José Edson Moreno Júnior

Linux

Módulo 1

Fundamentos para Administradores de Sistemas

2026

Direitos autorais

Copyright © 2026 José Edson Moreno Júnior.

Todos os direitos reservados.

Este documento, incluindo seu texto, estrutura, diagramas, imagens e exemplos originais, é protegido pela legislação brasileira de direitos autorais.

É permitida a citação de pequenos trechos, desde que acompanhada da identificação do autor e da referência completa da obra.

É proibida a reprodução integral, redistribuição, modificação, comercialização ou apresentação deste conteúdo como obra de terceiros sem autorização expressa do autor.

Título: Linux Módulo 1 - Fundamentos para Administradores de Sistemas
Autor: José Edson Moreno Júnior
Versão: 1.4
Data de publicação: 30 de julho de 2026
Identificador do documento: MORENO-LINUX-MOD1-2026-001

Sumário

Sumário **3**

Introdução **15**
Objetivos do módulo **15**
Características gerais do Linux **15**

1 DISCOS E PARTIÇÕES **18**
1.1 Objetivos **18**
1.2 Pré-requisitos **18**
1.3 Conceitos fundamentais **18**
1.4 Identificando discos e partições **19**
1.5 Tabelas de partição: MBR e GPT **19**
1.6 Ferramentas de particionamento **20**
1.7 Laboratório: criando um disco virtual **20**
1.8 Particionando o disco virtual **22**
1.9 Sistemas de arquivos **23**
1.9.1 EXT2, EXT3 e EXT4 23
1.9.2 XFS 24
1.9.3 VFAT 24
1.9.4 ReiserFS 24
1.9.5 Swap 24
1.10 Formatando as partições **24**
1.11 Montando sistemas de arquivos **25**
1.12 Ativando e verificando swap **26**
1.13 Montagem persistente com /etc/fstab **26**
1.14 Visão inicial de LVM **27**
1.15 Diagnóstico básico **27**
1.16 Limpeza do laboratório **28**
1.17 Resumo **28**
1.18 Perguntas de revisão **29**
1.19 Desafio opcional **29**

2 BACKUP BÁSICO **30**
2.1 Backup como processo **31**
2.2 Cópia, backup, snapshot e restauração **31**
2.3 O que proteger **32**
2.4 Local de armazenamento **32**

2.5 Retenção . . . 33
2.6 Backup com tar . . . 33
2.7 Caminhos absolutos e relativos no tar . . . 34
2.8 Backup com rsync . . . 34
2.8.1 Teste sem copiar . . . 35
2.8.2 Cuidado com –delete . . . 35
2.9 Integridade e verificação . . . 35
2.10 Restauração . . . 36
2.11 Automação e logs . . . 36
2.12 Diagnóstico básico . . . 37
2.13 Laboratório guiado: backup e restauração . . . 37
2.13.1 1. Preparar o ambiente . . . 37
2.13.2 2. Criar backup com tar . . . 38
2.13.3 3. Validar o arquivo . . . 38
2.13.4 4. Restaurar em diretório separado . . . 38
2.13.5 5. Sincronizar com rsync . . . 38
2.13.6 6. Simular perda e recuperar . . . 38
2.13.7 7. Limpeza do laboratório . . . 39
2.14 Resumo . . . 39
2.15 Perguntas de revisão . . . 39
2.16 Desafio opcional . . . 39

3 ESTRUTURA DO SISTEMA DE ARQUIVOS . . . 40
Objetivos do capítulo . . . 40
Pré-requisitos . . . 40
3.1 Diretório raiz (/) e Sistemas de arquivos . . . 42
3.1.1 /bin . . . 42
3.1.2 /boot . . . 43
3.1.3 /dev . . . 44
3.1.4 /etc . . . 45
3.1.5 /home . . . 45
3.1.6 /lib . . . 46
3.1.7 /media . . . 46
3.1.8 /mnt . . . 46
3.1.9 /opt . . . 47
3.1.10 /proc . . . 47
3.1.11 /sys . . . 48
3.1.12 /root . . . 48
3.1.13 /sbin . . . 48
3.1.14 /tmp . . . 49

3.1.15 /usr . . . 49
3.1.16 /var . . . 50
3.1.17 /run . . . 50
3.1.18 lost+found . . . 51
Resumo . . . **51**
Perguntas de revisão . . . **51**
Desafio opcional . . . **52**

4 USUÁRIOS, GRUPOS E PERMISSÕES . . . **53**
4.1 Criando e gerenciando usuários e grupos no GNU/Linux . . . **53**
4.1.1 Gerenciando usuários . . . 54
4.1.2 Gerenciando grupos . . . 58
4.2 Permissões e Donos de Arquivos . . . **59**
4.3 Comandos de manipulação de permissões . . . **61**
4.3.1 chmod, chown e chgrp . . . 61
4.3.2 umask . . . 63
4.4 Permissões de Acesso Especiais . . . **64**
4.4.1 *setuid, setgid, sticky bit* . . . 64

5 PROCESSOS LINUX . . . **68**
5.1 Conceito de processo . . . **68**
5.2 Listando processos . . . **69**
5.3 Estados comuns de processos . . . **70**
5.4 Monitoramento interativo . . . **70**
5.5 Localizando processos . . . **70**
5.6 Sinais e finalização de processos . . . **71**
5.7 Primeiro plano e segundo plano . . . **72**
5.8 Prioridade com nice e renice . . . **72**
5.9 Laboratório guiado . . . **73**

6 COMANDOS BÁSICOS . . . **75**
6.1 Objetivos do capítulo . . . **75**
6.2 Pré-requisitos . . . **75**
6.3 Sintaxe dos comandos . . . **75**
6.3.1 **man** (manual) . . . 76
6.3.2 **pwd** (print working directory) . . . 76
6.3.3 **ls** (list) . . . 76
6.3.4 **cd** (change directory) . . . 77
6.3.5 **mkdir** (make directory) . . . 78
6.3.6 **rmdir** (remove directory) . . . 78
6.3.7 **touch** . . . 78

6.3.8 **rm** (remove) . . . 79
6.3.9 **cp** (copy) . . . 80
6.3.10 **mv** (move) . . . 80
6.3.11 **cat** (concatenate) . . . 81
6.3.12 **find** . . . 81
6.3.13 **clear** . . . 82
6.3.14 **exit** . . . 82
6.3.15 **echo** . . . 82
6.3.16 **date** . . . 83
6.3.17 **df** . . . 83
6.3.18 **du** . . . 84
6.3.19 **cal** . . . 85
6.3.20 **uptime** . . . 85
6.3.21 **time** . . . 85
6.3.22 **ln** (links) . . . 85
6.4 Comandos manipulação texto . . . 86
6.4.1 **Redirecionamento E/S** . . . 86
6.4.2 **tail** . . . 87
6.4.3 **head** . . . 88
6.4.4 **more** . . . 88
6.4.5 **less** . . . 88
6.4.6 **sort** . . . 89
6.4.7 **grep e grep -E** . . . 89
6.4.8 **vim** (Vi IMproved) . . . 90
6.4.9 **awk** ou ***gawk*** . . . 92
6.5 Resumo . . . 98
6.6 Perguntas de revisão . . . 98
6.7 Desafio opcional . . . 99

7 PIPES E REDIRECIONAMENTO . . . 100
7.1 Objetivos do capítulo . . . 100
7.2 Pré-requisitos . . . 100
7.3 Entrada e saída padrão . . . 100
7.4 Pipe . . . 101
7.5 Redirecionamento de saída com > . . . 101
7.6 Redirecionamento de saída com >> . . . 101
7.7 Redirecionamento de entrada com < . . . 102
7.8 Here document com << . . . 102
7.9 Redirecionando erros . . . 102
7.10 Usando tee . . . 103

7.11 Laboratório guiado . . . 103
7.12 Resumo . . . 104
7.13 Perguntas de revisão . . . 104
7.14 Desafio opcional . . . 104

8 EXPRESSÕES REGULARES . . . 105
8.1 Objetivos do capítulo . . . 105
8.2 Pré-requisitos . . . 105
8.3 Conceito . . . 105
8.4 Arquivo de laboratório . . . 106
8.5 Busca literal . . . 106
8.6 Metacaractere ponto . . . 106
8.7 Listas de caracteres . . . 107
8.8 Lista negada . . . 107
8.9 Âncoras de início e fim . . . 108
8.10 Asterisco . . . 108
8.11 Expressões regulares estendidas . . . 108
8.12 Escapando metacaracteres . . . 109
8.13 Cuidados com aspas . . . 109
8.14 Laboratório guiado . . . 109
8.15 Resumo . . . 110
8.16 Perguntas de revisão . . . 110
8.17 Desafio opcional . . . 110

9 REDES NO LINUX . . . 111
9.1 Objetivos . . . 111
9.2 Pré-requisitos . . . 111
9.3 Conceitos fundamentais . . . 111
9.4 O comando ip . . . 112
9.5 Interfaces de rede . . . 113
9.5.1 Listando interfaces . . . 113
9.5.2 Ativando e desativando interfaces . . . 114
9.5.3 Alterando MTU . . . 114
9.6 Endereçamento IP . . . 114
9.6.1 Exibindo endereços . . . 114
9.6.2 Atribuindo endereço IP temporário . . . 115
9.6.3 Removendo endereço IP temporário . . . 115
9.6.4 Broadcast . . . 116
9.7 Roteamento . . . 116
9.7.1 Consultando rotas . . . 116
9.7.2 Acrescentando rota temporária . . . 117

9.7.3 Removendo rota temporária . . . 117
9.8 ARP e vizinhança . . . 117
9.8.1 Mostrando a tabela de vizinhança . . . 117
9.8.2 Removendo uma entrada de vizinhança . . . 118
9.9 Configuração persistente . . . 118
9.9.1 Debian e Ubuntu . . . 118
9.9.2 Rocky Linux, Red Hat e derivados . . . 119
9.10 Sequência de diagnóstico . . . 119
9.11 Comandos legados e equivalentes atuais . . . 120
9.12 Laboratório . . . 120
9.12.1 Prática 01: configuração básica de uma interface . . . 122
9.12.2 Prática 02: redes diretamente conectadas . . . 122
9.12.3 Prática 03: VLANs em laboratório . . . 123
9.12.4 Prática 04: roteamento entre redes . . . 125
9.13 Resumo . . . 125
9.14 Perguntas de revisão . . . 125
9.15 Desafio opcional . . . 126

10 NÍVEIS DE EXECUÇÃO . . . 127
10.1 systemd e systemctl . . . 128
10.1.1 Verificar o destino padrão . . . 128
10.1.2 Listar *targets* ativos . . . 128
10.1.3 Alterar o destino padrão . . . 128
10.1.4 Alterar o estado atual . . . 129
10.1.5 Controlar serviços . . . 129
10.1.6 Diagnóstico com journalctl . . . 130
10.2 Desligamento . . . 130
10.2.1 Shutdown . . . 130
10.2.2 Halt . . . 130
10.2.3 Power off . . . 131
10.2.4 Reboot . . . 131

11 GERENCIADORES DE PACOTES . . . 132
11.1 Objetivos . . . 132
11.2 Pré-requisitos . . . 132
11.3 Famílias de distribuições e formatos de pacote . . . 133
11.4 Conceitos importantes . . . 133
11.5 Configurações básicas . . . 134
11.5.1 APT . . . 134
11.5.2 DNF e YUM . . . 134
11.6 Atualizando a lista de pacotes . . . 135

11.7 Procurando pacotes e consultando informações . . . 135
11.8 Instalando pacotes . . . 136
11.9 Removendo pacotes . . . 136
11.10 Atualizando pacotes instalados . . . 136
11.11 Atualização pontual e atualização ampla . . . 137
11.12 Limpando cache e pacotes não utilizados . . . 137
11.13 Cuidados com repositórios . . . 138
11.14 Manuseando pacotes Debian com dpkg . . . 138
11.14.1 Instalando um arquivo .deb local . . . 139
11.14.2 Removendo pacote . . . 139
11.14.3 Consultando arquivos de um pacote instalado . . . 139
11.14.4 Descobrindo qual pacote instalou um arquivo . . . 139
11.14.5 Exibindo informações de um pacote instalado . . . 139
11.14.6 Listando pacotes instalados . . . 139
11.14.7 Listando arquivos dentro de um pacote .deb . . . 139
11.15 Manuseando pacotes Red Hat com rpm . . . 140
11.15.1 Instalando um arquivo .rpm local . . . 140
11.15.2 Atualizando pacote local . . . 140
11.15.3 Removendo pacote . . . 140
11.15.4 Consultando arquivos de um pacote instalado . . . 140
11.15.5 Descobrindo qual pacote instalou um arquivo . . . 140
11.15.6 Listando pacotes instalados . . . 140
11.15.7 Exibindo informações de um pacote . . . 140
11.16 Diagnóstico básico . . . 141
11.17 Laboratório guiado . . . 141
11.17.1 Objetivo do laboratório . . . 141
11.17.2 Ambiente . . . 142
11.17.3 Etapas em Debian ou Ubuntu . . . 142
11.17.4 Equivalência conceitual em Rocky Linux . . . 143
11.18 Exercícios de revisão . . . 143
11.19 Resumo . . . 143

