ouvir” (ou seja, atender requisições). O padrão oficial é 21/TCP (porta 21 em protocolo TCP), mas muitos provedores de acesso à Internet podem bloquear essas portas para seus clientes, de modo que estes não mantenham servidores. Nesses casos, mude a porta padrão nessa linha e lembre-se de a indicar na conexão.

A linha seguinte, `MaxInstances`, permite limitar o número de conexões simultâneas ao servidor FTP. Em conjunto com a opção `TransferRate` mostrada pouco abaixo da mesma, permite controlar o uso de banda pelas conexões.

O uso da linha `DefaultRoot` indica que os usuários do FTP só poderão acessar seus diretórios *home*, de forma que esse comando atua como um `chroot` bastante flexível, o que impede, além de usuários acessar arquivos sensíveis do servidor, que usuários acessem arquivos de outros usuários. De certo modo, a única forma de “violiar” essa opção é mediante o uso de ligações *hard*, mas mesmo isso deve ser evitado.

A opção `TransferRate` permite que seja estabelecido o tamanho de banda de passagem a ser usada por conexão FTP, *por usuário*. No exemplo em questão, o `TransferRate` é de 8 KB/s por usuário, se imaginarmos as 30 conexões, conseguiremos um consumo total de banda de 240 KB/s.

A estrutura `<Anonymous ftp>` estabelece um diretório para *login* FTP anônimo. No caso, optou-se pelo *home* do usuário `ftp`, o que permite uma

```
Port                21
MaxInstances        30
DefaultRoot         ~
TransferRate        RETR 8:10
<Anonymous ~ftp>
  User              ftp
  Group             nogroup
  UserAlias         anonymous ftp
  DirFakeUser       on   ftp
  DirFakeGroup      on   ftp
  RequireValidShell off
  MaxClients        20
  DisplayLogin      welcome.msg
  DisplayFirstChdir .message
  <Directory *>
    <Limit WRITE>
      DenyAll
    </Limit>
  </Directory>
  <Directory incoming>
    Umask            022 022
    <Limit READ WRITE>
      DenyAll
    </Limit>
    <Limit STOR>
      AllowAll
    </Limit>
  </Directory>
</Anonymous>
```

Trecho de Código 6.6.1: Exemplo de `/etc/proftpd.conf`

administração facilitada. As linhas abaixo, **User** e **Group**, define qual o usuário e grupo que será tratado a conexão anônima para efeito de permissões de arquivos. No caso, utiliza-se **User ftp** e **Group nogroup**, que são uma boa combinação de usuário e grupo anônimos para efeito de FTP, além de permitir uma fácil administração, principalmente no quesito permissões de arquivo.

A linha **UserAlias** indica apelidos a serem usados na conexão remota anônima. Na verdade, podemos dizer que são *pseudos-usuários* que serão roteados à conexão anônima pelo servidor FTP. Esses pseudos-usuários não precisam estar cadastrados no sistema.

A linha **RequireValidShell** obriga ou desobriga o usuário das conexões anônimas (no caso, **ftp**). No caso, com a **RequireValidShell off** desobrigamos o usuário em questão de ter um *shell* válido, o que é uma forma de aumentar a segurança: caso um invasor consiga, por algum motivo, “vazar” o **chroot** e conseguir o arquivo de senhas do servidor, e tentar se conectar por outros serviços, um *shell* especial, falso, pode ser associado tranquilamente o usuário **ftp**, impedindo potenciais ataques.

A linha **MaxClients** determina o número de conexões simultâneas que o sistema pode atender naquele tipo de conexão. Por exemplo, **MaxClients 20** para a seção **<Anonymous>** indica que no máximo 20 conexões anônimas poderão ser estabelecidas. Esse número de conexões é independente de **MaxInstances**, sendo normalmente menor (até porque ele irá recusar mais conexões que **MaxInstances**, independente do que aconteça). Uma sugestão é deixar algumas conexões extras para usuários não-anônimos, caso seu FTP venha a ter essa funcionalidade.

A linha **DisplayLogin** indica um arquivo de texto que será exibido para o usuário como mensagem de boas vindas. Ele sempre fica dentro do diretório em questão (no caso **/home/ftp**) e seu nome é sempre relativo a esse diretório. Por exemplo **DisplayLogin welcome.msg** mostra o arquivo **/home/ftp/welcome.msg**.

A seguir, temos as seções **<Directory>**, e dentro dele as seções **<Limit>**. Essas seções permitem que sejam configuradas restrições para acesso conforme o tipo de acesso. Os tipos em questão normalmente são **READ** (leitura), **WRITE** (escrita) e **STOR** (armazenagem remota). Em geral, as permissões por padrão são as mesmas do sistema de arquivos, mas podem

ser modificadas por `DenyAll` (negando para todos). Na seção `<Directory incoming>`, porém, ele já autoriza com `<Limit STOR>` e `<AllowAll>` que qualquer um grave arquivos nesse diretório, embora ele vá com permissão para que apenas o usuário do FTP consiga escrever nesses arquivos. Esse é um bom esquema de *backup*, permitindo que os usuários depositem arquivos que sejam importantes dentro do servidor de forma que processos de *backup* em fitas ou outras mídias específicas precisem recorrer a apenas um local para acesso.

Essa é uma configuração simples e rápida de servidor FTP ProFTPD. Na Internet você poderá obter informações mais especializadas, conforme suas necessidades. Além disso, Morimoto (2006) é uma ótima referência para isso.

6.6.2.2 Acessando um servidor FTP

Diferentemente do caso do NFS e do SaMBa, no FTP você **não** consegue montar um FTP como um dispositivo. Em compensação, você consegue acessar o servidor FTP com clientes disponíveis em todas as plataformas e modo, seja gráfico ou texto. É possível inclusive utilizar-se o FTP por meio de um *script* e automatizar tarefas de *backup*.

Por exemplo, você pode usar o *script* do Trecho de Código 6.6.2, na Página 169, para automatizar o “upload” de um arquivo de *backup* chamado `/usr/backup/backup.tar.bz2` no servidor `myserver`, na porta 2121, usando um usuário genérico *backup-client* com senha *backup*, previamente configurado.

Esse tipo de *backup* pode ser configurado com facilidade, o que ajuda bastante a administração. O *script* do Trecho de Código 6.6.2, na Página 169, pode ser facilmente corrigido para buscar arquivos de *backup* previamente gerados por outros *scripts* e enviado ao servidor em questão.

Você também pode recorrer a clientes como o FireFTP, o CuteFTP (para Windows) e o Midnight Commander, um gerenciador de arquivos para Unix com possibilidade de acessos FTP. Procure se informar conforme a sua plataforma. Todos os clientes permitem “subir” e “descer” arquivos para o servidor, conforme as configurações e permissões do servidor.

Porém, o mais interessante é conectar-se via modo texto. Para isso, em

```
#!/bin/bash

ftp <<<ENDFTP
open myserver 2121
backup-client
backup
lcd /usr/backup
cd /backup
binary
put backup.tar.bz2
bye
ENDFTP
```

Trecho de Código 6.6.2: Exemplo de *script* com conexão FTP

um *shell* ou Terminal (no Unix) ou em um Prompt de Comando (no Windows XP™) chamando o comando `ftp`. Você normalmente receberá um *prompt* como o seguinte:

```
ftp>
```

Uma vez nele, basta utilizar o comando `open`, seguido pelo servidor e porta para estabelecer a conexão. Depois, basta dar um *login* e senha específico no servidor, ou então utilizar a senha *anonymous* ou *guest* tendo como senha um *email* válido (qualquer um) para entrar no FTP. Dentro dele, você poderá usar os comandos da Tabela 6.3, na Página 170. Esses comandos podem ser usados em quaisquer clientes texto de FTP, e também podem ser usados em *scripts* de automação de *backup* usando FTP, como o do Trecho de Código 6.6.2, na Página 169.

6.6.2.3 Modos de transferência no FTP

Existe uma pegadinha *muito* séria no FTP: antigamente, as linhas de transmissão de dados eram **muito** lentas, sendo que mesmo grandes faculdades tinham *links* dedicados de por volta de 1200 bps. Nesses casos, transferir documentos de texto era pouco eficiente sem uma compressão decente.

Para facilitar isso, o FTP estabeleceu dois modos, o *ASCII* e o *binário*.

COMANDOS	O QUE FAZ?
open	Não é <i>realmente</i> enviado ao servidor remoto, sendo na realidade uma solicitação para uma abertura de conexão ao servidor dado como parâmetro e (caso passado) à porta TCP desejada
ls	Mostra os arquivos dentro do diretório remoto onde a pessoa se encontra
pwd	Informa o diretório onde o usuário se encontra no servidor
cd	Entra em um diretório no servidor remoto
lcd	Muda o diretório na máquina local
get	Copia o arquivo da máquina remota para a máquina local
mget	Copia vários arquivos da máquina remota
put	Copia o arquivo da máquina local para a máquina remota
mput	Copia vários arquivos para máquina remota
binary	Define o modo de transferência binário (Importante para garantir que o arquivo não seja corrompido)
ascii	Define o modo de transferência de texto puro (Importante para garantir que o arquivo não seja corrompido)
bye	Desconecta-se do servidor remoto

Tabela 6.3: Comandos FTP

No modo *ASCII*, o *hardware* e/ou o servidor realizam uma compressão de dados eliminando alguns *bits* normalmente não importantes em arquivos de texto puro, além de interpretar certos *bits* e caracteres como de controle.

A pergunta que você pode estar se fazendo é: “*e no que isso pode afetar a transferência de dados?*” Em transferências de arquivos ASCII que não possuam acentos ou caracteres especiais, como o símbolo de Euro (chamados de ASCII 7-bit, por seus caracteres estarem todos incluídos nos primeiros 7 bits da tabela ASCII — caracteres ASCII de 0 a 127), na verdade nada, e aumenta significativamente a transferência.

Mas no caso de dados binários e de arquivos ASCII que possuam caracteres acentuados as coisas começam a ficar sérias, pois, caso hajam *bytes* no arquivo que possam ser interpretados como caracteres de controle, existe uma grande chance de ocorrerem situações imprevisíveis na transferência do arquivo, o que *invariavelmente* irá resultar em corrupção de dados!

Nesse caso, se desejar maior segurança, antes de qualquer transferências de/para o servidor FTP, utilize o comando `binary` para definir a transferência como modo binário. Se tiver certeza de estar enviando apenas arquivos de texto *ASCII-7 bit*, utilize o comando `ascii` para acelerar a transferência de dados.

Se você tiver dúvidas, você pode recorrer a duas opções: a primeira, válida em qualquer cliente/servidor FTP, é definir o modo para binário com o comando `binary`. Você vai “perder” performance no caso dos arquivos que poderiam ser transmitidos com segurança por meio do modo ASCII, mas irá garantir que não irá ter problemas. A segunda, aceita pela maioria (mas não por todos) os clientes/servidores FTP, é utilizar um modo automático, onde o cliente irá se encarregar de detectar a presença de caracteres estranhos e irá ele próprio decidir qual o melhor modo de transferência. Se seu cliente possuir essa opção, mantenha-a ativa, para evitar dores de cabeça.

Isso deve ser o suficiente sobre FTP. Passemos a um servidor de arquivos muito importante no mundo GNU/Linux, paradoxalmente por permitir a comunicação com servidores do mundo proprietário do Windows: o SaMBa.

6.6.3 SMB (SaMBa)

Já vimos o NFS e o FTP. Mas uma das principais soluções de compartilhamento de arquivos nos ambientes de rede atualmente, é o sistema de compartilhamentos (*shares*) WindowsTM, até mesmo devido à popularidade²² pela alta penetração desse sistema operacional. Esse tipo de compartilhamento é normalmente ativado em clientes Windows 9x ou melhor, além de clientes Windows for Workgroups 3.11 (caso encontre algum). Ele utiliza um protocolo conhecido como SMB (*Server Message Block* — Servidor de Bloco de Mensagens), desenvolvido para atuar em cima do antigo protocolo NetBIOS. Nas versões mais atuais, a partir do Windows 9x, o SMB foi adaptado para trabalhar com o TCP/IP. Devido à alta penetração do SMB, ficou claro que grandes instalações baseadas em ambiente heterogêneo poderiam utilizar o SMB como um protocolo comum para compartilhamento de arquivos nos moldes do NFS.

O protocolo SMB, porém, é de propriedade da Microsoft, sendo que seu suporte a Unix era caro e restrito a alguns de seus “sabores”. Com isso, Andrew Tridgell desenvolveu um pacote GPL de suporte ao SMB, chamado SaMBa. Originalmente foi desenvolvido para Solaris, sendo que com o passar do tempo, ele foi portado para o Linux. O SaMBa consegue emular perfeitamente um servidor Windows para compartilhamento de arquivos e autenticação em redes WindowsTM. Sua popularidade aumentou rapidamente baseado em seu custo baixo e na sua velocidade, em vários casos, maior que a dos servidores Windows reais.

Em sua versão mais estável, o SaMBa ainda possui problemas para atuar como um servidor parte da estrutura *Active Directory*TM. Na versão 4.0, está prometida a possibilidade do SaMBa atuar como parte da estrutura *Active Directory*TM, através de um mini-servidor LDAP/Kerberos capaz de emular o servidor *Active Directory*TM do sistema.

6.6.3.1 Configurando um *share* SaMBa

Você irá precisar de dois pacotes, o **samba** (ocasionalmente chamado de **samba-server**), e o **smbclient** (ocasionalmente chamado de **samba-client**).

²²N.A.: Não entrarei no mérito de se a mesma é válida ou não, justa ou não, preferindo apenas reconhecer sua popularidade.

Além deles, outros pacotes úteis são o **samba-doc** (documentação do SaMBa) e o **swat**, uma ferramenta gráfica de configuração do SaMBa via Web. Cheque a documentação de sua distribuição para maiores informações, ou dê uma olhada no Capítulo 9, na Página 216.

Esses pacotes poderão forçar a instalação de um pacote **samba-common**, com arquivos e configurações comuns a todos os pacotes do SaMBa. Não se preocupe, pois isso é necessário. Você pode se interessar pelo *Samba Web Administration Tool*, o **swat**, uma ferramenta *web-based* que poderá lhe auxiliar na administração do SaMBa. Procure por pacotes com o nome **swat** ou **samba-swat**. Se você tiver sorte e sua distribuição trazer a mais nova versão do SaMBa, não precisará instalar manualmente o **swat**, já que ele faz parte do pacote básico do SaMBa em sua mais nova versão.

Uma vez instalado os pacotes, você precisará editar o arquivo `/etc/samba/smb.conf`. Muitas distros trazem um arquivo que é parte dos exemplos do SaMBa, e que tende a ser **muito bem** documentado, além da grande quantidade de tutoriais e configurações possíveis de serem encontradas na Internet ou em livros como Morimoto (2006), inclusive passando para o SaMBa a possibilidade de atuar como um PDC (*Primary Domain Controller* — Controlador Primário de Domínio), e com isso oferecer um ambiente de *login* de redes Windows centralizado, com alta performance e baixo custo. Vamos portanto nos deter em alguma configuração básica como fizemos no caso do NFS. Vamos estudar rapidamente o arquivo exemplo do Trecho de Código 6.6.3, da Página 174, livremente adaptado de Morimoto (2006) e Ferreira (2003).