12 LOGS DO SISTEMA . . . 144
12.1 Conceito de log . . . 145
12.2 Onde os logs ficam . . . 145
12.3 Logs clássicos e o syslog . . . 146
12.4 systemd-journald e journalctl . . . 146
12.4.1 Consultar logs de um serviço . . . 146
12.4.2 Filtrar por período . . . 147
12.4.3 Consultar a inicialização atual . . . 147

12.4.4 Acompanhar eventos em tempo real . . . 147
12.5 Lendo arquivos de log . . . 148
12.6 Tipos comuns de eventos . . . 148
12.6.1 Sistema . . . 148
12.6.2 Autenticação . . . 148
12.6.3 Kernel . . . 149
12.6.4 Serviços . . . 149
12.7 Rotação de logs . . . 149
12.8 Sequência básica de diagnóstico . . . 149
12.8.1 Serviço que não inicia . . . 150
12.8.2 Falha de login . . . 150
12.8.3 Disco cheio . . . 150
12.8.4 Erro de permissão . . . 151
12.9 Laboratório guiado . . . 151
12.10 Resumo . . . 152
12.11 Perguntas de revisão . . . 152
12.12 Desafio opcional . . . 152

13 AGENDAMENTO DE TAREFAS COM CRON . . . 154
13.1 Conceito de agendamento . . . 154
13.2 O serviço cron . . . 155
13.3 Crontab de usuário . . . 155
13.4 Sintaxe do cron . . . 156
13.5 Exemplos de agendamento . . . 157
13.6 Atalhos especiais . . . 157
13.7 Crontab do sistema . . . 158
13.8 Diretórios /etc/cron.* . . . 158
13.9 Ambiente reduzido do cron . . . 159
13.10 Saída, erro e logs . . . 159
13.11 Cuidados administrativos . . . 160
13.12 Diagnóstico de tarefas que não executam . . . 160
13.13 Systemd timers . . . 161
13.14 Laboratório guiado: backup simples com cron . . . 162
13.14.1 1. Preparar diretórios . . . 162
13.14.2 2. Criar o script de backup . . . 162
13.14.3 3. Testar manualmente . . . 162
13.14.4 4. Agendar execução . . . 162
13.14.5 5. Verificar execução . . . 163
13.14.6 6. Encerrar o teste . . . 163
13.15 Resumo . . . 164

13.16 Perguntas de revisão . . . 164
13.17 Desafio opcional . . . 164

14 FERRAMENTAS DE REDES . . . 165
14.1 Objetivos . . . 165
14.2 Pré-requisitos . . . 165
14.3 Conceito: ferramenta não é diagnóstico . . . 165
14.4 Sequência prática de diagnóstico . . . 166
14.5 Ferramentas essenciais . . . 166
14.5.1 ip . . . 166
14.5.2 ss . . . 167
14.5.3 ping . . . 167
14.5.4 traceroute e tracepath . . . 167
14.5.5 dig . . . 168
14.6 Netcat e tcpdump no diagnóstico . . . 168
14.7 Netcat . . . 168
14.7.1 Descrição . . . 168
14.7.2 Cuidados de uso . . . 169
14.7.3 Variantes do netcat . . . 169
14.7.4 Testando se uma porta TCP responde . . . 169
14.7.5 Conectando manualmente a um serviço TCP . . . 170
14.7.6 Criando um serviço simples de laboratório . . . 170
14.7.7 Testando UDP em laboratório . . . 170
14.7.8 Transferência simples de arquivo em laboratório . . . 171
14.7.9 Diagnóstico com netcat . . . 171
14.7.10 Laboratório guiado . . . 171
14.7.11 Resumo . . . 171
14.8 tcpdump . . . 172
14.8.1 Descrição . . . 172
14.8.2 Identificando interfaces . . . 172
14.8.3 Captura básica . . . 172
14.8.4 Limitando a quantidade de pacotes . . . 173
14.8.5 Filtros essenciais . . . 173
14.8.6 Capturando tráfego DNS . . . 174
14.8.7 Capturando tráfego ICMP . . . 174
14.8.8 Lendo uma linha de saída . . . 174
14.8.9 Salvando e lendo capturas . . . 175
14.8.10 Laboratório guiado . . . 175
14.8.11 Diagnóstico com tcpdump . . . 175
14.8.12 Resumo . . . 175

14.9 Laboratório guiado: diagnóstico em rede local . . . 176
14.9.1 Cenário . . . 176
14.9.2 Parte 1: identificar a configuração da máquina . . . 176
14.9.3 Parte 2: verificar portas locais . . . 176
14.9.4 Parte 3: testar conectividade e DNS . . . 176
14.9.5 Parte 4: testar uma porta com netcat . . . 176
14.9.6 Parte 5: observar pacotes com tcpdump . . . 177
14.10 Resumo . . . 177
14.11 Perguntas de revisão . . . 177
14.12 Desafio opcional . . . 177

15 INTRODUÇÃO AO MODELO CLIENTE-SERVIDOR . . . 179
15.1 Objetivos . . . 179
15.2 Pré-requisitos . . . 179
15.3 Conceito de cliente e servidor . . . 180
15.4 Requisição, resposta e estado da comunicação . . . 180
15.5 Portas, protocolos e sockets . . . 181
15.6 Relação com OSI e TCP/IP . . . 181
15.7 TCP e UDP no contexto cliente-servidor . . . 183
15.8 Observando serviços em uma máquina Linux . . . 183
15.9 Laboratório guiado: comunicação local com netcat . . . 184
15.9.1 Preparação . . . 184
15.9.2 Teste do cliente . . . 184
15.9.3 Observação com ss . . . 185
15.9.4 Observação com tcpdump . . . 185
15.10 Diagnóstico básico . . . 185
15.11 Relação com DNS e SSH . . . 186
15.12 Resumo . . . 186
15.13 Perguntas de revisão . . . 186
15.14 Desafio opcional . . . 186

16 SERVIDORES DE SERVIÇOS . . . 187
16.1 DNS . . . 187
16.1.1 Objetivos . . . 187
16.1.2 Pré-requisitos . . . 187
16.1.3 Histórico . . . 188
16.1.4 Espaço de nomes do DNS . . . 189
16.1.5 DNS nas Organizações . . . 191
16.1.6 Zonas e Domínios . . . 192
16.1.7 Configurações de Servidores de DNS . . . 193
16.1.8 Tipos de Servidores de DNS . . . 193

16.1.8.1 Root Name Servers . . . 193
16.1.8.2 TLD – Top Level Domain Servers . . . 194
16.1.8.3 Servidor Interativo (não-Recursivo) e Recursivo . . . 194
16.1.8.4 Servidor Caching . . . 198
16.1.8.5 DNS Autoritativo e não-Autoritativo . . . 198
16.1.9 O Sistema BIND . . . 198
16.1.9.1 Biblioteca resolver . . . 199
16.1.9.2 NSS - Name Service Switch . . . 200
16.1.9.3 resolv.conf . . . 201
16.1.9.4 Consultas e diagnóstico básico . . . 201
16.1.10 Configuração do Servidor . . . 202
16.1.10.1 Base de Dados do DNS . . . 205
16.1.11 Laboratorio com Debian . . . 207
16.2 SSH . . . 211
16.2.1 Objetivos . . . 211
16.2.2 Pré-requisitos . . . 211
16.2.3 Cliente, servidor e porta 22 . . . 212
16.2.4 Instalação do servidor SSH . . . 213
16.2.5 Primeiro acesso e *known_hosts* . . . 213
16.2.6 Autenticação por senha . . . 214
16.2.7 Chaves SSH . . . 214
16.2.8 Permissões do diretório *.ssh* . . . 215
16.2.9 Instalando a chave pública no servidor . . . 215
16.2.10 Agente SSH . . . 216
16.2.11 Copiando arquivos com *scp* . . . 217
16.2.12 Transferência interativa com *sftp* . . . 218
16.2.13 Encaminhamento de portas . . . 218
16.2.14 Acesso gráfico com X11 . . . 219
16.2.15 Boas práticas básicas . . . 219
16.2.16 Diagnóstico de problemas comuns . . . 220
16.2.16.1 Conexão recusada . . . 220
16.2.16.2 Timeout . . . 221
16.2.16.3 Permissão negada . . . 221
16.2.16.4 Chave de host alterada . . . 222
16.2.17 Laboratório guiado . . . 222
16.2.17.1 Cenário . . . 222
16.2.17.2 Etapa 1: preparar o servidor . . . 223
16.2.17.3 Etapa 2: testar conexão por senha . . . 223
16.2.17.4 Etapa 3: gerar chave no cliente . . . 223
16.2.17.5 Etapa 4: instalar a chave pública . . . 224

16.2.17.6 Etapa 5: copiar arquivo 224
16.2.17.7 Etapa 6: observar logs 224
16.2.18 Exercícios de revisão 224
16.2.19 Resumo 225

REFERÊNCIAS 226

Introdução

O curso **Linux - Módulo 1** apresenta os fundamentos necessários para a operação inicial de servidores GNU/Linux em ambientes reais. O foco desta etapa é formar a base técnica de profissionais que estão iniciando na administração de sistemas, com atenção especial à navegação no sistema, manipulação de arquivos, administração de usuários, permissões, pacotes, discos, redes e serviços básicos.

Ao longo do material, os conceitos são apresentados antes das ferramentas. Essa abordagem permite compreender não apenas qual comando executar, mas também o que o sistema está fazendo e quais cuidados devem ser observados em uma rotina de administração.

Objetivos do módulo

Ao final deste módulo, o aluno deverá ser capaz de:

- navegar pela estrutura de diretórios do Linux;
- manipular arquivos, diretórios, processos e permissões;
- compreender a organização básica do sistema operacional;
- instalar, atualizar e remover pacotes;
- interpretar informações básicas de rede;
- utilizar ferramentas iniciais de diagnóstico;
- executar tarefas rotineiras de um administrador Linux júnior.

Características gerais do Linux

- Proteção entre processos executados na memória RAM
- Suporte a mais de 63 terminais virtuais (consoles)
- Modularização - O Linux somente carrega para a memória o que é usado durante o processamento, liberando totalmente a memória assim que o programa/dispositivo é finalizado
- Devido a modularização, os drivers dos periféricos e recursos do sistema podem ser carregados e removidos completamente da memória RAM a qualquer momento. Os drivers (módulos) ocupam pouco espaço quando carregados na memória RAM.

- Suporte nativo a rede e tecnologias avançadas como: balanceamento de carga, ips alias, failover, vlans, bridge, trunking, OSPF, BGP.
- Não há a necessidade de se reiniciar o sistema após a modificar a configuração de qualquer periférico ou parâmetros de rede. Somente é necessário reiniciar o sistema no caso de uma instalação interna de um novo periférico, falha em algum hardware (queima do processador, placa mãe, etc.).
- Não precisa de um processador potente para funcionar. O sistema roda bem em computadores simples e pouca memória RAM (sem rodar o sistema gráfico X).
- Suporte nativo a múltiplas CPUs, tanto em 32 ou 64 bits.
- Suporte nativo a dispositivos SATA, PATA, Fiber Channel
- Suporte nativo a virtualização.
- O crescimento e novas versões do sistema não provocam lentidão, pelo contrário.
- O GNU/Linux é distribuído livremente e licenciado de acordo com os termos da GPL.
- Acessa diversos formatos de partição, como DOS, Windows, Novell, OS/2, NTFS, SunOS, Amiga, Atari, Mac, etc.
- O Linux possui um modelo de segurança robusto, baseado em separação de privilégios, permissões, isolamento entre processos e uso controlado da conta administrativa. Isso reduz o impacto de códigos maliciosos, mas não torna o sistema imune a vírus, worms, trojans, rootkits, falhas de configuração ou exploração de vulnerabilidades. Sistemas desatualizados, serviços expostos sem controle, senhas fracas e uso indevido de contas administrativas, como o usuário *root* no Linux, continuam sendo riscos reais para qualquer sistema operacional.
- O GNU/Linux tem suporte nativo a redes TCP/IP e não depende de uma camada intermediária.
- São feitos extensivos testes em uma comunidade aberta antes do lançamento de cada versão para atingir um alto grau de confiabilidade.

Símbolo	Base 10	Base 2	Nome
K	10^3	2^{10}	Quilo
M	10^6	2^{20}	Mega
G	10^9	2^{30}	Giga
T	10^{12}	2^{40}	Tera
P	10^{15}	2^{50}	Peta
E	10^{18}	2^{60}	Eta
Z	10^{21}	2^{70}	Zetta
Y	10^{24}	2^{80}	Yotta

Tabela 1 – Notação cientifica e prefixos do Sistema Internacional de Unidades

1 Discos e Partições

Este capítulo apresenta os fundamentos de armazenamento local no GNU/Linux. O objetivo é entender a cadeia completa entre disco, partição, sistema de arquivos, ponto de montagem e uso pelo sistema operacional. Também serão apresentados comandos de diagnóstico e um laboratório seguro com disco virtual, evitando alterações em discos reais da máquina do aluno.

1.1 Objetivos

Ao final deste capítulo, o aluno deverá ser capaz de:

- diferenciar dispositivo de bloco, partição, sistema de arquivos e ponto de montagem;
- listar discos e partições com ***lsblk***, ***blkid*** e ***fdisk***;
- criar partições em um disco de laboratório;
- formatar partições com ***mkfs***;
- montar e desmontar sistemas de arquivos;
- ativar e desativar uma área de swap;
- compreender a função do arquivo ***/etc/fstab***;
- reconhecer, em nível introdutório, quando o LVM pode ser útil.

1.2 Pré-requisitos

Para acompanhar o laboratório, o aluno deve conhecer os comandos básicos de navegação, criação de diretórios e leitura de arquivos. Também é necessário ter acesso administrativo com ***sudo*** em uma máquina de laboratório.

> Os comandos de particionamento e formatação podem destruir dados quando executados sobre o disco errado. Neste capítulo, os exemplos usam um arquivo de disco virtual conectado a um *loop device*. Antes de executar qualquer comando em ambiente real, confirme o dispositivo com ***lsblk***.

1.3 Conceitos fundamentais

Antes de usar as ferramentas, é importante separar quatro conceitos:

Dispositivo de bloco: é o recurso que representa um disco ou volume para o kernel, como ***/dev/sda***, ***/dev/vda***, ***/dev/nvme0n1*** ou ***/dev/loop0***.

Partição: é uma divisão lógica dentro de um disco. Exemplos comuns são ***/dev/sda1***, ***/dev/vda2*** e ***/dev/nvme0n1p1***.

Sistema de arquivos: é a estrutura criada dentro de uma partição para organizar arquivos e diretórios, como ext4, XFS ou VFAT.

Ponto de montagem: é o diretório onde o sistema de arquivos fica acessível, como ***/mnt/-dados***, ***/home*** ou ***/boot***.

Na prática, a sequência mais comum é:

```
disco → partição → sistema de arquivos → montagem → uso pelos programas
```

Um disco recém-adicionado ao sistema não aparece automaticamente como diretório. Ele precisa ser identificado, particionado quando necessário, formatado e montado.

1.4 Identificando discos e partições

O primeiro passo antes de alterar discos é observar o estado atual do sistema. O comando ***lsblk*** mostra dispositivos de bloco em formato de árvore:

```
$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sda 8:0 0 40G 0 disk
|-sda1 8:1 0 1G 0 part /boot
‘-sda2 8:2 0 39G 0 part /
```

Para incluir o tipo de sistema de arquivos e o UUID, use:

```
$ lsblk -f
$ sudo blkid
```

O ***blkid*** é especialmente útil quando será configurada montagem persistente no ***/etc/fstab***, pois o UUID tende a ser mais estável que nomes como ***/dev/sdb1***.

1.5 Tabelas de partição: MBR e GPT

A tabela de partição registra onde cada partição começa e termina. Historicamente, o padrão MBR foi muito usado em computadores com BIOS. Ele permite até quatro partições primárias, ou três primárias e uma estendida contendo partições lógicas.

O padrão GPT é mais atual e normalmente usado em sistemas com UEFI. Ele remove limitações importantes do MBR e é a escolha comum em servidores e computadores modernos. Para o Módulo 1, o mais importante é compreender que MBR e GPT são formas diferentes de organizar a tabela de partição, não sistemas de arquivos.

Figura 1 – Lista de formatos de partições reconhecidos pelo particionador fdisk.

1.6 Ferramentas de particionamento

As ferramentas mais comuns para particionamento em modo texto são ***fdisk***, ***cfdisk*** e ***parted***. O ***gparted*** oferece interface gráfica, mas em servidores é comum trabalhar em terminal.

Neste capítulo será usado o ***fdisk***, por ser simples para laboratórios introdutórios. Em discos grandes, ambientes com GPT ou automação, o ***parted*** também é muito utilizado.

1.7 Laboratório: criando um disco virtual

Para reduzir o risco operacional, o laboratório usa um arquivo como se fosse um disco. O arquivo será criado com ***dd*** e associado ao sistema com ***losetup***.

Crie um diretório de trabalho e um arquivo de 200 MiB:

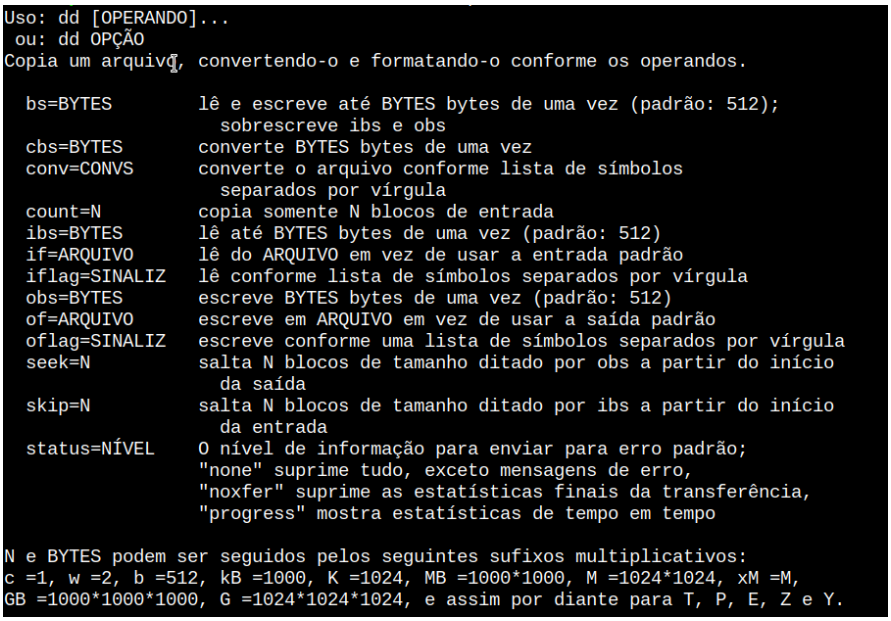


Figura 2 – Opções do comando ***dd***.

```
$ mkdir -p laboratorio_discos
$ cd laboratorio_discos
$ dd if=/dev/zero of=disco_teste.img bs=1M count=200 status=progress
200+0 registros de entrada
200+0 registros de saída
209715200 bytes (210 MB, 200 MiB) copiados
```

O ***/dev/zero*** é um arquivo especial que fornece bytes nulos. Ele é útil para criar arquivos preenchidos com zeros, como imagens de disco para laboratório.

Agora associe o arquivo a um dispositivo de loop. A opção ***–show*** informa qual dispositivo foi criado, e a opção ***–find*** escolhe automaticamente um loop livre:

```
$ sudo losetup –show –find disco_teste.img
/dev/loop17
```

O número do dispositivo pode ser diferente no seu laboratório. Se o comando retornar ***/dev/loop3***, use ***/dev/loop3*** nos próximos comandos. Não copie cegamente o número do exemplo.

Confira o dispositivo criado:

```
$ lsblk /dev/loop17
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
loop17 7:17 0 200M 0 loop
```

1.8 Particionando o disco virtual

Execute o ***fdisk*** sobre o dispositivo de loop:

```
$ sudo fdisk /dev/loop17
```

Dentro do ***fdisk***, crie a seguinte estrutura:

- partição 1: primária, 80 MiB, tipo Linux;
- partição 2: primária, 32 MiB, tipo Linux swap;
- partição 3: primária, espaço restante, tipo Linux.

Uma sequência típica de comandos internos do ***fdisk*** é:

```
n   p   1   ENTER   +80M
n   p   2   ENTER   +32M
t   2   82
n   p   3   ENTER   ENTER
p
w
```

Depois de gravar a tabela de partições, peça ao kernel para reler a tabela:

```
$ sudo partprobe /dev/loop17
```

Verifique o resultado:

```
$ lsblk /dev/loop17
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
loop17 7:17 0 200M 0 loop
|-loop17p1 259:0 0 80M 0 part
|-loop17p2 259:1 0 32M 0 part
`-loop17p3 259:2 0 86M 0 part
```

1.9 Sistemas de arquivos

Formatar uma partição significa criar nela uma estrutura que permita ao sistema operacional gravar, localizar e remover arquivos. Essa estrutura é o sistema de arquivos.

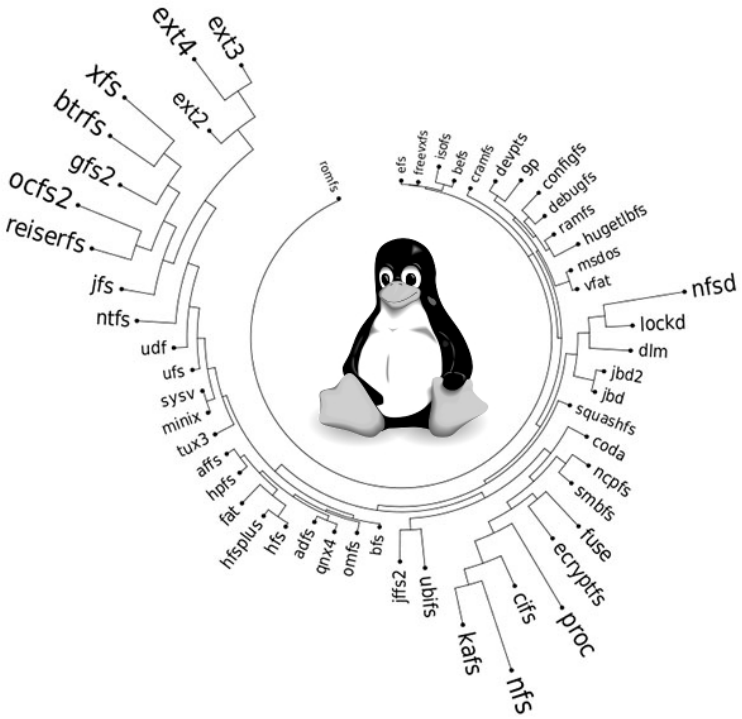


Figura 3 – Árvore filogênica de sistema de arquivos Linux (MUSăLOIU-E., 2009).

No GNU/Linux, os comandos de criação de sistemas de arquivos normalmente começam com ***mkfs***:

```
$ ls /sbin/mkfs.*
/sbin/mkfs.bfs    /sbin/mkfs.cramfs  /sbin/mkfs.ext2  /sbin/mkfs.ext4  /sbin/mkfs.minix  /sbin/mkfs.ntfs      /sbin/mkfs.vfat
/sbin/mkfs.btrfs  /sbin/mkfs.exfat   /sbin/mkfs.ext3  /sbin/mkfs.fat   /sbin/mkfs.msdos  /sbin/mkfs.reiserfs  /sbin/mkfs.xfs
$
```

1.9.1 EXT2, EXT3 e EXT4

O EXT2 foi um dos sistemas de arquivos tradicionais do Linux. Ele não possui *journaling*, característica que reduz sua utilidade em servidores modernos, mas ainda ajuda a compreender a evolução histórica dos sistemas de arquivos.

O EXT3 adicionou *journaling* ao EXT2. O journal registra metadados das operações, ajudando o sistema a recuperar consistência após uma queda de energia ou desligamento incorreto.

Journaling

O journaling registra operações em uma área especial antes da confirmação final no sistema de arquivos. Após uma falha, o sistema verifica esse registro e reduz o tempo necessário para voltar a um estado consistente (JONES, 2008).

O EXT4 é a evolução mais usada da família EXT em distribuições Debian, Ubuntu e vários ambientes Linux. Ele suporta journaling, arquivos maiores, melhor alocação de blocos e melhor desempenho geral que EXT3 na maioria dos cenários.

1.9.2 XFS

O XFS é um sistema de arquivos de 64 bits criado originalmente pela Silicon Graphics. Ele é muito usado em ambientes de servidor, especialmente em distribuições da família Red Hat, como Rocky Linux 9. O XFS trabalha bem com arquivos grandes e alto volume de entrada e saída (**??**).

Uma diferença operacional importante é que o XFS pode crescer online, mas não é reduzido de forma simples como ocorre em alguns cenários com EXT4. Esse detalhe deve ser considerado antes de planejar partições.

1.9.3 VFAT

O VFAT é usado principalmente para compatibilidade com outros sistemas operacionais, pendrives e partições de troca de arquivos. Ele não possui os mesmos recursos de permissões Unix disponíveis em EXT4 ou XFS.