A primeira coisa a notar é que o arquivo possui uma estrutura similar à dos arquivos `.ini` do Windows, dividido em seções indicadas pelo nome dentro de colchetes (`[]`), e que os valores são colocados linha por linha, com o nome do atributo separado do valor por iguais (`=`). Além disso, existem duas formas de comentar linhas nesse arquivo, para desativar opções ou para escrever documentações internas no arquivo: a forma Windows, onde os comentários são marcados por ponto-e-vírgula (`;`), e a forma Unix/Linux, usando a cerquilha ou *sharp* (`#`). Agora, vamos começar a analisar o exemplo do Trecho de Código 6.6.3, da Página 174. Esse arquivo está *sem* opções para compartilhamento de impressoras, pois isso foge do assunto desse capítulo. Consulte Morimoto (2006) para maiores informações sobre o compartilhamento de impressoras no Linux. Ferreira (2003) traz também um

```

[global]
    workgroup = mygroup
    netbios name = myserver
    server string = Samba Server %v
    announce as = NT Server
    message command = /usr/bin/linpopup "%f" "%m" %s; rm %s
    log file = /var/log/samba/%m.log
    max log size = 50
    map to guest = bad user
    security = user
    encrypt passwords = yes
    smb passwd file = /etc/samba/smbpasswd
    unix password sync = Yes
    pam password change = yes
    passwd chat = *New*UNIX*password* %n\n *Re*type*new*UNIX*password* %n\n \
*passwd:*all*authentication*tokens*updated*successfully*
    username map = /etc/samba/smbusers
    socket options = TCP_NODELAY SO_RCVBUF=8192 SO_SNDBUF=8192
    dns proxy = no
    logon drive=H:
    logon home=\\%L%\U\profiles
    logon path=\\%L\profiles\%U
    logon script=%U.bat

[homes]
    comment = Home Directories
    browseable = no
    writable = yes

[netlogon]
    comment = Network Logon Service
    browseable = no
    path = /var/samba/netlogon
    public = no
    guest ok = no
    writeable = no
    shareable = no
    share modes = no
    availble = yes

[public]
    comment = Public Stuff
    path = /home/samba/public
    public = yes
    writable = no
    write list = @staff

[fredsdird]
    comment = Fred's Service
    path = /usr/somewhere/private
    valid users = fred
    public = no
    writable = yes
    printable = no

[myshare]
    comment = Mary's and Fred's stuff
    path = /usr/somewhere/shared
    valid users = mary fred
    public = no
    writable = yes
    printable = no
    create mask = 0765

```

Trecho de Código 6.6.3: Exemplo de /etc/samba/smb.conf

capítulo sobre a configuração do SaMBa, inclusive com detalhes sobre como o configurar como PDC, BDC (*Backup Domain Controller* — Controlador de Domínio Secundário)

6.6.3.2 A seção [global]

A primeira seção em qualquer arquivo `smb.conf` é a seção [global], onde uma série de configurações-padrão para o servidor SaMBa são implementadas. Nessa seção, a primeira linha deve ser `workgroup`, que indica o grupo de trabalho/domínio ao qual o servidor faz parte. Se um ambiente DNS estiver configurado e essa linha não estiver presente, o grupo de trabalho/domínio será definido por *default* no SaMBa como o nome de domínio DNS do servidor em questão (por exemplo `hogwarts`, se imaginarmos a rede da Figura 6.2, Página 139).

Outra opção importante é a `netbios name`, que indica o nome da máquina em questão dentro do grupo de trabalho/domínio em questão (passaremos a usar o termo grupo de trabalho ou *workgroup* para facilitar a vida). Essa linha indica como o servidor SaMBa será conhecido dentro de uma rede Windows. Perceba que esse nome **pode ser diferente do *hostname* da GNU/Linux**. Por *default*, ele passa a ter o mesmo nome do *hostname* da máquina (a parte inicial do FQDN da máquina, como `hufflepuff` em `hufflepuff.hogwarts`, se imaginarmos a rede da Figura 6.2, Página 139), mas pode ser configurado para qualquer nome desejado. Isso é muito útil para migrações onde deseja-se que o sistema continue sendo “enxergado” normalmente após migrar-se um *host* Windows para o GNU/Linux com SaMBa: basta mudar o `netbios name` para o mesmo da antiga máquina.

Atenção: Isso deve ser feito *apenas* com o *host* Windows desconectado da rede, senão haverá um conflito de nomes NetBIOS (nome dos servidores na rede Windows) que provocará instabilidades e poderá (e provavelmente irá) retirar a rede Windows (ou ao menos os servidores em questão) do ar.

A opção `server string` é apenas uma descrição que será exibida sobre aquele servidor no “Meus locais de rede” no Windows e na lista de impressoras. Pode ser configurada para qualquer texto sem problemas, conforme

suas necessidades, como “Servidor de Arquivos do Departamento Financeiro” ou “Impressoras corporativas de grande porte” sem problema algum.

A opção `announce` *as* indica como o servidor SaMBa será “reconhecido” na rede heterogênea. Como conselho, é interessante manter essa opção em seu padrão, `NT Server`, e apenas utilizar outras opções se for adotar a opção `security` como `share` (mais sobre isso adiante). Nesse caso, você pode colocar essa opção em `Win95` que deve ser o suficiente. Outras opções são `NT Workstation` e `WfW` (Windows for Workgroups). Essa opção, se fora do seu padrão, irá impedir o servidor SaMBa de ser enxergado como um servidor SMB.

A linha `message command` indica um comando que deverá ser executado localmente quando o servidor receber uma mensagem pelas Notificações do Windows (*WinPopup*). Segundo a *manpage* do arquivo de configuração do SaMBa ele deixa claro que *o comando deverá retornar imediatamente*, sem “trancar terminal”. Por isso, em comandos que envolvam interface gráfica, é importante colocar o “E comercial” (&). Caso contrário, você poderá congelar seu sistema. Para mais sobre isso, a documentação na *manpage* do arquivo `/etc/smb.conf(5)` possui muitas orientações.

A linha `log file` determina o local aonde ficarão gravadas as informações de *log* do sistema. No caso, percebe-se que a opção no nosso arquivo de exemplo está definida como `/var/log/samba/%m.log`. O `%m` é “parseado” para (ou seja, equivale ao) o nome da máquina cliente. Portanto, cada máquina da rede tem um *log* diferente no SaMBa.

A opção `map to guest` é uma forma que o SaMBa encontra de mapear usuários que tenham entrado *logins* e senhas inválidas. Para isso, ele utiliza uma conta especial `guest`, com baixas permissões, da mesma forma que no caso do FTP. A opção `default` no SaMBa é `never` (ou seja, nunca permitir que usuários com *login* ou senha inválidos acessem os *shares* do servidor), mas um mapeamento *bad user* pode ser uma boa estratégia em servidores sem informações críticas. Nessa opção, caso o usuário tenha um *login* ou senha inválidos, ele é mapeado para esse usuário `guest`, o que pode ser útil para ambientes de recuperação e/ou acesso público.

A opção seguinte, `security`, é *a mais importante* do SaMBa. Ela define como o SaMBa irá tratar a segurança de acesso do sistema. Nas versões acima da 2.0, o valor *default* é `user`. Nas versões abaixo da 2.0, o padrão

(na verdade, *o único modo disponível é share*).

Existem basicamente 4 valores diferentes nessa opção:

- **share(Compartilhamento):** o *login* fica basicamente definido por senhas especiais em cada compartilhamento, definidas na descrição dos *share*. Embora tenha caído em desuso, pode ser útil para *shares* cujo acesso público, embora restrito, possa ser útil (por exemplo, um local com instaladores e *updates* de programas usados nas estações que possa ser acessado pelas equipes de manutenção);
- **user(Usuário):** o *login* é feito baseando-se em usuários, da mesma maneira que é feita em pequenas redes Windows 98 ou XP. É o padrão do SaMBa 3.0 e permite que o SaMBa possa atuar como um PDC ou BDC. Uma nota importante aqui é que cada usuário do SaMBa deve ter um usuário no GNU/Linux. Esse usuário, porém, *poderá ter senha travada* para impedir o *login* remotos via terminais, se necessário, uma vez que as senhas do SaMBa ficam em outro arquivo. Isso apenas é necessário porque o SaMBa utiliza a estrutura de acesso de arquivos do Unix/Linux. Isso vale para **todos** os modos, mesmo para o modo **share**;
- **domain(Domínio):** nesse padrão, o SaMBa torna-se de certa forma dependente de um PDC, atuando apenas como servidor de arquivos e repassando as autenticações de usuário (parte do processo de acesso a arquivos do SaMBa) para o PDC;
- **server(Servidor):** Ele mistura os dois modelos de segurança anterior: primeiro, ele verifica se um PDC está ativo na rede Windows. Se tiver, ele irá utilizar esse servidor para realizar a autenticação do usuário. Caso contrário, ele assume a responsabilidade de fazer a autenticação de rede;

A opção seguinte, **encrypt passwords** é usada quando desejamos que as senhas do SaMBa circulem encriptadas pela rede.

Aqui está umas das maiores dificuldades na configuração do SaMBa: estações e servidores Windows NT 4.0 com *Service Pack 2* (SP2) ou melhor, assim como estações Windows 98 ou melhor, *esperam por default* que as

senhas sejam encriptadas. Por sua vez, estações abaixo dessas trabalham com o envio de senhas em *clear text*²³

O grande problema aqui é que no caso de ambos, uma vez que eles definam um *default*, é necessário ativar uma chave de Registro nas estações de modo a garantir que o padrão do servidor será aceito. Essas configurações estão disponíveis em arquivos *.reg* distribuídos com o SaMBa. Por segurança, opte por ativar a chave nas máquinas que pedem senhas *clear text* (estações Windows 95 ou Windows for Workgroups 3.11) e mantenham essa chave de configuração no padrão (*encrypted passwords=yes*)

A opção *smb passwd file* indica aonde o SaMBa irá buscar e armazenar as entradas de senha para as estações e usuários do SaMBa. A configuração do SaMBa é bastante complexa quanto a usuários e senhas, mas o que você precisa saber *de imediato* é:

1. Cada usuário e máquina deve ter um *login* no sistema do Linux. Esse *login* **PODE** estar bloqueado ou desabilitado no GNU/Linux, o que pode ser útil no caso de usuários que não deverão ter acesso ao ambiente GNU/Linux via terminal, Telnet ou *Secur Shell* SSH. Isso pois o SaMBa irá utilizar a estrutura de permissões de acesso do GNU/Linux para liberar o acesso aos arquivos, sem criar uma estrutura própria. Portanto, os usuários precisarão “existir” para o sistema de permissão de arquivos do GNU/Linux. Se eles são válidos ou não pouco interessará: basta que eles existam;
2. As senhas de acesso ao SaMBa *não precisam ser as mesmas* do ambiente GNU/Linux. Se você desejar, pode habilitar uma senha para um usuário no servidor GNU/Linux e uma senha diferente para o SaMBa. Para habilitar uma máquina/usuário no SaMBa, devemos usar o utilitário *smbpasswd*, incluído no SaMBa. Trataremos disso na Seção 6.6.3.6, na Página 185. Esse utilitário deve ser configurado para manipular esse arquivo, de modo que as senhas do SaMBa fiquem corretas;

O padrão do SaMBa é */etc/samba/smbpasswd*, podendo ser definido para qualquer valor que você desejar.

²³**Texto claro:** é o termo usado quando dados são enviados de/para o servidor sem que o mesmo recorra a nenhum tipo de criptografia.

A opção `unix password sync` define se o SaMBa irá atualizar as senhas entre seu próprio ambiente de senhas e as senhas do Unix. Isso pode ser útil em alguns casos específicos, como no caso de servidores que possam ser usados para desenvolvimento ou ambientes envolvendo CVS²⁴ ou Subversion²⁵, onde as senhas podem ser atualizadas para sincronizar com o SaMBa, principalmente no caso do CVS se este estiver preparado para *login* via RSH²⁶ ou SSH. Porém, se esse não for o caso, não é aconselhável manter essa opção ativa, uma vez que isso não afetará a atualização das senhas para os *shares*, servidores e impressoras, além de poder (potencialmente) provocar falhas sérias de segurança, principalmente se o administrador liberar *shells* locais para todos os usuários. Veremos mais sobre esse assunto na Seção 6.6.3.6, na Página 185.

`pam password change` é uma opção muito útil no caso de ambientes GNU/Linux onde esteja ativa a estrutura PAM²⁷, pois permite que o SaMBa procure usar a estrutura de senhas definida pelo PAM (por exemplo, um servidor LDAP) como autenticação. Porém, essa opção é anulada no caso da opção `encrypt passwords` estar ativa, pois o PAM não é (ainda) capaz de obedecer ao mecanismo de desafio e resposta exigido na autenticação de servidores Windows. Na maioria dos casos, deixar em `yes` essa opção não irá fazer a menor diferença, uma vez que ele irá ignorar os controles de acesso do PAM, portanto mudar essa opção é irrelevante na versão atual do SaMBa.

Atenção: para a versão 4.0 do SaMBa existe a possibilidade de compatibilidade do SaMBa com o PAM, principalmente para permitir o uso do OpenLDAP, um servidor aberto de LDAP, em

²⁴ **Concurrent Version System** — Sistema de Versões Concorrentes: um sistema de SCM (*Source Code Management* — Gerenciamento de Código Fonte) muito popular em projetos *open source*

²⁵ outro sistema SCM, este mais novo e com mais recursos em relação ao CVS

²⁶ *Remote Shell* — Shell Remoto

²⁷ **Pluggable Authentication Modules** — Módulos Anexáveis de Autenticação: PAM é uma biblioteca para o Unix que permite ao administrador definir como o sistema irá trabalhar com a autenticação de usuário. O efeito real disso é permitir que diversos sistemas que exijam autenticação é que sejam compatíveis ou com o PAM ou com ferramentas de autenticação suportadas pelo PAM trabalhem com uma senha única. Um exemplo seria uma estrutura baseada em LDAP (**Lightweight Directory Authentication Protocol** — Protocolo de Autenticação em Diretório Level) que, por meio do PAM, permitisse uma senha única para acesso SSH, Telnet, HTTP via Apache e *login* local

combinação com o PAM para oferecer uma estrutura de acesso compatível com a tecnologia *Active Directory*TM. Porém, o autor não pode checar a validade dessa informação, uma vez que, na época em que esse documento foi escrito, o Samba 4.0 ainda encontrava-se em versão *beta*.

A opção seguinte é muito importante, principalmente por *não dever ser manipulada*. A opção no caso é `passwd chat`. Ela permite que o Samba consiga configurar corretamente o que passar e como passar informações no caso da troca de senhas. Em geral, nos arquivos `smb.conf` fornecidos com as distros são pré-configurados para os utilitários `passwd`. Portanto, a dica é: *não altere essa opção*, exceto se você *realmente souber o que está sendo feito*.

A opção seguinte, `username map` permite um truque bastante interessante: imagine que você tem uma série de usuários, por exemplo, um grupo de contabilidade. Ao invés de se preocupar com desenvolver um compartilhamento para cada um, você pode recorrer a um “truque”, que é colocar uma linha no arquivo indicado nessa opção, mapeando aqueles usuários para um outro usuário/grupo. Por exemplo, imagine que você tenha um grupo UNIX `contabilidade` e você deseje que eles possam acessar algum compartilhamento cuja pasta local (no servidor) é de posse do usuário `contabil`. Você então precisaria apenas adicionar uma linha com a seguinte no arquivo:

```
contabil = @contabilidade
```

Ou então você poderia fazer diretamente isso, usuário a usuário, como no seguinte exemplo:

```
contabil = joao, maria, kim, john, hans, shurato
```

Esse truque também permite traduzir nomes Windows com espaços. Por exemplo, imagine o usuário Windows “Diretor Corporativo”. Permitir seu acesso sem o uso de `username map` é impossível, uma vez que o Unix (e por consequência, o GNU/Linux) não aceita espaços nos nomes de usuário. Você poderia criar um usuário `ceo` (ou qualquer nome que você ache válido) e colocar uma linha como a seguinte no arquivo de mapeamento:


```
ceo = "Diretor Corporativo"
```

O último truque interessante com o `username map` é criar um mapeamento genérico para diversos usuários. Isso pode garantir um bom nível de segurança, quando combinado com outras soluções, “cercando” usuários visitantes à sua rede. Para isso, crie uma conta qualquer (`guest` costuma ser um bom nome nesse caso) e coloque um asterisco (*) (um atalho para *todos os usuários*) após uma linha com `guest` no arquivo de mapeamento, como no exemplo abaixo:

```
guest = *
```

Existe, porém, uma pegadinha: o SaMBa segue o mapeamento até o fim do arquivo, linha a linha, substituindo os nomes encontrados pelo nome a se mapeado. Imagine que você então crie um arquivo com os exemplos adicionados anteriormente:

```
contabil = @contabilidade
contabil = joao, maria, kim, john, hans, shurato
ceo      = "Diretor Corporativo"
guest    = *
```

O problema aqui é que o SaMBa não sabe onde parar. Vamos imaginar que nosso diretor corporativo logue-se em um servidor configurado com esse arquivo. O problema é que, depois de traduzir “Diretor Corporativo” como o usuário `ceo`, ele irá continuar a “traduzir” nomes de usuário, até cair no `*` que irá traduzir *qualquer* usuário em `guest`. Portanto, nosso “Diretor Corporativo” será “enxergado” pelo SaMBa como um usuário qualquer.

Como dica, o manual do `smb.conf` (5) sugere que se coloque um ponto de exclamação no início da linha. Podemos entender esse ponto de exclamação como um “*se achar um usuário Windows se logando com um desses nomes, ‘traduza-o’ para o nome de usuário indicado e pare*”. Isso irá garantir o efeito desejado no caso. Para corrigir nosso arquivo, de modo que as “traduções” sejam feitas corretamente, escrevemos um arquivo como o abaixo:

```
!contabil = @contabilidade
!contabil = joao, maria, kim, john, hans, shurato
!ceo      = "Diretor Corporativo"
guest     = *
```

Importante notar que esse sistema possui ainda outros problemas, principalmente quando a opção `security` está definida para `share` ou `user`: nesses graus de segurança, a “tradução” do nome de usuário é feita *antes* da credencial do usuário. Ou seja: algumas vezes pode ser necessário uma senha diferente para entrar em um *share* com esse tipo de mapeamento.

O padrão do SaMBa é não trabalhar com arquivos de mapeamento, uma vez que ele prefere confiar explicitamente nas permissões de usuário do Unix. Esse arquivo pode ser muito útil se corretamente configurado, porém, e pode ser uma ótima idéia para o administrador manter um arquivo desse para *shares* de grupos de usuários (por exemplo, manter um *share* individual do usuário e um outro liberado para todos os usuários da divisão em questão).

As duas opções seguintes, `socks options` e `dns proxy` lidam respectivamente com ajustes de *performance* e com a habilitação do SaMBa como um *resolver* de WINS²⁸. Essas são opções avançadas e podem ser deixada em seus *defaults*, ao menos em um primeiro momento.

As opções seguintes, `logon drive`, `logon home`, `logon path` e `logon script`, são usadas pelo SaMBa para a administração do *logon* do usuário e são úteis apenas quando o usuário vai usar o SaMBa como PDC ou servidor de *logon* no grupo de usuários. A primeira delas, `logon drive`, indica qual o drive a ser mapeado pela estação Windows (98 ou 95) como o diretório *home* do usuário. Uma coisa importante aqui é que o *home* do usuário, exceto se houver alteração na seção `[homes]` do arquivo `smb.conf` (veja a Seção 6.6.3.3, na Página 184), será o mesmo indicado na conta Unix do usuário (em geral, `/home/<nome_do_usuario>`). O *default* é que o Windows passe a mapear o *home* do usuário para `H:`.

Em seguida, `logon home`, informa realmente o diretório do usuário no Windows onde estarão suas informações pessoais. No Windows 98,

²⁸ **Windows Internet Name Service** — Serviço de Nomes de Internet **Windows**: um “protocolo paralelo” ao DNS desenvolvido pela Microsoft como “substituto” ao DNS para suas redes Windows.

por exemplo, encontram-se dentro dela `Documents and Setting`, `Meus Documentos` e `Favoritos`, usados por diversas aplicações Windows para armazenar configurações e arquivos importantes. No caso, colocamos a opção `\\%L%\%U\.profiles`, que irá mapear o diretório do usuário no Windows para dentro do compartilhamento do usuário dentro do servidor. A “pegadinha” aqui é que essa pasta equivale ao *home* Unix do usuário. A importância de criar o nome do diretório de *profile* com o ponto está no fato de que o diretório passa a se tornar “desinteressante”, como visto na Seção 3.1.3, 67. Em geral, isso provê alguma segurança quanto a uma exclusão acidental se você permitir que os usuários do SaMBa acessem local ou remotamente um *shell*. Se não for o caso, você pode deixar sem o ponto mesmo.

A seguir vem a opção `logon path`, que cumpre a mesma função de `logon home`, mas para usuários de versões mais avançadas do Windows, como o NT e o XP. Importante ressaltar para ambos os casos:

1. O diretório Unix onde as informações dos perfis de viagem serão salvos deve ter permissão de leitura para o usuário no qual o usuário Windows será mapeado (normalmente o mesmo nome de usuário dos demais casos, exceto se um `username map` estiver definido), e também de escrita (ao mesmo no primeiro *logon*). Se quiser desabilitar os perfis de viagem, defina tanto `logon home` quanto `logon path` como “”, desabilitando o mesmo;
2. Não use aspas nos nomes de diretório definidos, pois isso irá romper a capacidade do SaMBa de *parsing* de diretórios, provocando erros de difícil detecção no ambiente SaMBa;

A última opção dessa seção, `logon script`, define um *script* que será executado na estação local assim que o *logon* for executado. Esse *script* não é um *shell script* Unix, e sim *script* `.bat` ou `.cmd` convencional, que deve respeitar codificações de arquivo e formato de arquivo DOS. Se você for criar um *script* “do zero” em uma estação GNU/Linux, você pode usar os editores de texto `vi` e `EMACS`, descritos no Capítulo 7, Página 187, tomando o cuidado de configurar corretamente as codificações e formatos de arquivos (para o `vi`, verificar a Seção 7.2.5, Página 196; para o `EMACS`, verificar a Seção 7.3.7, Página 209). Esse *script* é muito útil para instalações de atualizações ou montagem automatizada de *shares* SaMBa na estação, o que pode ser muito valioso em termos de segurança e satisfação do usuário.

Com isso, terminamos a seção `[global]` do nosso arquivo exemplo. Ainda existe muito a ser dito sobre isso, mas aconselho que você pesquise na Internet por conta própria ou consulte a *manpage* do `smb.conf(5)`. Vamos continuar a explicação, seguindo para a seção `[homes]`.

6.6.3.3 A seção `[homes]`

Essa seção é muito importante, pois ela trata de como os compartilhamentos *home* dos usuários irão trabalhar. Perceba que aqui já começamos a lidar com compartilhamentos de arquivo, então muito que é dito sobre compartilhamentos na Seção 6.6.3.5, Página 185 é válido aqui e vice-versa. Portanto, vamos concentrar-nos no que for possível de resolver apenas nesse caso. No caso, vamos tratar apenas de um parâmetro que não aparece no nosso exemplo do Trecho de Código 6.6.3, Página 174, que é `path`.

Como dissemos ao explicar as opções `logon home` e `logon path`, o SaMBa, por padrão aceita o *home* Unix do usuário como *home* do SaMBa. Essa opção permite que você configure um novo diretório para que os dados do usuário ou do compartilhamento sejam salvos em uma outra pasta qualquer.

6.6.3.4 A seção `[printers]` e configuração para impressão

Embora nosso arquivo exemplo não mostre compartilhamento de impressoras usando SaMBa, vamos falar rapidamente sobre a configuração do SaMBa como servidor de compartilhamento de impressora.

A primeira coisa é adicionar três opções na seção `[global]`: `printing`, `printcap name`, e `load printers`. Essas opções, respectivamente, indicam qual o servidor de impressão local (normalmente `lprng` ou `cups`, no GNU/Linux), qual arquivo contém as informações de configuração das impressoras disponibilizadas (conhecida como *printcap*, e normalmente definido em `/etc/printcap`) e se todas as impressoras disponibilizadas pelo *printcap* serão mapeadas pelo sistema (o padrão é `yes`).

Em seguida, temos que ter uma seção `[printers]` com as configurações gerais de impressora. Nelas, temos que usar a opção `path`, indicando o diretório de *spools*.

6.6.3.5 Configurando as seções dos compartilhamentos

6.6.3.6 Habilitando um usuário/máquina no SaMBa

6.6.3.7 Acessando um *share* SaMBa

6.7 Introdução ao Firewall

Capítulo 7

Editores de texto

Uma das principais ferramentas, não apenas para a administração, mas para o uso em geral, dentro do GNU/Linux é o editor de texto. No caso, entende-se editor de texto ao programa que permita a digitação, gravação e correção de textos em formato ASCII puro (diferentemente dos processadores de texto como o MS-Office ou o OpenOffice.org). Com o uso de um editor de texto e as ferramentas corretas, você pode ir desde criar páginas Web e *emails* até a editar novos programas e criar textos de alta qualidade¹.

Esse capítulo será dedicado a uma introdução aos dois principais editores de texto para Unix e GNU/Linux: o **vi** (e clones como **vim** e **elvis**) e o GNU **emacs** (e seus principais clones, o **jed**, e o **XEmacs**). Na prática, existem muitos outros editores, como **pico**, **nano**, **joe** e até ferramentas visuais, como **kate** e outras. Mas esses dois são facilmente encontrados em qualquer distro GNU/Linux (principalmente o **vi**)².

¹No caso, com ferramentas como DocBook e L^AT_EX

²Antes que perguntem: o autor prefere (e muito) o **emacs**, mas é capaz sim de operar, com alguma dificuldade, o **vi**. Mas essa é a grande vantagem do software livre: se não gosta de um, use outro e seja feliz! *Live and Let Live!!* :P

7.1 Por que é importante manipular um editor de texto

Nos Capítulos anteriores, comentamos sobre algumas noções básicas da administração do GNU/Linux. Em geral, essa administração passa pela edição de arquivos de configuração. Isso é uma norma em qualquer ambiente. Em alguns, como no WindowsTM e na maioria das distros GNU/Linux, existem ferramentas gráficas de configuração poderosas.

Mas o GNU/Linux foi “projetado” segundo o padrão original do Unix, o que quer dizer que você cedo ou tarde *irá precisar editar manualmente* um arquivo de configuração, o que irá querer dizer que você precisará cedo ou tarde lidar com um editor de texto.

Com a evolução do GNU/Linux, clones dos principais editores de texto puro, como os encontrados em IDEs e editores antigos como o WordStarTM foram surgindo para o GNU/Linux. Mas, como derivado do mundo Unix, o GNU/Linux tem dois que são os maiores contendores nessa arena: o **vi** (através dos seus clones **vim** e **elvis**) e o GNU **emacs** (junto com seus clones **jed** e **XEmacs**). Então será nesses dois que iremos nos focar, pelo fato de ser muito fácil encontrar *pelo menos* um deles em qualquer instalação GNU/Linux.

7.1.1 Qual deles é o melhor?

Dizer qual dos dois é o melhor entre **vi** e **emacs** é o mesmo que perguntar qual time de futebol é o melhor: é mais uma questão de afinidade.

Os defensores do **vi** gostam de dizer que o **vi** segue o KISS (*Keep It Simple and Safe* — Mantenha simples e garantido)³, o princípio do Unix que dita que um software deve fazer uma coisa só, mas fazer ela *bem*. Portanto o **vi** seria um bom editor de texto por fazer apenas isso. Além disso, por ser *apenas* um editor de texto, o **vi** é enxuto e rápido e pode ser instalado em qualquer lugar sem maiores problemas. Uma grande vantagem do **vi** é o fato de que ele é parte do *Single Unix Specification* (THE OPEN GROUP,

³Na verdade, é *Keep It Simple, Stupid* — Mantenha isso simples, imbecil — mas preferi ser mais positivo e politicamente correto. :P

2006), o que indica que qualquer sistema baseado em Unix que queira usar o termo de alguma forma *tem* que incluir o **vi**, ou um clone. Isso também inclui o uso do **vi** em testes para o LPIC⁴ nível 1.

Os defensores do **emacs** porém, gostam de dizer que o mesmo é muito versátil, pois o mesmo pode atuar como editor de texto com características avançadas, como *syntax highlight*⁵, recursos de formatação simples, como centralização e justificação, modos específicos conforme o tipo de documento e até mesmo módulos extras com novos recursos (o **emacs** traz até um terapeuta freudiano com IA). Além disso, por ter sido construído em LISP⁶ ela permite poderosa personalização e configuração do sistema. A vantagem do **emacs** é o fato de que ele é incluído dentro do pacote de utilitários GNU, que atualmente é portado em quase todos os ambientes Unix.

Qual deles é o melhor? A única forma de descobrir é usando os dois⁷, que é o objetivo do capítulo.

Um último conselho: *jamais* entre em uma discussão de qual dos dois editores é melhor. Esse é considerado assunto religioso e motivo até de *flame wars*⁸. *Você foi avisado!!!!*

7.2 Introdução ao vi

Agora veremos uma introdução ao **vi** que lhe permitirá usar o mesmo rapidamente. Se quiser mais informações, use o comando **vimtutor** na linha de comando ou então veja o **vilearn**, de Klinger e Craig (1992). Outra opção boa é o Guia de Consulta Rápida “Editor **vi**”, da Editora Novatec, do Roberto Coelho(COELHO, 2002).