1.9.4 ReiserFS

O ReiserFS teve importância histórica como alternativa com journaling, mas hoje não é uma escolha recomendada para novos servidores. Neste curso, ele deve ser entendido como tecnologia legada.

1.9.5 Swap

Swap é uma área usada pelo kernel para mover páginas de memória que não estão sendo usadas ativamente. Ela não substitui memória RAM: disco é muito mais lento que RAM. Mesmo assim, swap pode ajudar o sistema a evitar encerramentos abruptos de processos quando há pressão de memória.

A swap pode existir em uma partição ou em um arquivo. Em instalações modernas, ambas as opções são comuns. O mais importante é monitorar o uso: se o servidor usa swap constantemente, pode haver falta de memória ou aplicação consumindo recursos em excesso.

1.10 Formatando as partições

Neste laboratório, a primeira partição será formatada como EXT4, a segunda será preparada como swap e a terceira será formatada como XFS.

Confirme o dispositivo antes de formatar. O comando ***mkfs*** apaga a estrutura anterior da partição informada.

```
$ sudo mkfs.ext4 /dev/loop17p1
$ sudo mkswap /dev/loop17p2
$ sudo mkfs.xfs -f /dev/loop17p3
```

Se o pacote de ferramentas XFS não estiver instalado, instale-o conforme a distribuição:

```
$ sudo apt install xfsprogs
$ sudo dnf install xfsprogs
```

Verifique os sistemas de arquivos criados:

```
$ lsblk -f /dev/loop17
$ sudo blkid /dev/loop17p1 /dev/loop17p2 /dev/loop17p3
```

1.11 Montando sistemas de arquivos

Para acessar os arquivos de uma partição, é necessário montá-la em um diretório. Esse diretório é chamado ponto de montagem.

```
mount -t sistema_de_arquivos dispositivo ponto_de_montagem
```

Crie os pontos de montagem e monte as partições:

```
$ mkdir -p acesso_ext4 acesso_xfs
$ sudo mount -t ext4 /dev/loop17p1 acesso_ext4
$ sudo mount -t xfs /dev/loop17p3 acesso_xfs
```

Confirme a montagem:

```
$ findmnt | grep loop17
$ df -hT | grep loop17
```

Teste a gravação:

```
$ sudo touch acesso_ext4/arquivo_ext4.txt
$ sudo touch acesso_xfs/arquivo_xfs.txt
$ ls -l acesso_ext4 acesso_xfs
```

Para desmontar:

```
$ sudo umount acesso_ext4
$ sudo umount acesso_xfs
```

Se o ***umount*** informar que o alvo está ocupado, verifique se há algum terminal dentro do diretório montado ou algum processo usando arquivos nele.

1.12 Ativando e verificando swap

Ative a swap criada no laboratório:

```
$ sudo swapon /dev/loop17p2
$ cat /proc/swaps
Filename Type Size Used Priority
/dev/loop17p2 partition 32764 0 -2
```

Também é possível verificar com:

```
$ free -h
$ swapon –show
```

```
$ free -h
               total        usada       livre    compart.  buff/cache  disponível
Mem.:           15Gi        13Gi       185Mi       1,2Gi       1,9Gi       633Mi
Swap:           31Gi       760Mi        31Gi
```

Desative a swap ao final do teste:

```
$ sudo swapoff /dev/loop17p2
```

1.13 Montagem persistente com /etc/fstab

Montagens feitas manualmente com ***mount*** desaparecem após a reinicialização. Para montar sistemas de arquivos automaticamente, o Linux usa o arquivo ***/etc/fstab***.

Uma linha típica contém:

```
UUID=valor-do-uuid    /mnt/dados    ext4    defaults    0    2
```

Os campos principais são:

- dispositivo ou UUID;

- ponto de montagem;
- tipo de sistema de arquivos;
- opções de montagem;
- controle de dump;
- ordem de verificação pelo fsck.

Em servidores reais, prefira usar UUID:

```
$ sudo blkid /dev/sdb1
/dev/sdb1: UUID="1111-2222"TYPE="ext4"
```

Antes de reiniciar um servidor depois de alterar o ***/etc/fstab***, teste a configuração:

```
$ sudo mount -a
```

Um erro no ***/etc/fstab*** pode impedir a montagem correta de sistemas de arquivos durante a inicialização. Em ambiente de produção, mantenha acesso de console ou procedimento de recuperação antes de alterar esse arquivo.

1.14 Visão inicial de LVM

O LVM (*Logical Volume Manager*) cria uma camada lógica entre discos e sistemas de arquivos. Ele permite agrupar dispositivos físicos, criar volumes lógicos e redimensionar volumes com mais flexibilidade que partições tradicionais.

No LVM, os principais elementos são:

PV: *Physical Volume*, normalmente uma partição ou disco preparado para LVM.

VG: *Volume Group*, grupo formado por um ou mais PVs.

LV: *Logical Volume*, volume lógico onde normalmente se cria o sistema de arquivos.

Para o Módulo 1, o aluno deve reconhecer o LVM e entender por que ele existe. A criação e administração detalhada de LVM ficam para módulos posteriores.

1.15 Diagnóstico básico

Problemas de disco e montagem costumam ser investigados com comandos simples:

```
$ lsblk -f
$ sudo blkid
$ findmnt
$ df -hT
$ dmesg | tail
$ journalctl -k
```

Situações comuns:

- o disco aparece em ***lsblk***, mas não tem sistema de arquivos;
- a partição foi formatada, mas não está montada;
- o ponto de montagem não existe;
- o tipo informado no ***mount*** está incorreto;
- há erro no ***/etc/fstab***;
- o dispositivo está ocupado e não pode ser desmontado.

1.16 Limpeza do laboratório

Ao finalizar, desmonte os sistemas de arquivos, desative a swap e solte o *loop device*:

```
$ sudo umount acesso_ext4
$ sudo umount acesso_xfs
$ sudo swapoff /dev/loop17p2
$ sudo losetup -d /dev/loop17
$ cd ..
$ rm -rf laboratorio_discos
```

1.17 Resumo

Neste capítulo, o armazenamento local foi apresentado como uma sequência de camadas. O disco fornece blocos, a partição delimita uma área, o sistema de arquivos organiza dados e o ponto de montagem torna esses dados acessíveis na árvore de diretórios do Linux. Também foram vistos comandos atuais para identificar, particionar, formatar, montar, diagnosticar e desmontar dispositivos.

1.18 Perguntas de revisão

1. Qual é a diferença entre disco, partição, sistema de arquivos e ponto de montagem?
2. Por que o uso de UUID é recomendado no ***/etc/fstab***?
3. O que acontece quando uma partição é formatada com ***mkfs***?
4. Qual comando mostra a árvore de dispositivos de bloco?
5. Qual é a diferença entre montar uma partição e ativar uma área de swap?
6. Por que é perigoso executar comandos de particionamento sem conferir o dispositivo?

1.19 Desafio opcional

Crie outro disco virtual, agora com tabela GPT, usando ***parted***. Crie uma partição EXT4, monte-a em ***/mnt/lab-gpt***, grave um arquivo de teste, desmonte a partição e remova o *loop device*. Registre os comandos usados e explique a função de cada etapa.

2 Backup Básico

Em administração de sistemas, backup não é apenas copiar arquivos. Backup é um processo planejado para preservar dados, permitir recuperação e reduzir impacto quando ocorre falha humana, problema de disco, corrupção de arquivos, remoção acidental, atualização malsucedida ou incidente de segurança.

Um administrador Linux Júnior não precisa começar usando ferramentas corporativas complexas, mas precisa entender o raciocínio operacional: o que deve ser protegido, com que frequência, onde será armazenado, por quanto tempo será mantido e como será restaurado quando necessário. Um backup que nunca foi testado deve ser tratado como uma promessa, não como uma garantia.

Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- diferenciar cópia simples, backup, snapshot e restauração;
- identificar dados importantes para uma rotina básica de backup;
- criar arquivos compactados com `tar`;
- sincronizar diretórios com `rsync`;
- compreender noções iniciais de retenção;
- validar integridade e conteúdo de um backup;
- executar uma restauração em diretório separado;
- reconhecer cuidados operacionais antes de usar opções destrutivas.

Pré-requisitos

Para acompanhar este capítulo, é recomendável conhecer comandos básicos do shell, estrutura de diretórios, permissões, redirecionamento, processos, agendamento com *cron* e noções iniciais de discos e sistemas de arquivos. Os exemplos devem ser executados em máquina de laboratório, preferencialmente Debian 12, Ubuntu 24.04 LTS ou Rocky Linux 9.

2.1 Backup como processo

O erro mais comum ao iniciar o estudo de backup é começar pela ferramenta. Antes do comando, o administrador precisa responder perguntas simples:

- quais arquivos ou diretórios precisam ser protegidos?
- quais dados podem ser recriados e quais não podem?
- qual é a frequência aceitável de perda?
- por quanto tempo as cópias devem ser mantidas?
- onde o backup será armazenado?
- quem tem permissão para acessar esse backup?
- como será feita a restauração?
- quando foi o último teste de restauração?

Essas perguntas transformam backup em uma rotina administrativa, e não apenas em um comando executado quando sobra tempo. Em servidores reais, a ausência dessas definições costuma aparecer somente no pior momento: quando a restauração é necessária.

2.2 Cópia, backup, snapshot e restauração

Os termos a seguir são relacionados, mas não significam a mesma coisa:

Cópia: duplicação simples de um arquivo ou diretório. Pode ser útil, mas normalmente não possui histórico, política de retenção nem validação.

Backup: cópia planejada, identificada, armazenada em local adequado e mantida por um período definido. Deve permitir restauração.

Snapshot: registro de estado de um volume, sistema de arquivos, máquina virtual ou armazenamento em determinado momento. Ajuda em alguns cenários, mas não substitui backup externo e testado.

Restauração: processo de recuperar dados a partir do backup. A restauração deve ser testada antes de uma emergência.

Um snapshot no mesmo disco pode ajudar a voltar rapidamente uma alteração, mas não protege contra perda do próprio disco. Uma cópia em outro diretório da mesma partição pode proteger contra edição acidental, mas não contra falha física. Por isso, o local do backup é parte central do planejamento.

2.3 O que proteger

Nem tudo em um sistema Linux tem o mesmo valor. Alguns diretórios guardam dados críticos; outros contêm arquivos temporários, caches ou conteúdo que pode ser reinstalado.

Exemplos de dados normalmente importantes:

- arquivos de usuários em `/home`;
- configurações em `/etc`;
- dados de aplicações em `/var/lib`;
- sites e aplicações locais em `/var/www` ou diretórios equivalentes;
- scripts administrativos em `/usr/local/bin` e `/usr/local/sbin`;
- bancos de dados exportados corretamente pela ferramenta apropriada.

Exemplos de dados que normalmente não devem ser copiados em uma rotina simples:

- `/proc`, `/sys`, `/run` e `/dev`, pois são interfaces dinâmicas do sistema;
- caches que podem ser recriados;
- diretórios temporários como `/tmp`;
- montagens remotas ou discos externos sem intenção explícita.

Antes de montar uma rotina de backup em servidor real, confirme se há bancos de dados, aplicações em execução ou arquivos abertos que exigem procedimento próprio de exportação. Copiar diretamente arquivos internos de um banco de dados em uso pode gerar backup inconsistente.

2.4 Local de armazenamento

Guardar o backup no mesmo disco dos dados originais é melhor do que não ter nada, mas não resolve a maioria dos incidentes graves. Se o disco falhar, os dados originais e a cópia podem ser perdidos juntos.

Locais possíveis em ambientes simples:

- outro disco local;
- volume montado especificamente para backup;
- servidor de arquivos interno;
- armazenamento removível usado com controle;

- destino remoto acessado por SSH, quando permitido pela política do ambiente.

Para o Módulo 1, o foco será backup local em diretórios de laboratório. Em ambiente real, o aluno deve lembrar que backup precisa sair do ponto único de falha sempre que possível.

2.5 Retenção

Retenção é a regra que define por quanto tempo cada backup será mantido. Sem retenção, duas situações ruins aparecem rapidamente: ou o disco enche com backups antigos, ou backups importantes são apagados cedo demais.

Exemplo simples de política:

- manter backups diários dos últimos 7 dias;
- manter backups semanais das últimas 4 semanas;
- manter backups mensais dos últimos 3 meses.

Essa política é apenas um exemplo. A retenção correta depende do tipo de dado, da criticidade, da capacidade de armazenamento e dos requisitos do ambiente.

2.6 Backup com tar

O `tar` cria arquivos que agrupam diretórios e arquivos. Ele é muito usado em Linux para empacotar conteúdo e, opcionalmente, comprimir o resultado.

Criando um arquivo `tar.gz` de um diretório:

```
$ tar -czf backup-home-aluno.tar.gz /home/aluno
```

Opções usadas:

- `-c`: cria um novo arquivo;
- `-z`: usa compressão gzip;
- `-f`: informa o nome do arquivo de saída.

Para listar o conteúdo sem extrair:

```
$ tar -tzf backup-home-aluno.tar.gz
```

Para restaurar em um diretório específico:

```
$ mkdir -p restauracao
$ tar -xzf backup-home-aluno.tar.gz -C restauracao
```

O teste de listagem com `tar -tzf` ajuda a verificar se o arquivo pode ser lido e quais caminhos foram gravados. Ele não substitui um teste real de restauração, mas já evita algumas surpresas.

2.7 Caminhos absolutos e relativos no tar

Ao criar um backup com caminho absoluto, o `tar` pode registrar a árvore com o caminho original. Dependendo da forma de extração, isso pode confundir ou até sobrescrever arquivos em locais indesejados.

Uma prática mais segura em laboratório é entrar no diretório pai e criar o backup usando caminho relativo:

```
$ cd /home
$ tar -czf /tmp/backup-aluno.tar.gz aluno
```

Na restauração:

```
$ mkdir -p /tmp/restauracao
$ tar -xzf /tmp/backup-aluno.tar.gz -C /tmp/restauracao
```

Assim, o conteúdo será restaurado abaixo de `/tmp/restauracao/aluno`, sem escrever diretamente sobre `/home/aluno`.

2.8 Backup com rsync

O `rsync` sincroniza arquivos entre origem e destino. Ele compara o que existe nos dois lados e copia apenas o necessário. Por isso, é muito usado para rotinas recorrentes.

Sincronização básica:

```
$ rsync -av /home/aluno/documentos/ /backup/documentos/
```

Opções usadas:

- `-a`: modo arquivo, preservando permissões, datas e estrutura;
- `-v`: saída detalhada da operação.

A barra final na origem é importante:

- `/home/aluno/documentos/` copia o conteúdo do diretório;
- `/home/aluno/documentos` copia o diretório `documentos` como item dentro do destino.

2.8.1 Teste sem copiar

Antes de executar uma sincronização sensível, use `-dry-run`. Essa opção mostra o que seria feito, sem alterar o destino:

```
$ rsync -av --dry-run /home/aluno/documentos/ /backup/documentos/
```

2.8.2 Cuidado com –delete

A opção `-delete` remove do destino arquivos que não existem mais na origem. Ela é útil para espelhamento, mas perigosa quando usada sem conferência.

```
$ rsync -av --delete /home/aluno/documentos/ /backup/documentos/
```

Nunca use `rsync -delete` sem confirmar origem e destino. Se os caminhos forem invertidos, o comando pode apagar dados válidos no local errado. Em ambiente de produção, teste primeiro com `-dry-run`.

2.9 Integridade e verificação

Backup precisa ser verificável. Algumas verificações simples já melhoram muito a confiabilidade da rotina:

- conferir se o arquivo foi criado;
- conferir tamanho e data;
- listar conteúdo do arquivo `tar`;
- restaurar em diretório separado;
- comparar arquivos restaurados com os originais;
- registrar saída e erro da rotina.

Exemplos:

```
$ ls -lh backups/
$ tar -tzf backups/backup-dados.tar.gz
$ sha256sum backups/backup-dados.tar.gz
```

O `sha256sum` gera uma soma de verificação do arquivo. Ela ajuda a identificar alteração ou corrupção posterior do arquivo de backup, desde que o valor gerado seja guardado em local confiável.

2.10 Restauração

Restauração é a parte mais importante do backup. O objetivo final não é ter um arquivo `.tar.gz` bonito, mas recuperar dados úteis quando necessário.

Uma restauração segura deve começar em diretório separado:

```
$ mkdir -p teste-restauracao
$ tar -xzf backup-dados.tar.gz -C teste-restauracao
$ find teste-restauracao -type f
```

Depois de conferir o conteúdo, o administrador decide se os arquivos restaurados devem substituir dados em produção, ser mesclados manualmente ou ficar disponíveis para consulta.

> Evite restaurar diretamente sobre o diretório original sem uma conferência prévia. Uma restauração mal direcionada pode sobrescrever arquivos atuais e transformar um incidente pequeno em perda maior.

2.11 Automação e logs

Depois que a rotina funciona manualmente, ela pode ser automatizada com *cron*. A automação, porém, deve registrar saída e erro:

```
0 2 * * * /usr/local/bin/backup-dados.sh \
    >> /var/log/backup-dados.log 2>&1
```

Antes de agendar:

1. execute o script manualmente;
2. confirme origem e destino;
3. confira permissões;
4. teste restauração;
5. verifique se há espaço no destino;
6. registre saída e erro;
7. documente a política de retenção.

Esse encadeamento conecta backup, logs, cron e discos. Um backup agendado que não registra erro e nunca é restaurado em teste não deve ser considerado confiável.

2.12 Diagnóstico básico

Quando uma rotina de backup falha, investigue por partes:

1. A origem existe?
2. O usuário tem permissão de leitura na origem?
3. O destino existe?
4. O usuário tem permissão de escrita no destino?
5. Há espaço livre no destino?
6. O comando funciona manualmente?
7. A tarefa agendada está no usuário correto?
8. Existe saída de erro registrada?
9. O arquivo gerado pode ser listado ou restaurado?

Comandos úteis:

```
$ df -h
$ du -sh dados backups
$ ls -ld dados backups
$ ls -lh backups
$ tar -tzf backups/backup-dados.tar.gz
$ rsync -av --dry-run dados/ backups/dados/
```

Em caso de falha por espaço, lembre que o problema pode estar no destino do backup, no diretório temporário usado pela compressão ou na própria partição de origem.

2.13 Laboratório guiado: backup e restauração

O objetivo deste laboratório é criar dados de teste, gerar backup com `tar`, sincronizar com `rsync`, validar o conteúdo e restaurar em diretório separado.

2.13.1 1. Preparar o ambiente

```
$ mkdir -p ~/lab-backup/dados/projetos
$ mkdir -p ~/lab-backup/backups
$ echo "relatorio inicial" > ~/lab-backup/dados/relatorio.txt
$ date > ~/lab-backup/dados/projetos/gerado_em.txt
$ printf "linha 1\nlinha 2\n" > ~/lab-backup/dados/projetos/notas.txt
```

2.13.2 2. Criar backup com tar

```
$ cd ~/lab-backup
$ tar -czf backups/backup-dados.tar.gz dados
$ ls -lh backups
```

2.13.3 3. Validar o arquivo

```
$ tar -tzf backups/backup-dados.tar.gz
$ sha256sum backups/backup-dados.tar.gz > backups/backup-dados.sha256
$ cat backups/backup-dados.sha256
```

2.13.4 4. Restaurar em diretório separado

```
$ mkdir -p restauracao
$ tar -xzf backups/backup-dados.tar.gz -C restauracao
$ find restauracao -type f
$ cat restauracao/dados/relatorio.txt
```

Observe que a restauração foi feita em `restauracao`, e não diretamente sobre o diretório original.

2.13.5 5. Sincronizar com rsync

```
$ mkdir -p espelho
$ rsync -av --dry-run dados/ espelho/dados/
$ rsync -av dados/ espelho/dados/
$ find espelho -type f
```

Altere um arquivo e sincronize novamente:

```
$ echo "nova linha" >> dados/relatorio.txt
$ rsync -av dados/ espelho/dados/
$ cat espelho/dados/relatorio.txt
```

2.13.6 6. Simular perda e recuperar

Remova apenas um arquivo de laboratório:

```
$ rm dados/projetos/notas.txt
$ ls dados/projetos
```

Recupere o arquivo a partir da restauração já testada:

```
$ cp restauracao/dados/projetos/notas.txt dados/projetos/
$ cat dados/projetos/notas.txt
```

2.13.7 7. Limpeza do laboratório

Ao final, remova somente o diretório criado para o laboratório:

```
$ cd
$ rm -rf ~/lab-backup
```

Antes de usar `rm -rf`, confirme o caminho com `pwd` e `ls`. A remoção recursiva não pergunta se o caminho está errado.

2.14 Resumo

Backup é um processo de proteção e recuperação de dados. Neste capítulo, foram apresentadas diferenças entre cópia, backup, snapshot e restauração, além de rotinas básicas com `tar` e `rsync`. Também foram discutidos retenção, validação, teste de restauração, automação e diagnóstico inicial.

2.15 Perguntas de revisão

1. Qual é a diferença entre uma cópia simples e um backup?
2. Por que snapshot não deve ser tratado automaticamente como substituto de backup?
3. Qual comando lista o conteúdo de um arquivo `tar.gz` sem extrair?
4. Por que é recomendável restaurar primeiro em um diretório separado?
5. Para que serve a opção `-dry-run` do `rsync`?
6. Qual é o risco operacional da opção `-delete` no `rsync`?
7. Que verificações básicas devem ser feitas antes de agendar uma rotina de backup no *cron*?

2.16 Desafio opcional

Crie um script chamado `backup-lab.sh` que gere um arquivo `tar.gz` com data e hora no nome, grave a saída em um arquivo de log e mantenha apenas os cinco backups mais recentes no diretório de destino. Antes de automatizar com *cron*, execute manualmente, valide o conteúdo com `tar -tzf` e restaure pelo menos um arquivo em diretório separado.

3 Estrutura do Sistema de Arquivos

Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a organização hierárquica do sistema de arquivos Linux;
- identificar a função dos principais diretórios definidos pelo FHS;
- diferenciar sistemas de arquivos persistentes, temporários e virtuais;
- reconhecer pontos de montagem comuns em servidores Linux;
- relacionar diretórios do sistema com atividades reais de administração.

Pré-requisitos

Para acompanhar este capítulo, é recomendável que o aluno conheça os comandos básicos de navegação, como *pwd*, *cd*, *ls* e *tree*, além da diferença conceitual entre arquivos, diretórios e processos.

A grosso modo, pode-se dizer que, no Linux, tudo é arquivo. Se há algo que não seja um arquivo, então este algo é um processo. No GNU/Linux (como no UNIX), não há diferença entre arquivo e diretório, uma vez que um diretório é apenas um arquivo contendo nomes de outros arquivos. Imagens, músicas, textos, programas, serviços e assim por diante são todos arquivos. Dispositivos de entrada e saída, e geralmente, todos os dispositivos, são considerados como arquivos. Todos estes arquivos estão organizados de acordo com uma hierarquia, isto é, há critérios que preveem os principais diretórios e seu conteúdo. Estes critérios são definidos por um padrão, o FHS (*Filesystem Hierarchy Standard*) (The Linux Foundation, 2015).

O FHS, originalmente administrado pelo *Free Standards Group* e agora pela *Linux Foundation*, especifica os principais diretórios que precisam estar presentes, e descreve suas finalidades. Ele pode ser obtido a partir do endereço <http://www.pathname.com/fhs/>.

Ao especificar um layout padrão, o FHS simplifica a tarefa de administrar arquivos locais. Enquanto a maioria das distribuições Linux respeita o FHS, provavelmente nenhuma delas o segue integralmente, e a última versão oficial é antiga e não leva em consideração alguns avanços.

Distribuições gostam de experimentar e, eventualmente, algumas das experiências são aceitas. As distribuições Linux gastam muito tempo para garantir que o layout do sistema de arquivos é coerente e evolui ao longo do tempo.

É possível encontrar diretórios de distribuições específicas no diretório raiz. Entre eles */misc* que tem uso geral, normalmente dados, e o */tftpboot* que é usado para o processo de

boot através da rede usando TFTP. Arquivos encontrados neste diretório, são provavelmente relacionados com a inicialização da estações de trabalho sem disco. Observe que ter outros diretórios não viola a FHS, no entanto ter diretórios com finalidades semelhantes ao que o padrão determina é uma violação.