O **vi** (*visual editor*) foi escrito originalmente por Evans Hall e entre

⁴*Linux Professional Institute Certificate* – Certificado do Instituto de Profissionais Linux

⁵*reforço de sintaxe*, uma característica muito útil ao editar arquivos de configuração e códigos-fonte, aonde as palavras são ressaltadas conforme o contexto: comandos, variáveis, ...

⁶**LIS**t **P**rocessor, Processador de Listas, uma poderosa e complexa linguagem de programação

⁷o autor aconselha o **emacs**, mas quem tem que definir isso não é ele, e sim o leitor. :P

⁸*guerra de chamadas*, a versão Internet das brigas de torcida


outros mantenedores teve Bill Joy, que viria a ser um importante desenvolvedor no 4.0BSD e fundador da Sun Microsystems⁹, na época um graduando de Ciências da Computação na Universidade da Califórnia, campus Berkeley (UCB). Ele foi criado para o 3.3BSD em substituição a um outro editor de texto, o `ed`, e tinha um licenciamento que não era livre, mas era aberto. Com o tempo, foram criados vários clones do editor, sendo o mais importante e mais usado o `vim`(*vi Improved* — `vi` melhorado), desenvolvido por Bram Moolenaar e outros. Ele foi licenciado pela GPL, mas o autor também pede que aqueles que queiram fazer um bem para alguém, como ele fez, que doe dinheiro para instituições de caridade na Uganda, um país castigado pela AIDS e pela pobreza¹⁰.

O `vi` é um editor *modal*, no qual as teclas do sistema assumem características e funções diferenciadas conforme o modo em que o editor se encontra. Isso é muito importante, pois certos comandos só podem ser dados no `vi` em um ou outro modo.

O comando para entrar-se no `vi` é `vi <nome_arq>`, aonde `nome_arq` representa o arquivo que deseja-se editar. Se você não chamar nenhum arquivo, você receberá uma tela similar à do Trecho de Código 7.2.1, Página 191.

Perceba que no caso irá aparecer a janela do `vim`. Não se preocupe, pois todos os comandos que veremos funcionam em qualquer versão ou clone do `vi`.

Perceba que o `vi` mostra uma série de linhas com `~` no seu começo. Essas linhas na verdade não existem. Elas são apenas para ocupar o espaço da tela.

Atenção: Os comandos do `vi` são de dois tipos: *teclas de atalho* e *comandos*. Os comandos iniciam com `:`, enquanto as *teclas de atalho* são mostradas nesse tutorial com um desenho de tecla (). Chamaremos ambos de “comando” apenas para facilitar a compreensão do funcionamento do `vi`.

⁹uma das maiores empresas de informática de todos os tempos e a primeira grande empresa a trabalhar com Unix para o mercado de massa

¹⁰o que não deixa de ser um bom ato


```

\subsection{Edição e navegação por arquivos}

Uma coisa importante antes de começar a trabalhar com o \texttt{vi} é entender o
s \emph{modos de uso} do mesmo. O \texttt{vi} trabalha em dois modos básicos: \e
mph{comando} e \emph{edição}. O \texttt{vi} entra em modo de comando. Nesse modo
você \emph{não pode} entrar texto no \emph{buffer} onde você está (o \emph{buff
er} é uma cópia do arquivo que você deseja editar que fica na memória, de modo a
preservar o arquivo). Para editar o arquivo, você deve passar para o \emph{modo
de edição}. Para isso, pressione a tecla \keystroke{i} ou a tecla \Insert. Para
sair do modo de edição e voltar ao modo de comando, tecle \Esc. O modo de ediçã
o pode ser identificado pelo símbolo \verb|-- INSERT --|

\begin{codigo}[htp]
\scriptsize
\begin{Verbatim}[frame=single]

                                end{Verbatim}
\caption{Janela do \texttt{vim} em modo de edição}
\label{code:viminit}
\end{codigo}


\section{Introdução ao \texttt{emacs}}





-- INSERT --

```

159,1
















Bot




















Trecho de Código 7.2.2: Janela do vim em modo de edição

de comando, você utiliza as teclas  ,  ,  e  , respectivamente.

Atenção: Em algumas compilações do vi para PC, os direcionais podem também ser usados no modo de comando. Porém, esse não é a forma normal de navegar-se pelo documento vi em modo de comando.

Em modo de comando, pode-se usar alguns outros comandos interessantes para navegar-se no texto:

-  : vai para o começo da próxima palavra;
-  : vai para o fim da próxima palavra;
-  : vai para o fim da palavra anterior;
-  e  : vai para o fim e o começo da linha;
-  : vai para a coluna 0 da linha;
-  : Vai para a coluna de número digitado (por exemplo, digitando-se  , vai para a coluna 23 da linha);
-  : procura a próxima ocorrência de um determinado caracter na linha (que deve ser digitado em seguida);
-  : procura a ocorrência anterior de um determinado caracter na linha (que deve ser digitado em seguida);
-  : move-se até a próxima ocorrência de um determinado caracter (que deve ser digitado em seguida);
-  : move-se até a ocorrência anterior de um determinado caracter (que deve ser digitado em seguida);
-  e  : repete a última dessas quatro operações e volta para o último resultado de um deles, respectivamente;
-  : Marca uma determinada posição na tela, nomeando-o com um determinado caracter (que deve ser informado em seguida);

-  : Move-se para uma determinada posição na tela nomeada pelo caracter informado em seguida (usar um outro  devolve o usuário para o local anterior;
-  : Vai para a última linha do documento. Se um número for informado antes do , salta para aquela linha em questão;
-  e  : Vai para o início e para o fim de um parágrafo¹¹;
-  e  : Vai para o início e para o fim de uma sentença;
-  : Procura por uma determinada *string* ou trecho de *string* para baixo¹²;
-  : Idem a , mas para cima no texto;
-  e  : Avança ou volta na última pesquisa  ou  realizada;
-  +  e  +  : permitem que o sistema role a tela uma janela para cima ou para baixo;

7.2.2 Abrindo e salvando arquivos

Normalmente você já vai abrir o arquivo a partir do comando na linha de comando. Porém, algumas vezes pode ser que você queira carregar o arquivo dentro de uma seção do `vi`. Para isso, utilize o comando `:e` e o nome do arquivo. Por exemplo, se você quiser abrir o arquivo `/etc/fstab`, use `:e /etc/fstab`.

Se você tiver editando um arquivo, precisará salvar as alterações antes de carregar um novo arquivo. Para isso, use o comando `:w`. Se quiser salvar o *buffer* do arquivo com outro nome, digite o nome do arquivo logo após de `:w`. Por exemplo, se você quiser gravar o arquivo `potionessay` que você abriu do usuário `~hgranger` no seu diretório *home*, utilize o comando `:w ~/potionessay`.

¹¹a maioria dos editores Unix, por causa do sistema de processamento de textos `TEX` adota a convenção de um parágrafo sendo determinado por texto separado por uma linha em branco entre si

¹²na verdade, realiza uma pesquisa por expressões regulares, mas isso não é tão relevante nesse momento

É possível dividir a janela do **vi** em duas para editar dois arquivos. Para isso, no modo de comando, digite `:spl` (de *split* — dividir). A janela será dividida em duas no mesmo *buffer*. Você pode abrir outro arquivo em uma das janelas, colocando outro *buffer* nela, com o comando normal `:e`. Para alternar entre as janelas, utilize, no modo de comando, `[Ctrl] + [W]`, e em seguida `[↑]` e `[↓]` para deslocar-se entre as divisões. Esse comando também permite alternar entre *buffers* do **vi**.

Para sair de uma das divisões, use o comando `:q`. Se quiser gravar antes as alterações, use o comando `:wq`. Perceba que aqui estou combinando os comandos `:w` e `:q`. Se quiser ignorar as alterações do *buffer*, use `:q!`. O ponto de exclamação(!) indica que você está ignorando alertas do sistema que te impediriam de fechar o *buffer*. Quando houver apenas uma janela (ou *buffer*), o uso desse comando encerra o **vi**.

7.2.3 Cortar, Copiar, Colar e Apagar

Realizar operações de recortar, copiar, colar e apagar trechos de texto é uma das funções mais comuns da edição de texto, e o **vi** oferece alguns comandos.

Para deletar um trecho de texto, você utiliza o comando `d`, seguido de um comando de movimentação de texto. Por exemplo, se você quiser apagar todo o texto até o final do texto, utilize o comando `d$`. Se quiser apagar todo o texto até a próxima ocorrência de um caractere `&`, utilize `df&`. A única situação especial é apagar uma linha completa. Para esse caso, utilize `dd`. Perceba que esses comandos funcionam no modo de comando. No modo de edição, você pode recorrer a `[Del]` e `[←]`.

Essa deleção funciona como o comando “*Cortar*” de qualquer editor de texto: ele remove o texto de dentro do *buffer* e o envia para uma área especial de memória do sistema, a área de transferência (ou *clipboard*). No caso, como esse texto está no *clipboard*, o **vi** permite que você o cole em qualquer posição do texto. Para isso, utilize `[p]` ou `[P]`, para colar o texto na área de transferência antes ou depois do cursor, respectivamente.







No caso, se você quiser, você pode recortar mais textos, colocando um tanto de números após o comando `d` e antes do comando de movimentação de texto; Por exemplo, usando `d3{`, removemos os três próximos parágrafos.





Para copiar, ou *yank*, você faz o mesmo tipo de comando da deleção, mas usando *y* ao invés de *d*. Para copiar uma linha, use *yy*, ao invés de *dd*.




Tanto para deleção quanto para a cópia, você pode usar marcas. Por exemplo, se você marcou um trecho do texto com a letra *k*, use *y'k* para criar uma cópia de todo o texto entre o cursor e o ponto *k*.

7.2.4 Truques de edição

O *vi* permite que, ao passar para o modo de edição, você pode usar truques que lhe permitam escolher o modo em que se entra:

-  ou  : abre uma linha antes ou depois do cursor;
-  : insere texto antes do cursor;
-  : insere texto no início da linha;
-  : adiciona texto após o cursor;
-  : adiciona texto no final da linha.;

O comando  no modo de comando permite emendar linhas uma nas outras. Você pode usar também  para substituir (deletar e inserir) os # caracteres do texto com os do *clipboard*, e usando  para substituir linhas inteiras. E você pode usar  para fazer a conversão maiúscula/minúscula do caracter atual.

Você pode desfazer qualquer coisa que você fez na sua linha, *exceto deleção total da linha* com *dd*, *enquanto não mover-se para outra linha* usando  . Para desfazer as alterações uma a uma você pode usar o  , e para repetir o último comando em outra posição é usando  .

7.2.5 Mudando a codificação do arquivo

Algumas vezes, você terá que mudar a codificação do arquivo para editar arquivos específicos no seu GNU/Linux. Por exemplo, você poderá precisar editar um *script* de *logon* na rede SaMBa ou um arquivo em Unicode no seu

vi. Se o arquivo em questão for aberto, não há problemas: o vi detecta automaticamente a codificação (ou *encoding*) do arquivo. Mas se for um arquivo novo, será um pouco mais complicado que isso.

Para modificar a codificação do arquivo, primeiro passe para o modo de comando para alterar a do *buffer*. No vi, precisa-se fazer a configuração da codificação do arquivo propriamente dita. Em geral, no Brasil usará-se as codificações `latin1` e `utf-8`. Utilize o comando `:set encoding` (ou `:set enc`) para definir a “página de código” (codificação do arquivo) do *buffer* em questão. Use `latin1`, pois essa opção irá definir a codificação com a qual o *buffer* será exibido.

Em seguida, use `:set fileencoding` (`:set fenc`), que definirá em que codificação o arquivo do *buffer* em questão será salvo. Para o Brasil, os padrões normais serão, além dos já citados `latin1` e `utf-8`, o `iso-8559-1` (apenas uma outra forma de definir `latin1`). Em todos os casos, não se preocupe com a tradução de codificações: o vi se responsabilizará quanto a isso por você.

Ainda falta uma opção, que é `:set fileformat` (`:set ff`, onde voc definirá o *formato de arquivo* quanto a terminações de linha e afins. Esse valor poderá ser `dos` (DOS/Windows), `unix` (Unix e derivados) ou `mac` (Macintosh). Isso deve ser feito se precisar que o final de linha e outras especificações pessoais de cada sistema operacional sejam respeitadas pelo vi.

7.2.6 O Arquivo `/.exrc` ou `/.vimrc`

Para finalizar, existe um arquivo de configuração para o vi, o `/.exrc` (que pode aparecer `/.vimrc`) também como que permite o ajuste do comportamento do vi. Como esse não é o assunto dessa introdução, não entraremos nos detalhes desse arquivo. Basicamente, o estado atual de configuração do seu vi é oferecido pelo comando `:set`. Para listar todas as opções de configuração disponíveis em seu ambiente, use `:set all`.

Usando esse comando você também pode acessar e modificar as opções do vi. Por exemplo, se você quiser configurar um tamanho máximo de linha de 72 caracteres, configure a variável ambiental do vi `wm` (*Wrap Margin* — Margem de quebra de linha), ou seja, não permitir que alguma linha tenha

mais que 72 caracteres, use `:set wm=8`¹³. Para colocar isso no arquivo de configurações, abra o mesmo e digite `set wm=10`. Perceba que *não foram* colocados os pontos. O mesmo vale para qualquer outra opção.

Você também pode utilizar esse arquivo para configurar abreviaturas que serão substituídas caso sejam digitadas. Por exemplo, se você ao digitar `HP` quer que apareça `Harry Potter`, utilize em modo de comando `:ab HP Harry Potter`. Nesse caso, as teclas deverão ser acionadas no modo de edição para que isso faça efeito. No arquivo `.exrc`, utilize `ab HP Harry Potter` para que a abreviatura funcione.

Você pode também criar mapeamentos, que fazem a mesma função das abreviaturas, mas no modo de comando. Utilize para isso `:map <character> <texto>`, lembrando que ele pode ser gravado dentro de `.exrc`, usando `map <character> <texto>`.

Essa introdução deve ser o suficiente para que você consiga editar um arquivo `.exrc` que lhe seja adequado. Você pode encontrar farta informação sobre esse assunto no *vim User Manual* de Bram Moolenaar (MOOLENAAR, 2003).

7.3 Introdução ao emacs

7.3.1 A História do emacs

O emacs (*Editing macros*) foi criado por Richard Stallman (futuro fundador do Projeto GNU e da *Free Software Foundation*) e Guy Steele, como um editor de macros para o editor TECO (*Tape Editor and Corrector* — Editor e Corretor de Fitas, ou também *Text Editor and Corrector* — Editor e Corretor de Texto), que rodava em uma plataforma de sistema operacional chamada ITS (*Incompatible TimeSharing* — Compartilhamento de Tempo Incompatível), produzidos ambos no MIT. Depois de algum tempo, o emacs acabou se tornando independente do teco, de forma similar ao caso vi e ex. E, também como o vi, o emacs também gerou clones, ou melhor dizendo,

¹³no caso, o vi trata a margem como um determinado número de caracteres à esquerda que não serão preenchidos com texto

*forks*¹⁴:

- *Gosling EMACS*: Criado por James Gosling (que viria a ser um dos arquitetos da tecnologia JavaTM), foi o primeiro clone do **emacs** a rodar em plataforma Unix. Inicialmente era de código aberto (embora não livre) e construído em LISP, assim como a versão original. Em 1984, porém, foi comprado pela UniPress, que fechou seu código. Com o código que pode aproveitar do **gosmacs** (apelido do *Gosling EMACS* e claro partes do **emacs** para o TECO, ele veio a criar o GNU **emacs**. Utilizava uma variante do LISP chamada *Mocklisp*;
- *GNU EMACS*: Foi o primeiro de uma legião de softwares produzidos pelo Projeto GNU. Criado por Richard Stallman, utilizou como base o **gosmacs**, mas tendo substituído todo o código proprietário de maneira bem rápida. Substituiu o *mocklisp* por uma variante completa do LISP, o EMACS LISP. Era similar ao **gosmacs** e também rodava no Unix, mas com o tempo foi ganhando muitas melhorias, desde coisas incrivelmente poderosas que os modos relacionados a programação com *syntax highlight* e auto-identação, até recursos realmente bizarros para um editor de texto (mas com certeza úteis) como leitor de email, cliente de IRC e até mesmo alguns *plugins* baseados em inteligência artificial, como jogos e um terapeuta freudiano (!!!)¹⁵. Acabou por se tornar o padrão *de facto* no Unix quando o assunto é EMACS;
- *XEmacs*: Anteriormente conhecido como *Lucid Emacs*, foi um *fork* do código de uma versão alfa do GNU EMACS versão 19, cujo código rapidamente divergiu um do outro. Embora utilizar o EMACS LISP, possui algumas diferenças no uso, o que impede (ao menos parcialmente) que pacotes (como são chamados os *plugins* do EMACS) funcionem no XEmacs e divide com o GNU EMACS a ponta no uso ambiente Unix.

¹⁴O termo vem do comando `fork()` do C — utilizado para gerar novos processos — e denomina a ação que pode ocorrer de, por questões técnicas, filosóficas ou de encaminhamento de um projeto, alguns dos autores do software abandonam o projeto original e criam uma versão do mesmo com outro encaminhamento.

¹⁵baseado na série de sistemas de IA ELIZA, coincidentemente desenvolvida em LISP

Existem outros clones do EMACS, como `jed`, `jove`, `freemacs`, e afins, mas vamos nos manter no GNU EMACS¹⁶, pois a maior parte dos comandos são iguais para todas as versões e clones do EMACS.

Os fãs do EMACS o admiriam pela sua versatilidade e capacidades de customização, em grande parte oferecidas pelo EMACS LISP (que é um LISP completo). Como o GNU EMACS é incluído como parte do pacote de software GNU, os seus defensores afirmam que, se um sistema possui os pacotes GNU (o que quase todos possuem atualmente no mundo UNIX), ele terá o EMACS.

Seus detratores, porém, acusam o `emacs` de ser *bloatware*¹⁷, usando termos derogativos como “*Eight Megabytes And Constantly Swapping*” (Oito Megabytes e uma porrada de Memória Virtual), “*Eventually malloc()s All Computer Storage*” (Ocasionalmente aloca todo o disco do sistema¹⁸) e “*EMACS Makes A Computer Slow*” (O EMACS deixa um computador lento)(RAYMOND, 2003). Em parte isso é verdade, uma vez que a distribuição do EMACS costuma ocupar algo em torno de 50 MB de disco em suas versões mais atuais. O principal motivo, porém, é a grande quantidade de modos específicos de linguagem suportados (C/C++, L^AT_EX, T_EX original, Texinfo, `diff`¹⁹, CVS²⁰) e pelas necessidades do EMACS LISP. De qualquer forma, seus detratores afirmam que não é possível incluir um clone do EMACS em um *rescue disk*²¹, por exemplo. Isso pode ser desmentido por clones do EMACS como `jed` ou `jove`.

Para entrar-se no EMACS, digite o comando `emacs <nome_arq>` na linha de comando (você não precisa obrigatoriamente entrar com `nome_arq`, caso em que você receberá uma janela como a mostrada no Trecho de Código 7.3.1, na Página 201).

¹⁶**NA:** Já vou avisando que não darei muitas dicas quanto a ao `emacs`, embora seja fã assumido, para não ser injusto com o `vi`

¹⁷Software que possui código demais em relação à quantidade de *features* — características úteis — do sistema

¹⁸aqui há um pequeno erro: `malloc()` representa alocação dinâmica de memória RAM, não de disco. Mas não dá para afirmar que não foi uma boa tentativa

¹⁹utilitário Unix usado para mostrar as diferenças entre duas versões de um determinado código ou arquivo

²⁰*Concurrenty Version System* — Sistema de Versões Concorrentes, um sistema de SCM — *Source Code Management*

²¹disco de recuperação

```

File Edit Options Buffers Tools Help
Welcome to GNU Emacs, one component of a Linux-based GNU system.

Get help          C-h (Hold down CTRL and press h)
Undo changes      C-x u          Exit Emacs          C-x C-c
Get a tutorial     C-h t          Use Info to read docs C-h i
Ordering manuals  C-h RET
Activate menubar   F10 or ESC ' or M-'
('C-' means use the CTRL key. 'M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)

GNU Emacs 21.3.2 (i386-redhat-linux-gnu)
of 2004-10-18 on tweety.build.redhat.com
Copyright (C) 2001 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
of Emacs and modify it; type C-h C-c to see the conditions.
Type C-h C-d for information on getting the latest version.

--uuu:---F1 *scratch*          (Lisp Interaction)--L1--All-----
For information about the GNU Project and its goals, type C-h C-p.
```

Trecho de Código 7.3.1: Janela inicial do emacs

Essa janela mostra as quatro divisões do EMACS: menu, *buffer*, *modeline* e *minibuffer*. O *buffer* contém uma *cópia* do arquivo carregada na memória (essa diferença é importante), além de alguns *buffers* especiais serem possíveis. Em geral, eles são marcados com * antes e depois do nome do *buffer*. Alguns deles são:




- **scratch**: é um local que você pode usar para editar arquivos novos. Depois veremos como salvar tais arquivos. Além disso, **scratch** pode ser usado como prática para aprendizado de EMACS LISP, como se fosse um terminal de comandos EMACS LISP. Não é nossa função nesse tutorial nos aprofundarmos no EMACS LISP, uma linguagem de programação/macro incrivelmente poderosa e complexa. Para maiores informações sobre o EMACS LISP, consulte “*An Introduction to Programming in Emacs Lisp, Second Edition*” (CHASSELL, 2006) e “*GNU Emacs Lisp Reference Manual*” (FREE SOFTWARE FOUNDATION, 2002);
- **Messages**: apresenta mensagens de erro e alertas que tenham sido dis-

parados durante a operação do sistema. É de bom tom dar uma olhada nele quando houver problemas;

- **Completion:** Mostra uma lista das opções válidas para determinados comandos. É usado nos modos e comandos que permitem *completion* (preenchimento automático) de opções e parâmetros.







Depois dos *buffers*, a linha seguinte é o *modeline*. Ela é chamada assim pois indica que documento estamos trabalhando e em que modo está nosso documento. No caso, estamos trabalhando no *buffer* especial **scratch**, no modo *Lisp Interaction* (Interação LISP). Esse modo permite que se use comandos LISP diretamente. Essa parte pode mudar conforme o documento que se está trabalhando.











A última linha é chamada de *minibuffer*. Essa linha pode ser usada para lançar-se comandos do EMACS diretamente, sem recorrer a atalhos de teclado. Além disso, serve para lançar parâmetros de comandos e informar sobre erros e mensagens do sistema.





Da mesma forma que no caso do **vi**, nosso objetivo aqui *não é* ensinar totalmente o EMACS. O EMACS é um software muito complexo, mais complexo que o **vi**. Nesse caso, sugerimos que consulte a documentação do EMACS(FREE SOFTWARE FOUNDATION, 2006). Além disso, você pode consultar o próprio tutorial que o EMACS traz consigo (da mesma forma que o **vi** traz o **vitutor**). Para isso, digite  +  e em seguida digite a tecla .

7.3.1.1 Convenções do emacs

O **emacs**, em sua ajuda, possui algumas convenções. A primeira delas é que alguns comandos são apresentados como *várias seqüências* de caracteres. Por exemplo, para acessar o tutorial do **emacs** como demonstramos anteriormente, a ajuda mostra **C-h t** (na verdade, você pode ver esse tipo de notação no Trecho de Código 7.3.1, na Página 201).

Nesses casos, o caracter **C** representa a tecla , e a letra **M** representa a tecla . Normalmente, na maioria dos ambientes GNU/Linux, a tecla  é mapeada para a tecla . Caso isso não aconteça, outra tecla usada como  é  (isso por causa da opção de passar-se para o

minibuffer com M-x —  +  — isso acabou sendo tratado como uma espécie de semi-compatibilidade com o vi). No caso, quando o EMACS pede que pressione-se C-x C-f, primeiro digite  +  e em seguida, digite  + . Se for pedido para usar M-x, tecele  +  (ou  + ).









Quando duas sequências estão separadas por espaço, deve-se usar uma e em seguida a outra. Por exemplo, citamos a já citada C-x C-f ( +  e depois  + ).






Quando o emacs menciona arquivos, ele costuma chamar de *buffers*. O motivo disso é que é possível salvar-se o conteúdo de *buffers* em arquivos. Uma das vantagens do uso de *buffers* é que, caso haja uma queda de energia, o risco de corrupção de dados é mínimo pois os dados são manipulados na memória.

É importante que você se acostume com os nomes de regiões da tela, pois muitas vezes você terá que visualizar ou fazer referência a cada uma dessas seções.

7.3.2 Edição e navegação por arquivos

Normalmente, ao entrar no emacs você irá para um *buffer* (um arquivo que você tenha aberto ou o *buffer scratch*). Para editar um texto, basta sair digitando naturalmente com o teclado.

Para navegar no texto, você irá usar as teclas C-p ( + ), C-n ( + ), C-b ( + ) e C-f ( + ). Você também pode utilizar-se das setas direcionais para se deslocar pelo texto. Diferentemente do vi, você pode se deslocar, editar texto e realizar operações tudo ao mesmo tempo. A única situação adversa é no caso de comandos complexos que não possuam atalhos de teclado: estes devem ser utilizados no *minibuffer*, portanto fora do *buffer* normal de edição.

Para navegar através de palavras, você pode usar as teclas M-b ( + ) e M-f ( + ) para ir a palavra anterior ou seguinte. Nesse caso, como o vi, o emacs não permite o uso dos direcionais com o  como nos editores do ambiente WindowsTM, por padrão. Porém, o emacs oferece um modo de compatibilidade com o sistema WindowsTM. Veremos mais sobre

isso quando falarmos do arquivo de configuração do `emacs`, na Seção 7.3.8, Página 209.

Para ir ao começo e ao fim de uma linha, use `C-a` (`Ctrl` + `A`) e `C-e` (`Ctrl` + `E`), e para ir ao início e ao final de uma sentença, utilize `M-a` (`Meta` + `A`) e `M-e` (`Meta` + `E`). Você pode ir ao começo do arquivo usando `M-<` (`Meta` + `<`) e `M->` (`Meta` + `>`). Perceba que você precisará usar `Shift ↑` nesse caso, pois se você tentar digitar (`Meta` + `<`) sem usar `Shift ↑`, você estará na verdade digitando (`Meta` + `,`).

Um recurso muito legal do `emacs` (presente no `vi`) é a repetição do comando. Para isso, você irá usar `C-u` (`Ctrl` + `U`) e digitar no *minibuffer* o número de vezes que o comando deverá ser repetido. Por exemplo, se você quiser avançar 10 palavras dentro do seu texto, utilize `C-u` (`Ctrl` + `U`), digite em seguida o número 10 e logo depois o comando `M-f` (`Meta` + `F`).

7.3.3 Abrindo e salvando arquivos

Para carregar um arquivo dentro de um *buffer* do `emacs`, utilize `C-x C-f` (`Ctrl` + `X` e `Ctrl` + `F`). Ele vai levar você até o *minibuffer* e perguntar qual o arquivo a ser aberto. Escolha o arquivo desejado normalmente. Um recurso ótimo é que ele aceita *completion*, ou seja, ele vai completando o nome do arquivo aonde não houver pedaços diferentes e irá parar caso haja outras opções. Para chamar o *completion*, utilize a tecla `⇐⇒`. Se vários arquivos possuir um nome parecido, o *completion* irá preencher todo o trecho de nome comum a todos. Pressionar novamente `⇐⇒` irá exibir no *minibuffer* a mensagem

Complete, but not unique

(*Completo, mas não único*). Pressionar uma terceira vez `⇐⇒` irá fazer com que a tela seja dividida em duas seções e que os arquivos que tenham nomes similares apareçam na seção de baixo, no *buffer* `Completions`. Vá digitando trechos que eliminem as ambigüidades até obter o arquivo desejado. Conseguindo isso, tecla `⇐`.

Ele irá carregar um *buffer* com o arquivo em questão. Em geral, o modo apresentado será `Text`, mas outros modos poderão aparecer conforme o tipo de documento. Um exemplo de `EMACS` com um *buffer* de arquivo pode ser visto no Trecho de Código 7.3.2, Página 205.


```

File Edit Options Buffers Tools TeX Help
% www.cenapad.unicamp.br/servicos/treinamentos/tutorial_unix
% http://proaluno.if.usp.br/minicursos/mini03/mini03/mini03.html





\chapter{Editores de texto}





Uma das principais ferramentas, não apenas para a administração, mas
para o uso em geral, dentro do GNU/Linux é o editor de texto. No caso,
entende-se editor de texto ao programa que permita a digitação,
gravação e correção de textos em formato ASCII puro (diferentemente
dos processadores de texto como o MS-Office ou o OpenOffice.org). Com
o uso de um editor de texto e as ferramentas corretas, você pode ir
desde criar páginas Web e \emph{emails} até a editar novos programas e
criar textos de alta qualidade\footnote{No caso, com ferramentas como
DocBook e \LaTeX{}}.




Esse capítulo será dedicado a uma introdução aos dois principais
editores de texto para Unix e GNU/Linux: o \texttt{vi} (e clones como
\texttt{vim} e \texttt{elvis}) e o GNU \texttt{emacs} (e seu principal
clone, o \texttt{jed}). Na prática, existem muitos outros editores,
como \texttt{pico}, \texttt{nano}, \texttt{joe} e até ferramentas
visuais, como \texttt{kate} e outras. Mas esses dois são facilmente
-uul(DOS)---F1 cap7.tex (LaTeX)---L1--Top-----
Loading tex-mode...done

```

Trecho de Código 7.3.2: Janela do emacs com *buffer*



Uma dica é que você sempre pode sair do *minibuffer* usando o atalho C-g ( + ). Além disso, você pode passar do *minibuffer* para o *buffer* usando C-x o ( +  e em seguida 0). Você pode abrir vários *buffers* de uma seção do emacs. Na Seção 7.3.6, na Página 208, veremos mais sobre como trabalhar com vários *buffers* abertos ao mesmo tempo.