Abaixo a lista dos principais diretórios encontrados sob o /:

Diretório	Está no FHS?	Propósito
/	Sim	Diretório principal de toda a hierarquia de diretórios.
/bin	Sim	Executáveis essenciais que devem estar disponíveis em modo mono-usuário.
/boot	Sim	Arquivos necessários para iniciar o sistema, como o kernel, a imagem initrd ou initramfs, arquivos de configuração do boot e os programas do boot loader.
/dev	Sim	Nós de Dispositivo (Device Nodes) usados para interagir com dispositivos de hardware e software.
/etc	Sim	Arquivos de configuração globais, que afetam todo o sistema.
/home	Sim	Pastas home incluindo configurações pessoais, arquivos, etc
/lib	Sim	Bibliotecas necessárias para executar binários em */bin* e */sbin*.
/lib64	Não	Bibliotecas de 64-bit necessárias para binários executáveis das pastas */bin* e */sbin*, para sistemas que são capazes de rodar programas tanto de 32-bits quanto de 64-bits.
/media	Sim	Ponto de montagem padrão para mídias removíveis, como CDs, DVDs e pendrives USB.
/mnt	Sim	Sistemas de Arquivos Montados Temporariamente.
/opt	Sim	Pacotes de aplicativos opcionais.
/proc	Sim	Sistema de arquivos pseudo-virtual provendo informações sobre o sistema e processos em execução. Pode ser usado para alterar parâmetros do sistema.
/sys	Não	Sistema de arquivos pseudo-virtual provendo informações sobre o sistema e processos em execução. Pode ser usado para alterar parâmetros do sistema. Semelhante a árvore de dispositivos (device tree) e é parte do Sistema Unificado de Dispositivos (Unified Device Model).
/root	Sim	Pasta home do usuário root.
/sbin	Sim	Binários essenciais do sistema.
/srv	Sim	Dados específicos do local servido pelo sistema. Raramente utilizado.
/tmp	Sim	Arquivos temporários; em várias distribuições, o conteúdo é perdido entre reinicializações ou armazenado em memória por meio de tmpfs.
/usr	Sim	Aplicativos multiusuário, utilitários e dados; teoricamente é uma pasta somente leitura.
/var	Sim	Dados que variam durante a execução do sistema.

Tabela 2 – Descrição diretórios no FHS

3.1 Diretório raiz (/) e Sistemas de arquivos

Embora todo o sistema de arquivos possa ser visto como uma grande árvore, como já mencionado, pode haver várias partições e sistemas de arquivos unidos.

A partição e o sistema de arquivos que armazenam o diretório raiz é especial e pode ficar em uma partição dedicada, contendo diretórios como */home*, */var* e */opt* para serem usados como pontos de montagem.

A partição raiz (root) deve conter todos os arquivos essenciais necessários para inicializar o sistema e para depois montar os outros sistemas de arquivos. Desta forma, ela precisa de utilitários, arquivos de configuração, informações do boot loader, e outros dados de inicialização essenciais. Ela deve ser adequada para:

- Inicializar o sistema.
- Restaurar o sistema a partir de backups em mídias externas, como fitas, pendrives, e NAS/Storage.
- Recuperar e/ou reparar o sistema; um administrador de sistema precisa ter as ferramentas para diagnosticar e reconstruir um sistema danificado.

De acordo com o FHS, nenhum aplicativo ou pacote deve criar novos diretórios na raiz.

3.1.1 /bin

O diretório */bin* é muito importante e ele:

Contém programas executáveis e scripts necessários para administradores de sistema e usuários sem privilégios, que são necessários quando há sistemas de arquivos que ainda não foram montados, por exemplo, quando iniciando no modo mono-usuário ou no modo recovery.

Também pode conter arquivos executáveis que são usados indiretamente por scripts.

Pode não conter subdiretórios.

Utilitários binários que não são considerados essenciais para ter lugar no */bin* são colocados no */usr/bin*. Programas que o usuário root nunca utiliza são considerados não essenciais.

Nota: Algumas distribuições recentes estão abandonando a estratégia de separar o */bin* e o */usr/bin* (assim como o */sbin* e o */usr/sbin*) e só tem um diretório com links simbólicos, preservando, assim, a visão com duas pastas. Eles acreditam que o conceito consagrado pelo tempo de colocar o */usr* em uma partição separada se tornou obsoleto.

Programas obrigatórios que devem existir no */bin* incluem:

cat, chgrp, chmod, chown, cp, date, dd, df, dmesg, echo, false, hostname, kill, ln, login, ls, mkdir, mknod, more, mount, mv, ps, pwd, rm, rmdir, sed, sh, stty, su, sync, true, umount, e uname.

e opcionalmente podem estar lá:

csh, ed, tar, cpio, gunzip, zcat, netstat, and ping.

3.1.2 /boot

Arquivos essenciais para iniciar o sistema devem estar no diretório */boot*. Os dois arquivos que são absolutamente essenciais são:

vmlinuz: O kernel do Linux compactado.

initramfs: O sistema de arquivos em memória para partida (initial RAM Filesystem), que é montado antes do diretório raiz estar disponível.

Esses arquivos têm nomes mais longos que dependem da versão do kernel, e com a forma exata dependendo da distribuição Linux. Além disso ao invés de *initramfs*, o arquivo pode ser chamado initrd, que significa RAM disk inicial(initial RAM disk)que era o método antigo, mas o nome sobreviveu.

O conteúdo exato do */boot* varia de acordo com a distribuição e com o tempo; Em um sistema Ubuntu 20 temos:

```
$ ls -lF /boot
total 223492
-rw-r--r-- 1 root root   235822 jun 23 07:01 config-5.3.0-62-generic
-rw-r--r-- 1 root root   235822 jul 10 16:22 config-5.3.0-64-generic
drwx------ 3 root root     4096 dez 31  1969 efi/
drwxr-xr-x 5 root root     4096 ago 14 06:40 grub/
lrwxrwxrwx 1 root root       27 ago 11 08:37 initrd.img -> initrd.img-5.3.0-64-generic
-rw-r--r-- 1 root root 97145259 jul  3 11:20 initrd.img-5.3.0-62-generic
-rw-r--r-- 1 root root 98380078 set  1 16:53 initrd.img-5.3.0-64-generic
lrwxrwxrwx 1 root root       27 ago 11 08:37 initrd.img.old -> initrd.img-5.3.0-62-generic
drwx------ 2 root root    16384 ago 13  2019 lost+found/
-rw-r--r-- 1 root root   182704 jan 28  2016 memtest86+.bin
-rw-r--r-- 1 root root   184380 jan 28  2016 memtest86+.elf
-rw-r--r-- 1 root root   184840 jan 28  2016 memtest86+_multiboot.bin
-rw------- 1 root root  4709896 jun 23 07:01 System.map-5.3.0-62-generic
-rw------- 1 root root  4710447 jul 10 16:22 System.map-5.3.0-64-generic
lrwxrwxrwx 1 root root       24 ago 11 08:37 vmlinuz -> vmlinuz-5.3.0-64-generic
-rw------- 1 root root 11416320 jun 23 07:04 vmlinuz-5.3.0-62-generic
-rw------- 1 root root 11416320 jul 10 16:24 vmlinuz-5.3.0-64-generic
lrwxrwxrwx 1 root root       24 ago 11 08:37 vmlinuz.old -> vmlinuz-5.3.0-62-generic
$
```

Em um passado distante, estes arquivos essenciais eram colocados no /, ao invés de ficarem no diretório */boot*, de acordo com práticas tradicionais do UNIX, que hoje são consideradas obsoletas.

Há várias versões do kernel disponíveis no diretório */boot* e, para cada versão, existem quatro arquivos. (A escolha entre kernels será feita usando o GRUB durante o boot.)

Os outros dois arquivos além do *vmlinuz* e *initramfs* são:

config: O arquivo de configuração usado para compilar o kernel. Está aqui apenas para registro/auditoria e para facilitar o debug.

System.map: A tabela de símbolos do kernel, que é muito útil para debug. Ela contém endereços hexadecimais de todos os símbolos do kernel.

Esses dois arquivos não são necessários para o boot ou a execução do sistema.

Distribuições podem colocar outros arquivos e diretórios no */boot*, como backup dos setores de inicialização e outros dados gerados automaticamente.

3.1.3 /dev

Este diretório contém arquivos especiais que representam dispositivos embarcados ou conectados ao sistema. Estes arquivos especiais são essenciais para que o sistema funcione adequadamente.

Estes arquivos representam dispositivos de caractere *(char / character) (byte-stream)* e dispositivos de bloco de I/O . As interfaces de rede, a princípio, não possuem um arquivo de dispositivo em */dev*; em vez disso, são referenciadas por nome, como *eth0*, *ens33* ou *wlan0*.

Todas as distribuições Linux modernas não embarcadas utilizam o sistema ***udev***, que cria arquivos especiais durante a inicialização do sistema e quando um novo dispositivo é detectado.

O conteúdo do diretório */dev*, quando não está montado, é um diretório vazio; durante a inicialização um sistema de arquivos virtual é criado e montado em */dev* e os nós de dispositivos são criados por demanda pelo ***udev***. Sistemas embarcados Linux podem ser estáticos, ou seja, ter dispositivos fixos e podem não usar o ***udev*** ou usar uma alternativa, como o ***mdev***. Exemplo do conteúdo do */dev*:

```
$ ls -lF /dev/
total 0
crw-------   1 root  root      10,  55 set 14 15:21 acpi_thermal_rel
crw-rw-rw-   1 root  root      14,  12 set 14 15:21 adsp
crw-r--r--   1 root  root      10, 235 set 14 15:21 autofs

crw--w----   1 root  tty         5,   1 set 14 15:21 console
lrwxrwxrwx   1 root  root           11 set 14 15:21 core -> /proc/kcore
drwxr-xr-x  10 root  root          220 set 20 19:16 cpu/
crw-------   1 root  root      10,  59 set 14 15:21 cpu_dma_latency
crw-------   1 root  root      10, 203 set 14 15:21 cuse
drwxr-xr-x   8 root  root          160 set 14 15:21 disk/
brw-rw----   1 root  disk     253,   0 out  8 00:19 dm-0

lrwxrwxrwx   1 root  root           13 set 14 15:21 fd -> /proc/self/fd/
crw-rw-rw-   1 root  root       1,   7 set 14 15:21 full
crw-rw-rw-   1 root  root      10, 229 set 14 15:21 fuse

brw-rw----   1 root  disk       7,  15 out  8 23:10 loop15
brw-rw----   1 root  disk     259,   0 out  9 23:05 loop15p1
brw-rw----   1 root  disk     259,   1 out  8 23:10 loop15p2
brw-rw----   1 root  disk     259,   2 out  9 23:08 loop15p5

brw-rw----   1 root  disk       8,   0 out  8 00:19 sda
brw-rw----   1 root  disk       8,   1 out  8 00:19 sda1
brw-rw----   1 root  disk       8,   2 out  8 00:19 sda2
brw-rw----   1 root  disk       8,   3 out  8 00:19 sda3
brw-rw----   1 root  disk       8,  16 out  8 00:19 sdb
brw-rw----   1 root  disk       8,  17 out  8 00:19 sdb1
brw-rw----   1 root  disk       8,  18 out  8 00:19 sdb2

lrwxrwxrwx   1 root  root           15 set 14 15:21 stderr -> /proc/self/
lrwxrwxrwx   1 root  root           15 set 14 15:21 stdin -> /proc/self/f
lrwxrwxrwx   1 root  root           15 set 14 15:21 stdout -> /proc/self/
drwxr-xr-x   2 root  root          160 set 14 15:21 sys_data/
crw-r--r--   1 root  root      30,  36 out 10 11:37 tcp
crw-rw-rw-   1 root  tty         5,   0 out  9 23:26 tty
crw--w----   1 root  tty         4,   0 set 14 15:21 tty0
crw--w----   1 root  tty         4,   1 out  7 20:50 tty1
crw--w----   1 root  tty         4,  10 set 14 15:21 tty10
```

Observe que dispositivos de caractere começam com **c**, dispositivos de bloco com **b** e links simbólicos com **l**. Outros tipos, como **p** para *pipe*, também podem aparecer. Os dois números antes do mês, nas colunas 5 e 6 da saída longa do *ls*, indicam os números *major* e *minor* do dispositivo. O *major-number* normalmente identifica o tipo ou driver do dispositivo; o *minor-number* identifica uma instância específica dentro desse grupo. A

combinação dos dois deve ser única. A documentação oficial dessa numeração está disponível em <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/admin-guide/devices.txt>.

3.1.4 /etc

Este diretório contém arquivos de configuração do servidor e alguns scripts de inicialização; não devem existir executáveis binários neste diretório. Distribuições costumam adicionar arquivos de configuração e diretórios na pasta */etc*. Por exemplo, a Red Hat adiciona uma série de diretórios como o */etc/sysconfig*, que armazena arquivos e diretórios usados para configurar o sistema.

Outros subdiretórios importantes incluem */etc/skel*, que contém um esqueleto usado como modelo para novos diretórios home, e */etc/init.d*, que contém os scripts de inicialização do System V.

Arquivos e diretórios que podem ser encontrados neste diretório incluem:

csh.login, exports, fstab, ftpusers, gateways, gettydefs, group, host.conf, hosts.allow, hosts.deny, hosts,equiv, hosts.lpd, inetd.conf, inittab, issue, ld.so.conf, motd, mtab, mtools.conf, networks, passwd, printcap, profile, protocols, resolv.conf, rpc, securetty, services, shells, syslog.conf.

3.1.5 /home

Em sistemas Linux, os diretórios de usuários são convencionalmente salvos dentro do diretório */home*, como em */home/coop*, */home/student*, etc. Tudo que seja pessoal como configurações, dados e programas executáveis é armazenado nesta hierarquia de diretórios. */home* também pode conter subdiretórios para grupos ou conjuntos de usuários como */home/estudantes*, */home/funcionarios*, */home/aliens* etc.