Para salvar um arquivo, utilize o comando C-x C-s ( +  e  + ). Se o *buffer* em questão for de um arquivo, ele irá salvar no arquivo em questão o conteúdo do *buffer*. Caso contrário, ele irá pedir um nome de arquivo ao qual salvar o conteúdo do *buffer*.




É possível salvar vários *buffers* de uma vez, usando-se C-x s ( +  e ). Ele irá mostrar uma mensagem como a seguinte:




```
Save file /mnt/CursoGNU/Linux/cap7.tex? (y, n, !, ., q, C-r, d or C-h)
```















Onde:










- Pressionar  ou  salva o *buffer* atual;

- Pressionar  ou  pula o *buffer* atual (não salva o *buffer*);
-  abandona o restante dos buffers (*não os salva*);
- C-g ( + ) cancela o comando como um todo, embora não volte atrás nos arquivos que já tenham sido salvos;
-  salva todos os *buffers* ainda não salvos;
- C-r ( + ) mostra o *buffer* a ser salvo;
- d () gera um *diff* entre a versão a ser salva e a última versão salva;
- . () salva o *buffer* a ser salvo e sai do comando;

Para salvar um *buffer* em outro arquivo, utilize C-x w ( +  e ). Esse comando irá lhe pedir um nome de arquivo no qual o *buffer* será salvo, podendo pedir uma confirmação de sobrescrita caso venha a ser necessário. Nesse caso, responda **yes** (digite **yes** no *minibuffer*) caso deseje que esse arquivo seja sobrescrito. Caso contrário, responda **no**.

Para fechar um *buffer*, utilize C-x k ( +  e ). Esse comando irá fechar *apenas o buffer*. Se o *buffer* tiver sido alterado, ele irá pedir uma confirmação similar à mostrada nos casos anteriores. Responda **y** ou **n** conforme seu desejo.

Para alternar entre *buffers*, você pode utilizar C-x b ( +  e ). Ele irá pedir o nome do *buffer* para o qual você deseja ir, aceitando como *default* o último *buffer* para o qual você alternou. Nesse caso, basta pressionar  . De outro modo, digite o nome do arquivo (sem caminho) que deseja visualizar e aperte  . Você pode utilizar  e *completions* normalmente. Se quiser antes confirmar o *buffer* para o qual deseja ir, utilize C-x C-b ( +  e  + ) para exibir a lista de *buffers* atualmente carregados no **emacs**. Nesse caso, utilize C-x 1 ( +  e  + ) para fechar a lista de *buffers*.

Para sair do **emacs**, utilize C-x C-c ( +  e  + ). Ele irá lhe pedir uma confirmação similar à do comando C-x s ( +  e ), mas com uma diferença: se você escolher alguma das opções de saída, como **!** ou **Q**, ele realiza a operação e sai do **emacs**. A única forma de impedir essa saída é com C-q ( + ).

Para desfazer qualquer alteração, utilize **C+₋** (**Ctrl** + **Q**). *Não utilize o atalho normal de desfazer, **Ctrl** + **Z** .* Esse comando é usado para minimizar janelas no X-Windows.

7.3.4 Pesquisa e substituição


Para pesquisar um determinado trecho de informação, use **C-s** (**Ctrl** + **S**). Vai aparecer no *minibuffer* um *prompt* **I-search:**. Em seguida digite o texto digitado. Ele irá ressaltar o conteúdo que foi pesquisado dentro do **I-search:**. Para seguir pesquisando, tecle novamente **C-s** (**Ctrl** + **S**), até localizar o conteúdo desejado. Nesse caso, tecle **↵**.

Para pesquisar e substituir, use **M-%** (**Ctrl** + **%**). Aparece no *frame-buffer* o *prompt* *Query string:*. Digite o texto a ser substituído e em seguida o texto a ser colocado no lugar. Todas as entradas que “casem” com o critério desejado irão ser ressaltadas. Elas serão selecionadas uma a uma e será perguntado se deseja alterar os dados. Para alterar-se uma entrada, digite **y**, senão utilize **n**. Para cancelar as alterações, utilize **C-g** (**Ctrl** + **G**).

7.3.5 Cortar, Copiar, Colar e Apagar

Da mesma forma que o **vi**, a seleção do texto no **emacs** não lembra em nada a forma de selecionar-se no WindowsTM. Mas ela ainda assim é um tanto melhor que o padrão adotado por programas mais antigos. Para começar a selecionar um texto, aperte **C-Espaço** (**Ctrl** + **Space**) no local aonde ele irá começar a copiar. O *minibuffer* irá mostrar a mensagem “**Mark set**”. Vá se movendo até o final do trecho a ser copiado (em algumas versões do EMACS, a seleção tem sua cor de fundo alterada, mas não são todas). Use então **C-w** (**Ctrl** + **w**) para recortar o texto e **M-w** (**Meta** + **w**) para copiar o texto.













Após isso, vá para a posição no seu *buffer* atual ou em outro *buffer* no **emacs**, e em alguns casos até para outros programas no ambiente operacional aonde o **emacs** está rodando (como o X-Windows). No caso do **emacs**, para colar o texto, basta usar **C-x** y(**Ctrl** + **X** e **Y**) no ponto aonde deseja-se colar o texto.

Para apagar um texto, você pode usar a seleção normalmente, e utilizar  para apagar a seleção.

7.3.6 Truques de edição

O `emacs` é extremamente poderoso, tão poderoso que alguns de seus maiores defensores já não o consideram apenas um editor de texto, mas quase um *ambiente operacional*²². A poderosa linguagem EMACS LISP ofereceu muitas coisas interessantes, como GNUS (Cliente de News), ERC (Cliente IRC) e Emacs/W3 (Navegador HTTP) embutidos no EMACS. Se o leitor desejar saber mais sobre esses pacotes, a Internet está repleta de documentação sobre eles.

Vamos trabalhar com truques a nível de uso do EMACS.



O primeiro truque interessante é dividir a janela. Você pode usar `C-x 2` ( +  e ) para “fracionar” a janela e, com isso, poder trabalhar dois documentos em paralelo. Ao dividir a janela, normalmente ela fica as duas “partes” da janela no mesmo *buffer*, mas você pode alternar elas independentemente uma da outra. Para alternar entre “partes” da janela, use `C-x o` ( +  e ). Nesse caso, você perceberá que as janelas divididas são destacada e cada uma possui seu próprio *modeline*. Pode-se dividir uma mesma janela várias vezes, e também pode-se dividir a janela na vertical (`C-x 3` —  +  e ). Quando você não quiser mais as divisões, digite `C-x 1` ( +  e ) para desfazer as divisões.


Atenção: Quando você desfizer as divisões, *todas* as divisões serão desfeitas. *Não é possível* escolher quais divisões a serem desfeitas.

Outro truque bastante interessante são os “comandos longos”.

Como já foi dito, o EMACS utiliza uma linguagem de programação de *macros* muito poderosa, o EMACS LISP. Na prática, quase todos os





²²Pode-se considerar um ambiente operacional como o local aonde uma pessoa trabalha no computador. Nessa definição, uma interface gráfica ou um *shell* são considerados ambientes operacionais

comandos do EMACS são também comandos EMACS LISP. Para usar-se tais comandos, utilize o *minibuffer*. Um exemplo de comando útil é o `comment-region`. Selecione um trecho de texto (no EMACS chamado de *region* — região) e chame o *minibuffer* usando `M-x` ( + ). Nele, digite o comando `comment-region`. Ele irá comentar o texto selecionado para você.

Importante notar que pode-se usar o *minibuffer* e o *completion* para obter-se uma listagem de todos os “comandos longos” dentro do EMACS. Basta, no *minibuffer*, pressionar  duas vezes, o que irá chamar o *completion*.

7.3.7 Mudando a codificação do arquivo

Algumas vezes, você terá que mudar a codificação do arquivo para editar arquivos específicos no seu GNU/Linux. Por exemplo, você poderá precisar editar um *script* de *logon* na rede SaMBa ou um arquivo em Unicode no seu EMACS. Se o arquivo em questão for aberto, não há problemas: o EMACS detecta automaticamente a codificação (ou *encoding*) do arquivo. Mas se for um arquivo novo, será um pouco mais complicado que isso.

Para modificar a codificação do arquivo, primeiro altere a do *buffer* em questão, usando o comando `set-buffer-file-coding-system` no *minibuffer*, ou então use o atalho de teclado `C-x RET f` ( +  , em seguida  e por fim ). No *minibuffer* irá aparecer a mensagem `Coding system for visited file (default, nil):`. Você pode usar o *completion* para procurar a codificação desejada. Em geral, no Brasil usará-se as codificações `iso-latin-1` e `utf-8`. Em ambos os casos, você pode adicionar à codificação os termos `dos`, `unix` ou `mac` se precisar que o final de linha e outras especificações pessoais de cada sistema operacional sejam respeitadas pelo EMACS.

7.3.8 O arquivo `.emacs`

O arquivo `/.emacs` é o arquivo de configuração do EMACS. Na prática, ele é um *script* em EMACS LISP que permite que você ative recursos padrões ou não. Um exemplo de código de um `/.emacs` pode ser visto no Trecho de

Código 7.3.3, na Página 211

Na prática, existem dois tipos básicos de configurações: *modos* e *variáveis do ambiente*.

A primeira configuração ativa conjuntos de *modos* específicos. No EMACS, um modo determina o comportamento que o EMACS irá assumir conforme o tipo de arquivo em uso. Determina, entre outras coisas, características de realce de sintaxe (*syntax highlight*), tabulação, configurações para menus e outras coisas mais.

Normalmente, para ativar um módulo, você invoca o modo como um comando LISP, cercado por parênteses (`()`). Por exemplo, existe um módulo muito útil no EMACS, principalmente para aqueles acostumados com o uso da seleção como no WindowsTM, que é o `pc-selection-mode` (modo de seleção estilo PC). Para ativar esse módulo, utilize tanto no *framebuffer* quando no arquivo `/.emacs` o comando citado. No arquivo `/.emacs`, utilize `(pc-selection-mode)`.

Algumas vezes, os comandos de modo, principalmente nos chamados *modos menores* (*minor modes*) (módulos que “apenas” oferecem recursos extras aos modos sem serem elas próprias módulos) podem exigir que o modo receba parâmetros. Em EMACS LISP um parâmetro é qualquer coisa que não seja um comando e que não esteja cercada por parênteses, o que determina uma *lista*. Parâmetros *string* podem ser cercados por aspas duplas`""` normalmente.

Por exemplo, você pode definir um tamanho padrão para a janela em operação usando os comandos `set-frame-height` e `set-frame-width`, colocando como parâmetro uma lista como os *frames* (janelas) desejados (ou `(selected-frame)` para usar a janela selecionada) e um tamanho em número de caracteres, como no exemplo abaixo:

```
(set-frame-height (selected-frame) 27)
(set-frame-width (selected-frame) 80)
```

.

Já as *váriaveis de ambiente* são definidas usando `setq` recebendo como parâmetros a variável e seu valor. Por exemplo, o EMACS normalmente não mostra, em modo gráfico, um nome de janela muito interativo (mostra

```

; Add this to your .emacs or .xemacs/init.el file.
...
(autoload 'ruby-mode "ruby-mode" "Ruby editing mode." t)
(add-to-list 'auto-mode-alist '("\\.rb$" . ruby-mode))
(add-to-list 'interpreter-mode-alist '("ruby" . ruby-mode))
(set-default-font "-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1")
(defun ruby-eval-buffer () (interactive)
  "Evaluate the buffer with ruby."
  (shell-command-on-region (point-min) (point-max) "ruby"))
...
(require 'pc-select)
(require 'delsel)
...
(delete-selection-mode +1)
(pc-selection-mode)
(setq frame-title-format "Emacs/Linux por Fábio Costa - %b")
...
(define-key global-map [end] 'end-of-line)
(define-key global-map [home] 'beginning-of-line)
(define-key global-map [C-end] 'end-of-buffer)
(define-key global-map [C-home] 'beginning-of-buffer)
(define-key global-map [backspace] 'delete-backward-char)
(define-key global-map [delete] 'delete-char)
(global-unset-key "\352")
(global-unset-key "\343")
(global-unset-key "\C-j")
(global-unset-key "\M-f")
(global-unset-key "\M-r")
(global-unset-key "\M-s\M-s")
(global-set-key "\C-l" (quote downcase-word))
(global-set-key "\M-l" (quote downcase-region))
(global-set-key "\M-u" (quote upcase-region))
(global-set-key "\352" (quote set-justification-full))
(global-set-key "\343" (quote center-line))
(global-set-key "\C-j" (quote justify-current-line))
(global-set-key "\M-f" (quote search-forward-regexp))
(global-set-key "\M-r" (quote replace-regexp))
(global-set-key "\C-x\M-b" (quote center-block-text))
(global-set-key "\C-x\C-u" (quote capitalize-word))
(global-set-key [C-tab] (indent-according-to-mode))
(tool-bar-mode 1)
(set-frame-height (selected-frame) 27)
(set-frame-width (selected-frame) 80)
(setq w32-use-w32-font-dialog nil)
(setq line-number-mode t)
(setq column-number-mode t)
(setq inhibit-startup-message t)
(setq comint-completion-addsuffix t)
(setq kill-emacs-query-functions
  (cons (lambda () (yes-or-no-p "Deseja Realmente Sair do EMACS? "))
    kill-emacs-query-functions))
...

```

Trecho de Código 7.3.3: Exemplo de arquivo `/ .emacs`

alguma coisa como `emacs@hufflepuff`). Para melhorar, você pode utilizar a variável do ambiente `frame-title-format`, como demonstrado abaixo.

```
(setq frame-title-format "Emacs/Linux por Fábio Costa - %b")
```

Essas foram apenas introduções ao uso de editores de texto que permitam ao leitor trabalhar confortavelmente em qualquer um deles conforme suas necessidades e preferências. O leitor deve, portanto, verificar qual dos dois é o que mais se adapta a seu gosto e pesquisar mais profundamente sobre eles.

De qualquer modo, creio que nosso objetivo aqui está cumprido. Vamos agora abandonar um pouco o modo texto e ver como funciona a Interface Gráfica no GNU/Linux.

Capítulo 8

Interfaces Gráficas

8.1 O que é o X-Windows

8.2 Configurando o X-Windows

8.3 O arquivo `/etc/X11R6/Xorgconf`

8.4 O arquivo `.Xresources`

8.5 Algumas interfaces interessantes

8.5.1 KDE

8.5.2 GNOME

8.5.3 WindowMaker

8.5.4 IceWM

8.5.5 Blackbox

Capítulo 9

Instalando Programas

9.1 Programas em GNU/Linux

9.2 Compilando do Fonte

9.2.1 Seqüência mágica

9.2.2 O utilitário make

9.2.3 Porque no `/usr/local` e não no `/usr`

9.2.4 Dificuldades de compilar no fonte

9.3 Sistemas de empacotamento

9.3.1 RPM - RedHat Package Manager

9.3.2 DEB

9.3.3 TGZ

9.4 Atualização via Internet

9.4.1 apt

9.4.2 smart

Capítulo 10

Serviços e inicialização

10.1 O que são “serviços”

10.1.1 Como “levantar” serviços

10.2 Um pouco sobre a inicialização do GNU/Linux

10.2.1 O arquivo `/etc/inittab`

10.2.2 Inicialização System V e BSD

10.2.3 *Runlevels*

10.2.4 *Login* e `ttys`

Capítulo 11

Ajuda

11.1 Como proceder com ajuda?

11.2 Porque tomar nota?

11.3 O diretório `/var/log`

11.4 Os comandos `uname` e `dmesg`

11.5 O log do sistema (*syslog*)

11.6 Fontes de ajuda:

11.6.1 Sites de suporte técnico

11.6.2 Fóruns

11.6.3 Listas de discussão

11.6.3.1 Um pouco sobre Netiqueta

11.7 Suporte telefônico

11.8 Documentações online

Capítulo 12

Mantendo-se atualizado

12.1 Sites de informação

12.2 Newsletters

Apêndice A

Licenciamento dessa obra

CREATIVE COMMONS — ATRIBUIÇÃO —
COMPARTILHAMENTO PELA MESMA LICENÇA 2.5

A INSTITUIÇÃO CREATIVE COMMONS NÃO É UM ESCRITÓRIO DE ADVOCACIA E NÃO PRESTA SERVIÇOS JURÍDICOS. A DISTRIBUIÇÃO DESTA LICENÇA NÃO ESTABELECE QUALQUER RELAÇÃO ADVOCATÍCIA. O CREATIVE COMMONS DISPONIBILIZA ESTA INFORMAÇÃO “NO ESTADO EM QUE SE ENCONTRA”. O CREATIVE COMMONS NÃO FAZ QUALQUER GARANTIA QUANTO ÀS INFORMAÇÕES DISPONIBILIZADAS E SE EXONERA DE QUALQUER RESPONSABILIDADE POR DANOS RESULTANTES DO SEU USO.

Licença

A OBRA (CONFORME DEFINIDA ABAIXO) É DISPONIBILIZADA DE ACORDO COM OS TERMOS DESTA LICENÇA PÚBLICA CREATIVE COMMONS (“CCPL” OU “LICENÇA”). A OBRA É PROTEGIDA POR DIREITO AUTORAL E/OU OUTRAS LEIS APLICÁVEIS. QUALQUER USO DA OBRA QUE NÃO O AUTORIZADO SOB ESTA LICENÇA OU PELA LEGISLAÇÃO AUTORAL É PROIBIDO.

AO EXERCER QUAISQUER DOS DIREITOS À OBRA AQUI CONCEDIDOS, VOCÊ ACEITA E CONCORDA FICAR OBRIGADO NOS TERMOS DESTA LICENÇA. O LICENCIANTE CONCEDE A VOCÊ OS DIREITOS AQUI CONTIDOS EM CONTRAPARTIDA À SUA ACEITAÇÃO DESTES TERMOS E CONDIÇÕES.

1. Definições

- (a) “Obra Coletiva” significa uma obra, tal como uma edição periódica, antologia ou enciclopédia, na qual a Obra em sua totalidade e de forma inalterada, em conjunto com um número de outras contribuições, constituindo obras independentes e separadas em si mesmas, são agregadas em um trabalho coletivo. Uma obra que constitua uma Obra Coletiva não será considerada Obra Derivada (conforme definido abaixo) para os propósitos desta licença.
- (b) “Obra Derivada” significa uma obra baseada sobre a Obra ou sobre a Obra e outras obras pré-existentes, tal como uma tradução, arranjo musical, dramatização, romantização, versão de filme, gravação de som, reprodução de obra artística, resumo, condensação ou qualquer outra forma na qual a Obra possa ser refeita, transformada ou adaptada, com a exceção de que uma obra que constitua uma Obra Coletiva não será considerada Obra Derivada para fins desta licença. Para evitar dúvidas, quando a Obra for uma composição musical ou gravação de som, a sincronização da Obra em relação cronometrada com uma imagem em movimento (?synching?) será considerada uma Obra Derivada para os propósitos desta licença.
- (c) “Licenciante” significa a pessoa física ou a jurídica que oferece a Obra sob os termos desta licença.
- (d) “Autor Original” significa a pessoa física ou jurídica que criou a Obra.

-
- (e) “Obra” significa a obra autoral, passível de proteção pelo direito autoral, oferecida sob os termos desta licença.
 - (f) “Você” significa a pessoa física ou jurídica exercendo direitos sob esta Licença que não tenha previamente violado os termos desta Licença com relação à Obra, ou que tenha recebido permissão expressa do Licenciante para exercer direitos sob esta Licença apesar de uma violação prévia.
 - (g) “Elementos da Licença” significa os principais atributos da licença correspondente, conforme escolhidos pelo licenciante e indicados no título desta licença: Atribuição, Compartilhamento pela Mesma Licença.
2. **Direitos de Uso Legítimo.** Nada nesta licença deve ser interpretado de modo a reduzir, limitar ou restringir quaisquer direitos relativos ao uso legítimo, ou outras limitações sobre os direitos exclusivos do titular de direitos autorais sob a legislação autoral ou quaisquer outras leis aplicáveis.
3. **Concessão da Licença.** O Licenciante concede a Você uma licença de abrangência mundial, sem royalties, não-exclusiva, perpétua (pela duração do direito autoral aplicável), sujeita aos termos e condições desta Licença, para exercer os direitos sobre a Obra definidos abaixo:
- (a) reproduzir a Obra, incorporar a Obra em uma ou mais Obras Coletivas e reproduzir a Obra quando incorporada em Obra Coletiva;
 - (b) criar e reproduzir Obras Derivadas;
 - (c) distribuir cópias ou gravações da Obra, exibir publicamente, executar publicamente e executar publicamente por meio de uma transmissão de áudio digital a Obra, inclusive quando incorporada em Obras Coletivas;
 - (d) distribuir cópias ou gravações de Obras Derivadas, exibir publicamente, executar publicamente e executar publicamente por meio de uma transmissão digital de áudio Obras Derivadas;
 - (e) De modo a tornar claras estas disposições, quando uma Obra for uma composição musical:

- i. **Royalties e execução pública.** O licenciante renuncia o seu direito exclusivo de coletar, seja individualmente ou através de entidades coletoras de direitos de execução (por exemplo, ECAD, ASCAp, BMI, SESAC), o valor dos seus direitos autorais pela execução pública da obra ou execução pública digital (por exemplo, webcasting) da Obra.
 - ii. **Royalties e Direitos fonomecânicos.** O licenciante renuncia o seu direito exclusivo de coletar, seja individualmente ou através de uma entidade designada como seu agente (por exemplo, a agência Harry Fox), royalties relativos a quaisquer gravações que Você criar da Obra (por exemplo, uma versão "cover") e distribuir, conforme as disposições aplicáveis de direito autoral.
- (f) **Direitos de Execução Digital pela internet (Webcasting) e royalties.** De modo a evitar dúvidas, quando a Obra for uma gravação de som, o Licenciante reserva o seu direito exclusivo de coletar, seja individualmente ou através de entidades coletoras de direitos de execução (por exemplo, SoundExchange ou ECAD), royalties e direitos autorais pela execução digital pública (por exemplo, Webcasting) da Obra, conforme as disposições aplicáveis de direito autoral, se a execução digital pública feita por Você for predominantemente intencionada ou direcionada à obtenção de vantagem comercial ou compensação monetária privada.
- Os direitos acima podem ser exercidos em todas as mídias e formatos, independente de serem conhecidos agora ou concebidos posteriormente. Os direitos acima incluem o direito de fazer modificações que forem tecnicamente necessárias para exercer os direitos em outras mídias, meios e formatos. Todos os direitos não concedidos expressamente pelo Licenciante ficam aqui reservados.
4. **Restrições.** A licença concedida na Seção 3 acima está expressamente sujeita e limitada aos seguintes termos:
- (a) Você pode distribuir, exibir publicamente, executar publicamente ou executar publicamente por meios digitais a Obra apenas sob os termos desta Licença, e Você deve incluir uma cópia desta licença, ou o Identificador Uniformizado de Recursos (Uniform Resource Identifier) para esta Licença, com cada cópia ou gravação

da Obra que Você distribuir, exibir publicamente, executar publicamente, ou executar publicamente por meios digitais. Você não poderá oferecer ou impor quaisquer termos sobre a Obra que alterem ou restrinjam os termos desta Licença ou o exercício dos direitos aqui concedidos aos destinatários. Você não poderá sublicenciar a Obra. Você deverá manter intactas todas as informações que se referem a esta Licença e à exclusão de garantias. Você não pode distribuir, exibir publicamente, executar publicamente ou executar publicamente por meios digitais a Obra com qualquer medida tecnológica que controle o acesso ou o uso da Obra de maneira inconsistente com os termos deste Acordo de Licença. O disposto acima se aplica à Obra enquanto incorporada em uma Obra Coletiva, mas isto não requer que a Obra Coletiva, à parte da Obra em si, esteja sujeita aos termos desta Licença. Se Você criar uma Obra Coletiva, em havendo notificação de qualquer Licenciante, Você deve, na medida do razoável, remover da Obra Coletiva qualquer crédito, conforme estipulado na cláusula 4 (c), quando solicitado. Se Você criar um trabalho derivado, em havendo aviso de qualquer Licenciante, Você deve, na medida do possível, retirar do trabalho derivado, qualquer crédito conforme estipulado na cláusula 4 (c), conforme solicitado.

- (b) Você pode distribuir, exibir publicamente, executar publicamente ou executar publicamente por meios digitais uma Obra Derivada somente sob os termos desta Licença, ou de uma versão posterior desta licença com os mesmos Elementos da Licença desta licença, ou de uma licença do Creative Commons internacional (iCommons) que contenha os mesmos Elementos da Licença desta Licença (por exemplo, Atribuição-Compartilhamento pela Mesma Licença 2.5 Japão). Você deve incluir uma cópia desta licença ou de outra licença especificada na sentença anterior, ou o Identificador Uniformizado de Recursos (Uniform Resource Identifier) para esta licença ou de outra licença especificada na sentença anterior, com cada cópia ou gravação de cada Obra Derivada que Você distribuir, exibir publicamente, executar publicamente ou executar publicamente por meios digitais. Você não poderá oferecer ou impor quaisquer termos sobre a Obra Derivada que alterem ou restrinjam os termos desta Licença ou o exercício dos direitos aqui

concedidos aos destinatários, e Você deverá manter intactas todas as informações que se refiram a esta Licença e à exclusão de garantias. Você não poderá distribuir, exhibir publicamente, executar publicamente ou executar publicamente por meios digitais a Obra Derivada com qualquer medida tecnológica que controle o acesso ou o uso da Obra de maneira inconsistente com os termos deste Acordo de Licença. O disposto acima se aplica à Obra Derivada quando incorporada em uma Obra Coletiva, mas isto não requer que a Obra Coletiva, à parte da Obra em si, esteja sujeita aos termos desta Licença.

- (c) Se Você distribuir, exhibir publicamente, executar publicamente ou executar publicamente por meios digitais a Obra ou qualquer Obra Derivada ou Obra Coletiva, Você deve manter intactas todas as informações relativas a direitos autorais sobre a Obra e exhibir, de forma razoável com relação ao meio ou mídia que Você está utilizando: (i) o nome do autor original (ou seu pseudônimo, se for o caso) se fornecido e/ou (ii) se o autor original e/ou o Licenciante designar outra parte ou partes (Ex.: um instituto patrocinador, órgão que publicou, periódico, etc.) para atribuição nas informações relativas aos direitos autorais do Licenciante, termos de serviço ou por outros meios razoáveis, o nome da parte ou partes; o título da Obra, se fornecido; na medida do razoável, o Identificador Uniformizado de Recursos (URI) que o Licenciante especificar para estar associado à Obra, se houver, exceto se o URI não se referir ao aviso de direitos autorais ou à informação sobre o regime de licenciamento da Obra; e no caso de Obra Derivada, crédito identificando o uso da Obra na Obra Derivada (exemplo: "Tradução Francesa da Obra de Autor Original", ou "Roteiro baseado na Obra original de Autor Original"). Tal crédito pode ser implementado de qualquer forma razoável; entretanto, no caso de Obra Derivada ou Obra Coletiva, este crédito aparecerá no mínimo onde qualquer outro crédito de autoria comparável aparecer e de modo ao menos tão proeminente quanto este outro crédito.

5. Declarações, Garantias e Exoneração

EXCETO QUANDO FOR DE OUTRA FORMA ACORDADO PELAS PARTES POR ESCRITO, O LICENCIANTE OFERECE A OBRA

?NO ESTADO EM QUE SE ENCONTRA? (AS IS) E NÃO PRESTA QUAISQUER GARANTIAS OU DECLARAÇÕES DE QUALQUER ESPÉCIE RELATIVAS AOS MATERIAIS, SEJAM ELAS EXPRESAS OU IMPLÍCITAS, DECORRENTES DA LEI OU QUAISQUER OUTRAS, INCLUINDO, SEM LIMITAÇÃO, QUAISQUER GARANTIAS SOBRE A TITULARIDADE DA OBRA, ADEQUAÇÃO PARA QUAISQUER PROPÓSITOS, NÃO-VIOLAÇÃO DE DIREITOS, OU INEXISTÊNCIA DE QUAISQUER DEFEITOS LATENTES, ACURACIDADE, PRESENÇA OU AUSÊNCIA DE ERROS, SEJAM ELES APARENTES OU OCULTOS. EM JURISDIÇÕES QUE NÃO ACEITEM A EXCLUSÃO DE GARANTIAS IMPLÍCITAS, ESTAS EXCLUSÕES PODEM NÃO SE APLICAR A VOCÊ.

6. **Limitação de Responsabilidade.** EXCETO NA EXTENSÃO EXIGIDA PELA LEI APLICÁVEL, EM NENHUMA CIRCUNSTÂNCIA O LICENCIANTE SERÁ RESPONSÁVEL PARA COM VOCÊ POR QUAISQUER DANOS, ESPECIAIS, INCIDENTAIS, CONSEQÜÊNCIAS, PUNITIVOS OU EXEMPLARES, ORIUNDOS DESTA LICENÇA OU DO USO DA OBRA, MESMO QUE O LICENCIANTE TENHA SIDO AVISADO SOBRE A POSSIBILIDADE DE TAIS DANOS.