Em outros sistemas operacionais baseados em UNIX, o conceito da pasta */home* existe, mas pode ser ligeiramente diferente. Por exemplo no Solaris os diretórios dos usuários são criados em */export/home* e o automount monta as pastas dos usuários sob demanda em */home*. Isso acontece porque é comum que o diretório home esteja na rede em ambientes corporativos, provavelmente em um compartilhamento NFS, e a pasta home será montada automaticamente para o uso.

O Linux tem os mesmos recursos de automount , mas a maioria dos usuários não percebe quando eles estão em uso, e para o caso de sistemas em que tudo é resolvido por apenas um servidor, o conceito de pontos de montagem NFS provavelmente não se aplica.

Um usuário sempre pode substituir a variável de ambiente $HOME pela raiz do seu diretório pessoal, ou utilizar o atalho ; por exemplo, os dois comandos abaixo são equivalentes:

```
$ ls -l $HOME/public_html
$ ls -l  /public_html
```

Existe uma exceção: o diretório home para o super usuário root no Linux é sempre salvo na pasta */root*. Alguns sistemas UNIX podem usar o diretório raiz do sistema / o que pode causar confusão.

3.1.6 /lib

Este diretório deve conter apenas as bibliotecas necessárias para executar os programas da pasta */bin* e */sbin*. Estas bibliotecas são particularmente importantes para iniciar o sistema e executar comandos no sistema de arquivo root(/).

Módulos do kernel ficam na pasta */lib/modules/*.

Os arquivos do PAM (Pluggable Authentication Modules) ficam na pasta */lib/security*.

Sistemas que suportam programas de 32-bit e 64-bit devem manter os dois tipos de biblioteca. Nos sistemas baseados em Red Hat existe um diretório separado para bibliotecas de 32-bit (*/lib*) e para as bibliotecas de 64-bit (*/lib64*).

3.1.7 /media

Este diretório é normalmente usado para montar sistemas de arquivos de mídias removíveis, como CDs, DVDs, pendrives USB e, historicamente, disquetes. Em sistemas desktop modernos, a detecção do dispositivo envolve *udev*, mas a montagem automática costuma ser feita por componentes como *udisks* e pelo gerenciador de arquivos da sessão gráfica. Após a desmontagem e remoção, os diretórios utilizados como pontos de montagem podem ser apagados automaticamente.

Se a mídia removível tiver mais de uma partição, cada uma poderá ser montada em uma pasta diferente. Em muitas distribuições Linux, principalmente em ambientes desktop, o gerenciador de arquivos, como o Nautilus, exibirá os volumes recém-montados.

Nota: Em distribuições modernas, incluindo sistemas derivados de Red Hat, mídias removíveis podem aparecer sob a pasta */run/media/[nome do usuário]/*.

3.1.8 /mnt

Este diretório é fornecido para que o administrador do sistema possa montar temporariamente um sistema de arquivos quando necessário. Um uso comum é para sistemas de arquivos de rede como NFS, Samba, CIFS, AFS

Historicamente, o */mnt* também foi utilizado para estes tipos de montagem que agora são montados sob o */media* (ou */run/media*) em sistemas modernos. De um modo geral, esse diretório não deve ser utilizado por programas de instalação.

3.1.9 /opt

Este diretório é projetado para pacotes de software que desejam manter a maioria dos seus arquivos em um lugar isolado, em vez de espalhá-los por toda a árvore de diretórios do sistema. Por exemplo, se *dolphy_app* fosse o nome de um aplicativo que fosse inteiramente instalado dentro do */opt*, todos os arquivos seriam colocados dentro da pasta */opt/dolphy_app*, incluindo */opt/dolphy_app/bin* para binários e */opt/dolphy_app/man* para man pages.

Isso pode facilitar instalar e desinstalar o software, já que tudo está em uma localização conveniente e isolada. Também torna mais fácil determinar a natureza de cada arquivo e associá-lo a um pacote específico.

Entretanto, ao usar sistemas de pacotes como RPM e APT, também é fácil instalar e remover software de uma forma limpa e controlada, mesmo quando os arquivos instalados por um pacote sejam espalhados por toda a árvore de diretórios.

No Linux, a pasta */opt* é frequentemente usada por software proprietário, ou por aqueles que gostam de evitar complicações causadas pelas diferenças entre distribuições.

Os diretórios */opt/bin*, */opt/doc*, */opt/include*, */opt/info*, */opt/lib*, e */opt/man* são reservados para uso do administrador do sistema local. Os pacotes podem fornecer arquivos que terão links nestas pastas ou até mesmo copiados para elas, mas os pacotes devem ser capazes de funcionar sem os arquivos que estão nestes diretórios especiais.

3.1.10 /proc

Este diretório é um ponto de montagem para um sistema de arquivos virtual dos processos do sistema, em que toda a informação está na memória e não no disco. Assim como o */dev*, o */proc* é vazio se for examinado antes do sistema estar rodando.

O kernel expõe estruturas de dados importantes através do */proc*. Cada processo em execução tem o seu próprio diretório dentro do */proc* contendo informações detalhadas sobre o estado do processo, os recursos em uso no momento e desde que começou a rodar.

Os arquivos no */proc* são chamados de arquivos virtuais e tem propriedades interessantes. Enquanto a maioria parece ter zero bytes de tamanho, eles podem exibir grande quantidade de informação.

Além disso, a maioria dos parâmetros de data e hora dos arquivos virtuais refletem a hora e a data atuais, indicando o fato de que eles estão sempre mudando. Na verdade, as informações contidas nestes arquivos é gerada apenas quando eles são lidos, porque eles não são atualizados periodicamente.

Arquivos virtuais importantes incluem */proc/interrupts, /proc/meminfo, /proc/mounts, e /proc/partitions*, que apresentam uma visão atual do hardware do sistema.

Outros como */proc/filesystems* e o diretório */proc/sys/* fornecem informações sobre a configuração do sistema e interfaces para alterá-las.

Por questões de organização, arquivos contendo informações sobre um mesmo tema são colocados em um subdiretório. Por exemplo, */proc/scsi/* contém informações sobre dispositivos

SCSI. Da mesma forma, cada diretório numérico dentro de */proc* contém informações sobre um processo em execução no sistema.

Vamos examinar extensamente os arquivos do */proc* durante o curso, e veremos com detalhes sobre configuração(tuning) do kernel e monitoramento do sistema.

3.1.11 /sys

Este diretório é o ponto de montagem para o sistema de arquivos virtual sysfs onde toda a informação reside apenas na memória, não no disco. Como o */dev* e o */proc*, o / diretório */sys* está vazio quando o sistema não está rodando.

O sysfs é usado tanto para reunir informações sobre o sistema quanto para modificar o seu comportamento durante a execução. Nesse sentido, se assemelha ao */proc*, porém é mais jovem e melhor estruturado. Por exemplo, quase todos os arquivos virtuais no */sys* contém apenas uma linha, ou apenas um valor; não existem arquivos que produzem saídas longas como acontece no */proc*.

Assim como fizemos com o */proc*, vamos examinar entradas no */sys* no decorrer deste curso para configurar o kernel e para monitorar do sistema.

3.1.12 /root

Este diretório é o home para o usuário root e só deve ser usada para ações que necessitam de privilégios do super usuário. Para todas as ações que podem ser executadas por um usuário não privilegiado, use outra conta.

3.1.13 /sbin

Este diretório contém binários essenciais para a inicialização, restauração, recuperação e/ou reparação além dos binários do diretório */bin*. Também é necessário poder montar outros sistemas de arquivo nas pastas */usr*, */home* e em outras pastas se necessário, após o sistema de arquivos root estiver corretamente montado durante o boot. Normalmente somente o usuário root se utiliza deste diretório.

Os seguintes programas devem estar presentes(alguns apenas se o seus subsistemas estiverem instalados):

fdisk, fsck, getty, halt, ifconfig, init, mkfs, mkswap, reboot, route, swapon, swapoff, update. Lembre-se: Algumas distribuições estão abandonando a estratégia de separar em partições diferentes as pastas */sbin* e */usr/sbin* (e também */bin* e */usr/bin*) e estão colocando os binários em uma pasta e criando links simbólicos para manter compatibilidade.

3.1.14 /tmp

Este diretório é usado para armazenar arquivos temporários e pode ser acessado por qualquer usuário ou aplicação. Entretanto arquivos salvos no diretório */tmp* não têm a existência garantida por muito tempo:

Algumas distribuições rodam cron jobs automáticos, que normalmente removem arquivos com mais de 10 dias de idade. Essa é a política padrão do RHEL 6. Algumas distribuições removem o conteúdo da pasta */tmp* a cada reboot. Essa tem sido a política padrão do Ubuntu. Outras distribuições modernas utilizam um sistema de arquivos virtual, usando o diretório */tmp* apenas como um ponto de montagem para um sistema de arquivos *tmpfs*. Esta tem sido a política padrão para as versões recentes do Fedora. Quando o computador é reiniciado, toda informação é consequentemente perdida.

No caso do *tmpfs*, deve-se evitar criar grandes arquivos, pois eles vão ocupar espaço na memória ao invés de ocupar espaço no disco, e é fácil afetar ou congelar o sistema por falta de memória.

Enquanto a regra geral é que aplicativos devem evitar colocar grandes arquivos na pasta */tmp*, há inúmeros programas que violam esta regra e colocam arquivos grandes no diretório */tmp*. O problema é que mesmo sendo possível colocá-los em outra pasta (talvez por especificar uma variável de ambiente), muitos usuários não têm conhecimento de como fazer esta configuração.

Esta política pode ser desativada em sistemas baseados no systemd, como o Fedora, com o comando:

```
$ systemctl mask tmp.mount
```

e seguido por um reboot.

3.1.15 /usr

O diretório */usr* pode ser visto como uma segunda hierarquia. Ele é usado para arquivos que não são necessários durante a inicialização do sistema. O */usr* não precisa estar na mesma partição do diretório raiz, e pode ser compartilhado através da rede entre os hosts usando a mesma arquitetura.

Os pacotes de software não devem criar subdiretórios diretamente dentro do */usr* , mas links simbólicos são tolerados para fins de compatibilidade.

Este diretório é normalmente montado como somente leitura. Ele contém os binários que não são necessários no modo monousuário. Porém a pasta */usr/local*, é onde binários locais podem ser salvos. As páginas de manual ficam dentro da pasta */usr/share/man*.

Diretório		Função
/usr/bin	⟶	Scripts e executáveis não essenciais, que não são necessários em modo monousuário. Geralmente, isso significa aplicativos do usuário que não são necessários para o sistema funcionar corretamente.
/usr/include	⟶	Arquivos header usados para compilar aplicativos.
/usr/lib	⟶	Bibliotecas para programas salvos nas pastas */usr/bin* e */usr/sbin*.
/usr/lib64	⟶	Bibliotecas para programas 64-bit nas pastas */usr/bin* e */usr/sbin*.
/usr/sbin	⟶	Executáveis não essenciais do sistema como daemons do sistema.
/usr/share	⟶	Dados compartilhados por aplicativos, geralmente independente da arquitetura.
/usr/src	⟶	Código fonte, normalmente para o kernel do Linux .
/usr/X11R6	⟶	Arquivos do ambiente gráfico X Window; Geralmente obsoleto. Não é mais presente em várias distribuições Linux modernas.
/usr/local	⟶	Aplicativos locais específicos do servidor. Subdiretórios comuns: *bin, sbin, lib, share, include, etc.*

Tabela 3 – Descrição diretórios dentro de */usr*

3.1.16 /var

Este diretório contém arquivos de dados variáveis (ou voláteis) que mudam frequentemente durante a operação do sistema. Estes incluem:

- Os arquivos de log.
- Arquivos e diretórios spool para fila de impressão, filas de mensagens, etc.
- Arquivos de dados administrativos.
- Arquivos temporários, como conteúdo do cache.

Obviamente, */var* não pode ser montado como um sistema de arquivos somente leitura. Por razões de segurança, é muitas vezes considerada uma boa ideia montar o */var* como um sistema de arquivos separado. Além disso, se o diretório ficar cheio, não deve bloquear todo o sistema. O */var/log* é diretório onde a maioria dos arquivos de log estão salvos, e do diretório */var/spool* armazena arquivos locais para processos como e-mail, impressão e tarefas agendadas.

3.1.17 /run

O diretório */run* armazena os arquivos transitórios: aqueles que contêm informações sobre o tempo de execução (run time), que podem ser salvas ainda nos primeiros estágios da inicialização do sistema (boot), e que não precisam ser persistentes entre reinicializações.

Geralmente o */run* é implementado como um ponto de montagem vazio. Logo este pseudo *filesystem* (sistema de arquivos artificial) só existe em memória.