7. **Terminação**

- (a) Esta Licença e os direitos aqui concedidos terminarão automaticamente no caso de qualquer violação dos termos desta Licença por Você. Pessoas físicas ou jurídicas que tenham recebido Obras Derivadas ou Obras Coletivas de Você sob esta Licença, entretanto, não terão suas licenças terminadas desde que tais pessoas físicas ou jurídicas permaneçam em total cumprimento com essas licenças. As Seções 1, 2, 5, 6, 7 e 8 subsistirão a qualquer terminação desta Licença.
- (b) Sujeito aos termos e condições dispostos acima, a licença aqui concedida é perpétua (pela duração do direito autoral aplicável à Obra). Não obstante o disposto acima, o Licenciante reserva-se o direito de difundir a Obra sob termos diferentes de licença ou de cessar a distribuição da Obra a qualquer momento; desde que, no entanto, quaisquer destas ações não sirvam como meio de retração desta Licença (ou de qualquer outra licença que tenha sido

concedida sob os termos desta Licença, ou que deva ser concedida sob os termos desta Licença) e esta Licença continuará válida e eficaz a não ser que seja terminada de acordo com o disposto acima.

8. Outras Disposições

- (a) Cada vez que Você distribuir ou executar publicamente por meios digitais a Obra ou uma Obra Coletiva, o Licenciante oferece ao destinatário uma licença da Obra nos mesmos termos e condições que a licença concedida a Você sob esta Licença.
- (b) Cada vez que Você distribuir ou executar publicamente por meios digitais uma Obra Derivada, o Licenciante oferece ao destinatário uma licença à Obra original nos mesmos termos e condições que foram concedidos a Você sob esta Licença.
- (c) Se qualquer disposição desta Licença for tida como inválida ou não-executável sob a lei aplicável, isto não afetará a validade ou a possibilidade de execução do restante dos termos desta Licença e, sem a necessidade de qualquer ação adicional das partes deste acordo, tal disposição será reformada na mínima extensão necessária para tal disposição tornar-se válida e executável.
- (d) Nenhum termo ou disposição desta Licença será considerado renunciado e nenhuma violação será considerada consentida, a não ser que tal renúncia ou consentimento seja feito por escrito e assinado pela parte que será afetada por tal renúncia ou consentimento.
- (e) Esta Licença representa o acordo integral entre as partes com respeito à Obra aqui licenciada. Não há entendimentos, acordos ou declarações relativas à Obra que não estejam especificadas aqui. O Licenciante não será obrigado por nenhuma disposição adicional que possa aparecer em quaisquer comunicações provenientes de Você. Esta Licença não pode ser modificada sem o mútuo acordo, por escrito, entre o Licenciante e Você.

O Creative Commons não é uma parte desta Licença e não presta qualquer garantia relacionada à Obra. O Creative Commons não será responsável perante Você ou qualquer outra parte por quaisquer danos, incluindo, sem limitação, danos gerais, especiais, incidentais ou consequentes, originados com relação a esta licença. Não obstante as duas frases anteriores, se o Creative Commons tiver expressamente se identificado como o Licenciante, ele deverá ter todos os direitos e obrigações do Licenciante.

Exceto para o propósito delimitado de indicar ao público que a Obra é licenciada sob a CCPL (Licença Pública Creative Commons), nenhuma parte deverá utilizar a marca "Creative Commons" ou qualquer outra marca ou logo relacionado ao Creative Commons sem consentimento prévio e por escrito do Creative Commons. Qualquer uso permitido deverá ser de acordo com as diretrizes do Creative Commons de utilização da marca então válidas, conforme sejam publicadas em seu website ou de outro modo disponibilizadas periodicamente mediante solicitação.

O Creative Commons pode ser contactado pelo endereço: <http://creativecommons.org/>.

Referências Bibliográficas

ALECRIM, E. *A história do Linux*. 2003. Disponível em: <<http://www.infowester.com/linux5.php>>. Acesso em: 12 de Abril de 2006.

ARCOMANO, R. *Linux Wireless HOWTO*. The Linux Documentation Project, 20002. Disponível em: <<http://www.tldp.org/HOWTO/Wireless-HOWTO.html>>. Acesso em: 23 de Abril de 2006.

CASHA, R. *Filesystems for Linux*. 2001. Disponível em: <<http://linux.org.mt/article/filesystems>>. Acesso em: 20 de Abril de 2006.

CASTELLS, M. *A Sociedade em Rede*. São Paulo, SP: Paz e Terra, 2000.

CHASSELL, R. J. *Programming in Emacs Lisp*. Segunda. Boston, USA: GNU Press, 2006. Disponível em: <<http://www.gnu.org/software/emacs/emacs-lisp-intro/emacs-lisp-intro.html>>. Acesso em: 26 de Abril de 2006.

CINEIROS, H. *The Linux Manual, versão 4.0*. 2005. Disponível em: <<http://www.devin.com.br/eitch/tlm4/>>. Acesso em: 21 de Abril de 2006.

COELHO, A. B. G. *Escalonamento de processos em Linux*. 2003. Disponível em: <<http://www.ginux.ufla.br/documentacao/bibginux/Alexandre-Escalonamento.pdf>>. Acesso em: 23 de Abril de 2006.

COELHO, R. S. de A. *Editor vi — Guia de Consulta Rápida*. São Paulo, SP: Novatec, 2002.

COMER, D. E. *Interligação em rede com TCP/IP volume 1*. 3rd. ed. São Paulo: Campus, 2001. 672 p. ISBN 85-352-0270-6.

DEUTSCH, P. *RFC 1950 – GZIP file format specification version 4.3*. 1996. Disponível em: <<http://www.ietf.org/rfc/rfc1950.txt>>. Acesso em: 23 de Abril de 2006.

DEUTSCH, P. *RFC 1951 — DEFLATE Compressed Data Format Specification version 1.3*. 1996. Disponível em: <<http://www.ietf.org/rfc/rfc1951.txt>>. Acesso em: 23 de Abril de 2006.

DEUTSCH, P.; GAILLY, J.-L. *RFC 1950 — ZLIB Compressed Data Format Specification version 3.3*. 1996. Disponível em: <<http://www.ietf.org/rfc/rfc1950.txt>>. Acesso em: 23 de Abril de 2006.

DRAKE, J. *Linux Networking HOWTO*. The Linux Documentation Project, 1999. Disponível em: <<http://www.tldp.org/HOWTO/NET3-4-HOWTO.html>>. Acesso em: 23 de Abril de 2006.

FERREIRA, R. E. *Linux: Guia do Administrador do Sistema*. São Paulo, SP: Novatec, 2003.

FREE SOFTWARE FOUNDATION. *GNU Emacs Lisp Reference Manual*. Boston, USA: GNU Press, 2002. Disponível em: <<http://www.gnu.org/software/emacs/emacs-lisp-intro/emacs-lisp-intro.html>>. Acesso em: 26 de Abril de 2006.

FREE SOFTWARE FOUNDATION. *GNU Emacs manual*. Boston, USA: GNU Press, 2006. Disponível em: <<http://www.gnu.org/software/emacs/manual/>>. Acesso em: 26 de Abril de 2006.

FREE STANDARDS GROUP. *Filesystem Hierarchy Standard, version 2.3*. 2004. Disponível em: <<http://www.pathname.com/fhs/pub/fhs-2.3.html>>. Acesso em: 18 de Abril de 2006.

FREE STANDARDS GROUP. *Linux Standard Base Core Specification 3.1*. 2005. Disponível em: <http://refspecs.freestandards.org/LSB_3.1.0/LSB-Core-generic/LSB-Core-generic/book1.html>. Acesso em: 18 de Abril de 2006.

GORTMAKER, P. *Linux Ethernet HOWTO*. The Linux Documentation Project, 2003. Disponível em: <<http://www.tldp.org/HOWTO/Ethernet-HOWTO.html>>. Acesso em: 23 de Abril de 2006.

JACKSON, M. H. *Linux Shadow Password HOWTO*. The Linux Documentation Project, 1996. Disponível em: <<http://www.tldp.org/HOWTO/Shadow-Password-HOWTO.html>>. Acesso em: 21 de Abril de 2006.

JARGAS, A. M. *Expressões Regulares — Guia de Referência Rápida*. São Paulo, SP: Novatec, 2003. Disponível em: <<http://guia-er.sourceforge.net/>>. Acesso em: 12 de Abril de 2006.

KIRCH, O.; DAWSON, T. *The Linux Network Administrators' Guide, Second Edition*. The Linux Documentation Project, 2000. Disponível em: <<http://www.tldp.org/LDP/nag2/index.html>>. Acesso em: 23 de Abril de 2006.

KLIGER, J.; CRAIG, W. *vilearn — The interactive vi tutorial*. 1992. Disponível em: <<http://vilearn.org>>. Acesso em: 24 de Abril de 2006.

LINUX DOCUMENTATION PROJECT. *Linux HOWTO Project*. The Linux Documentation Project, 1999. Disponível em: <<http://www.tldp.org/HOWTO/NET3-4-HOWTO.html>>. Acesso em: 23 de Abril de 2006.

LINUX DOCUMENTATION PROJECT. 2006. Disponível em: <<http://www.tldp.org>>. Acesso em: 21 de Abril de 2006.

LINUX SEM MISTÉRIO. *Sudo, o que é isso?* 2005. Disponível em: <http://linux-sem-misterio.blogspot.com/2006/02/sudo-o-que-isso_-27.html>. Acesso em: 14 de Abril de 2006.

LINUX WEEKLY JOURNAL. *The LWN.net Linux Distribution List*. 2006. Disponível em: <<http://lwn.net/Distributions/index.php3>>. Acesso em: 12 de Abril de 2006.

MOOLENAAR, B. *vim User Manual*. [s.n.], 2003. Disponível em: <http://vimdoc.sourceforge.net/html/doc/usr_toc.html>. Acesso em: 26 de Abril de 2006.

MORIMOTO, C. E. *Entendendo e Dominando o Linux — 7ª Edição*. São Paulo, SP: Guia do Hardware.net Press, 2005. Disponível em: <<http://www.guiadohardware.net/ebooks/linux/>>. Acesso em: 22 de Abril de 2006.

MORIMOTO, C. E. *Linux: Redes e servidores, Guia Prático — 2ª Edição*. Porto Alegre, RS: Sul Editores, 2006.

PROJETO BZIP2. 2005. Disponível em: <<http://www.bzip.org/>>. Acesso em: 23 de Abril de 2006.

PROJETO GNU. *GNU Info Manual 4.8*. 2006. Disponível em: <<http://www.gnu.org/software/texinfo/>>. Acesso em: 12 de Abril de 2006.

PROJETO GNU. *GNU não é Unix!* 2006. Disponível em: <<http://www.gnu.org/home.pt.html>>. Acesso em: 12 de Abril de 2006.

PROJETO GZIP. 2003. Disponível em: <<http://www.gzip.org/>>. Acesso em: 23 de Abril de 2006.

RAYMOND, E. S. *The Cathedral and the Bazaar*. Sebastopol, USA: O'Reilly, 2000. Disponível em: <<http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/>>. Acesso em: 12 de Abril de 2006.

RAYMOND, E. S. *Jargon File — The New Hacker's Dictionary (version 4.4.7)*. Sebastopol, USA: O'Reilly, 2003. Disponível em: <<http://www.catb.org/esr/jargon/html/frames.html>>. Acesso em: 26 de Abril de 2006.

RAYMOND, E. S. *Jargon File*. 2006. Disponível em: <<http://www.catb.org/esr/jargon/>>. Acesso em: 12 de Abril de 2006.

SETUID.ORG. 2006. Disponível em: <<http://setuid.org/>>. Acesso em: 14 de Abril de 2006.

SILVA, G. M. da. *Guia FOCA GNU/Linux*. 2005. Disponível em: <<http://focalinux.cipsga.org.br/>>. Acesso em: 14 de Abril de 2006.

SILVA, G. M. da. *Guia FOCA GNU/Linux Nível Avançado*. 2005. Disponível em: <<http://focalinux.cipsga.org.br/guia/avancado/index.htm>>. Acesso em: 14 de Abril de 2006.

STALLMAN, R. *Anúncio Inicial do Projeto GNU*. 1983. Disponível em: <<http://www.gnu.org/gnu/initial-announcement.pt.html>>. Acesso em: 12 de Abril de 2006.

STALLMAN, R. The GNU operating system and the free software movement. In: _____. *Open Sources : Voices from the Open Source Revolution*. Sebastopol, USA: O'Reilly, 1999. Disponível em: <<http://www.oreilly.com/catalog/opensources/book/stallman.html>>. Acesso em: 12 de Abril de 2006.

STEVENS, W. R. *TCP/IP Illustrated, Volume 1 - The Protocols*. [S.l.]: Addison Wesley, 1999. 576 p.

THE INTERNET ENGINEERING TASK FORCE. *Request for Comments 761 – Transmission Control Protocol*. 1980. Disponível em: <<http://www.ietf.org/rfc/rfc0761.txt>>. Acesso em: 2 de Maio de 2006.

THE INTERNET ENGINEERING TASK FORCE. *Request for Comments 791 – Internet Protocol*. 1981. Disponível em: <<http://www.ietf.org/rfc/rfc0791.txt>>. Acesso em: 2 de Maio de 2006.

THE INTERNET ENGINEERING TASK FORCE. *Request for Comments 1918 – Address Allocation for Private Internets*. 1996. Disponível em: <<http://www.ietf.org/rfc/rfc1918.txt>>. Acesso em: 4 de Maio de 2006.

THE OPEN GROUP. *Regular Expression*. 1997. Disponível em: <<http://www.opengroup.org/onlinepubs/007908799/xbd/re.html>>. Acesso em: 12 de Abril de 2006.

THE OPEN GROUP. *The Single UNIX Specification*. 2006. Disponível em: <http://www.unix.org/what_is_unix/single_unix_specification.html>. Acesso em: 24 de Abril de 2006.

UBUNTU LINUX. *Ubuntu Linux*. 2006. Disponível em: <<http://www.ubuntu.com/>>. Acesso em: 12 de Abril de 2006.

UNIX FOR ADVANCED USERS. *Manipulating Files – What’s the sticky bit?* 2006. Disponível em: <<http://www.ussg.iu.edu/UAU/files/sticky.html>>. Acesso em: 14 de Abril de 2006.

WIKIPEDIA EM PORTUGUÊS. *Usenet*. 2006. Disponível em: <<http://pt.wikipedia.org/wiki/Usenet>>. Acesso em: 12 de Abril de 2006.

WILLIAMS, S. *Free as in Freedom - Richard Stallman’s Crusade for Free Software*. Sebastopol, USA: O’Reilly, 2002. Disponível em: <<http://www.oreilly.com/openbook/freedom/>>. Acesso em: 12 de Abril de 2006.

WIRZENIUS, L. *et al.* Creating a user. In: _____. The Linux Documentation Project, 2005. cap. 11. Disponível em: <<http://www.tldp.org/LDP/sag/html/adduser.html>>. Acesso em: 21 de Abril de 2006.

WIRZENIUS, L. *et al.* (Ed.). *The Linux System Administrators’ Guide, version 0.9*. The Linux Documentation Project, 2005. Disponível em: <<http://www.tldp.org/LDP/sag/html/index.html>>. Acesso em: 21 de Abril de 2006.

ZAGO, A. F. *FAQ sobre processos em geral, prioridades, como iniciar e parar, logs...* 2006. Disponível em: <<http://www.zago.eti.br/processos.txt>>. Acesso em: 22 de Abril de 2006.