Alguns caminhos existentes, como */var/run* e */var/lock*, são apenas links simbólicos para subdiretórios dentro do */run*. Outras pastas, dependendo da distribuição podem também apenas apontar para locais dentro do */run*.

Diretório		Função
/var/ftp	⟶	Usado como a base do servidor FTP.
/var/lib	⟶	Dados persistentes, modificados por programas à medida que são executados.
/var/lock	⟶	Arquivos de Lock usados para controlar acesso simultâneo a recursos.
/var/log	⟶	Arquivos de Log.
/var/mail	⟶	Caixas de correio (mailboxes) dos usuários.
/var/run	⟶	Informação sobre a execução do sistema desde o último boot.
/var/spool	⟶	Tarefas em uma fila ou aguardando para serem processadas, como a fila de impressão.
/var/tmp	⟶	Arquivos temporários que serão preservados entre reboots do sistema, algumas vezes relacionados com o */tmp*.
/var/www	⟶	Raiz da hierarquia dos websites.

Tabela 4 – Descrição diretórios dentro de */var*

3.1.18 lost+found

O lost+found é um diretório especial, não faz parte do FHS mas que precisa ser mencionado, ele é criado automaticamente após a formatação da partição com o ***mkfs.ext2*** (sistemas de arquivos ext2/3/4), a função deste diretório é pré-alocar os blocos de arquivos/diretório durante a execução do programa ***fsck.ext2*** na recuperação de um sistema de arquivos. Isto garante que os blocos de disco não precisarão ser diretamente alocados durante a checagem.

Resumo

A estrutura do sistema de arquivos Linux segue uma organização hierárquica iniciada no diretório raiz, representado por /. O FHS define a finalidade esperada dos principais diretórios, mas cada distribuição pode adotar pequenas variações para atender suas próprias decisões de empacotamento, inicialização e administração.

Para o administrador de sistemas, compreender essa hierarquia é essencial. Diretórios como */etc*, */var*, */home*, */boot*, */dev*, */proc*, */sys* e */run* aparecem constantemente em tarefas de instalação, diagnóstico, recuperação, auditoria e operação de servidores Linux.

Perguntas de revisão

1. Qual é a função do diretório raiz / dentro da hierarquia do Linux?
2. Por que o diretório */etc* é importante para a administração do sistema?
3. Qual é a diferença entre */tmp* e */var/tmp*?
4. O que são os diretórios virtuais */proc*, */sys* e */run*?
5. Por que */var* costuma ser separado em uma partição própria em servidores?

6. Qual é a finalidade do diretório *lost+found* em sistemas de arquivos ext2, ext3 e ext4?

Desafio opcional

Em uma máquina Linux de laboratório, execute os comandos abaixo e registre a função de cada diretório observado:

```
$ ls -ld / /etc /var /home /boot /dev /proc /sys /run /tmp /usr
$ mount | grep -E "/ |/boot|/home|/var|/tmp|/run|/proc|/sys|/dev"
$ df -hT
```

Compare os resultados com a tabela deste capítulo e identifique quais diretórios são sistemas de arquivos reais, quais são pontos de montagem temporários e quais são sistemas de arquivos virtuais.

4 Usuários, Grupos e Permissões

Este capítulo apresenta o modelo tradicional de identidade e controle de acesso do GNU/-Linux. Antes de alterar permissões ou criar usuários em um servidor real, é importante compreender que o sistema decide o acesso a arquivos e processos a partir de três elementos básicos: usuário, grupo e permissões.

Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a relação entre usuários, grupos, UIDs e GIDs;
- identificar os arquivos básicos usados no cadastro de contas locais;
- criar, alterar e remover usuários e grupos;
- interpretar permissões no formato simbólico e octal;
- alterar dono, grupo e permissões de arquivos;
- compreender o uso básico de *umask*, *setuid*, *setgid* e *sticky bit*.

Pré-requisitos

Para acompanhar os exemplos, recomenda-se que o aluno já saiba navegar pelo sistema de arquivos, listar arquivos com *ls*, editar comandos no shell e usar *sudo* quando necessário. Os testes devem ser executados em uma máquina de laboratório, e não diretamente em servidores de produção.

4.1 Criando e gerenciando usuários e grupos no GNU/Linux

Uma das coisas que torna seguro o sistema operacional GNU/Linux (na verdade, qualquer sistema baseado no Unix), é a sua exigência de que cada coisa tenha dono e permissões de uso. Assim, para que seja possível restringir ou permitir o acesso e o uso de determinados recursos a uma ou mais pessoas, é necessário que cada uma tenha um usuário devidamente criado no sistema operacional (DALHEIMER et al., 2002).

4.1.1 Gerenciando usuários

Criar uma conta para cada usuário no sistema operacional não serve apenas para restringir ou permitir o acesso aos recursos oferecidos, mas também para respeitar o espaço que cada usuário tem. Com uma conta, é possível ter os seus próprios diretórios, personalizar o seu desktop, ter atalhos e configurações para os seus programas preferidos, entre outros. Além disso, mesmo que o computador onde o GNU/Linux está instalado seja usado apenas por uma pessoa, é recomendável criar um usuário próprio para ela. Mas, por qual motivo, se o sistema já conta com um usuário nativo, o root? O usuário *root* é o que "manda"no sistema, pois ele tem poderes de administrador, o que significa que ele tem acesso a todos os recursos do sistema operacional. Usá-lo no dia-a-dia não é recomendável, pois se fizer alguma coisa errada, o sistema operacional poderá ser seriamente comprometido.

Para criar, gerenciar ou eliminar contas de usuários no GNU/Linux, é necessário estar "logado"no sistema operacional com o usuário *root* ou outro usuário que tenha privilégios de administrador podendo ser o privilégio concedido pelo comando '***sudo***', como acontece por padrão com a distribuição Ubuntu.

No GNU/Linux, cada conta criada fica armazenada no arquivo de nome *passwd* localizado dentro do diretório */etc/* (ou seja, */etc/passwd*). Esse arquivo contém várias informações sobre cada usuário:

- o seu nome de login (ou seja, o nome que é necessário digitar para entrar no sistema);
- senha criptografada: nos sistemas Unix mais antigos, o resumo da senha podia ficar no próprio */etc/passwd*. Nas distribuições atuais, esse campo normalmente aparece como **x**, indicando que o resumo protegido da senha está em */etc/shadow*, arquivo legível apenas por processos privilegiados. Quando aparecem caracteres como **!** ou ***** no campo correspondente do *shadow*, geralmente a senha está bloqueada ou não é utilizável para autenticação direta;
- UID (User IDentification), número de identificação do usuário. O UID 0 pertence ao usuário *root*. Em distribuições atuais, como Debian, Ubuntu e Rocky Linux, os UIDs abaixo de 1000 normalmente são reservados para contas do sistema, enquanto usuários comuns costumam começar em 1000. A faixa exata pode variar conforme a distribuição e a configuração em */etc/login.defs*;
- GID (Group IDentification), número de identificação do grupo do usuário, segue as mesmas diretrizes do UID. Um mesmo usuário pode fazer parte de mais de um grupo (geralmente, o GNU/Linux permite a participação do usuário em até 32 grupos);
- informações adicionais sobre o usuário (nome completo, dados de contato, etc). Também chamado de GECOS (General Electric Comprehensive Operating System) em alusão a uma funcionalidade existente em um sistema operacional Unix que tinha esse nome. Cada informação é separada da outra por uma vírgula, por exemplo: Usuário do Sistema,Rua

X,1234-4321. Na prática, você pode inserir as informações que achar melhor, não apenas os dados informados anteriormente;;

- diretório "home", ou seja, o diretório principal, e se bem configurado exclusivo, de cada usuário. Em geral, esses diretórios ficam dentro de */home/*;
- shell (interpretador de comandos) do usuário, um programa que interpretará os comandos que o usuário digitar. O GNU/Linux trabalha com vários, entre eles, o bash, o sh e o csh. Caso nenhum shell seja informado, o sistema utilizará o sh como padrão.

Cada parâmetro do usuário é separado por **:** (dois pontos), e cada usuário formatado ocupa uma linha dentro do arquivo /etc/passwd, exemplo de uma linha:

```
lab:x:1001:500:Usuario de TesTe:/home/lab:/bin/bash
```

Os comandos de gerência do usuário são ***useradd, userdel, usermod e passwd***

```
$ useradd -h
Uso: useradd [opções] LOGIN
     useradd -D
     useradd -D [opções]

Opções:
  -b, --base-dir BASE_DIR       diretório base para o diretório pessoal da
                                nova conta
  -c, --comment COMENTÁRIO      campo GECOS da nova conta
  -d, --home-dir DIR_PESSOAL    diretório pessoal da nova conta
  -D, --defaults                exibe ou altera configuração padrão do useradd
  -e, --expiredate DATA_DE_EXPIRAÇÃO  data de expiração da nova conta
  -f, --inactive INATIVO        período de inatividade de senha da nova conta
  -g, --gid GRUPO               nome ou ID do grupo primário da nova
                                conta
  -G, --groups GRUPOS           lista de grupos complementares da nova
                                conta
  -h, --help                    mostrar esta mensagem de ajuda e sair
  -k, --skel SKEL_DIR           use este diretório esqueleto (skeleton) alternativo
  -K, --key CHAVE=VALOR         sobreescreve os padrões de /etc/login.defs
  -l, --no-log-init             não adiciona o usuário aos bancos de dados
                                lastlog e faillog
  -m, --create-home             cria o diretório pessoal do usuário
  -M, --no-create-home          não cria o diretório pessoal do usuário
  -N, --no-user-group           não cria um grupo com o mesmo nome do usuário
  -o, --non-unique              permite criar usuários com UID duplicado
                                (não-único)
  -p, --password SENHA          senha criptografada da nova conta
  -r, --system                  cria uma conta de sistema
  -R, --root CHROOT_DIR         diretório a ser transformado em root
  -s, --shell SHELL             shell de login da nova conta
  -u, --uid UID                 ID de usuário da nova conta
  -U, --user-group              cria um grupo com o mesmo nome do usuário
  -Z, --selinux-user USUÁRIO_SE  usa um USUÁRIO_SE específico para o mapeamento de
                                 usuário SELinux
      --extrausers              Use the extra users database
```

```
$ userdel -h
Uso: userdel [opções] LOGIN

Opções:
  -f, --force                   força remoção dos arquivos,
                                mesmo se não forem do usuário
  -h, --help                    mostrar esta mensagem de ajuda e sair
  -r, --remove                  remove o diretório pessoal e spool de mensagens
  -R, --root CHROOT_DIR         diretório a ser transformado em root
      --extrausers              Use the extra users database
  -Z, --selinux-user            remover todos os mapeamentos de usuário SELinux
                                do usuário
```

```
$ usermod -h
Uso: usermod [opções] LOGIN

Opções:
  -c, --comment COMENTÁRIO      novo valor do campo GECOS
  -d, --home DIR_PESSOAL        novo diretório de login para a nova conta de
                                usuário
  -e, --expiredate DATA_EXPIRA  define data de expiração de conta para
                                DATA_EXPIRA
  -f, --inactive INATIVO        define inatividade de senha após expiração
                                para INATIVO
  -g, --gid GRUPO               forçar usar GRUPO como novo grupo primário
  -G, --groups GRUPOS           nova lista de GRUPOS suplementares
  -a, --append                  anexa o usuário para os GRUPOS suplementares
                                mencionados pela opção -G sem remove-lo de
                                outros grupos
  -h, --help                    mostrar esta mensagem de ajuda e sair
  -l, --login LOGIN             novo valor do nome de login
  -L, --lock                    trava a conta de usuário
  -m, --move-home               move o conteúdo do diretório pessoal para
                                a novo localização (use somente com -d)
  -o, --non-unique              permitir usar UID duplicados (não-únicos)
  -p, --password SENHA          usar senha criptografada para a nova senha
  -R, --root CHROOT_DIR         diretório a ser transformado em root
  -s, --shell SHELL             novo shell de login para a conta de usuário
  -u, --uid UID                 novo UID para a conta de usuário
  -U, --unlock                  destravar a conta de usuário
  -v, --add-subuids FIRST-LAST  add range of subordinate uids
  -V, --del-subuids FIRST-LAST  remove range of subordinate uids
  -w, --add-subgids FIRST-LAST  add range of subordinate gids
  -W, --del-subgids FIRST-LAST  remove range of subordinate gids
  -Z, --selinux-user USUÁRIO    novo mapeamento de usuário SELinux para a conta
                                de usuário
```

```
$ passwd -h
Uso: passwd [opções] [LOGIN]

Opções:
  -a, --all                     reportar estado de senhas em toda as contas
  -d, --delete                  remover a senha para a conta indicada
  -e, --expire                  forçar expiração da senha para a conta indicada
  -h, --help                    mostrar esta mensagem de ajuda e sair
  -k, --keep-tokens             mudar senha somente caso expirada
  -i, --inactive INATIVO        definir senha inativa após expiração para
                                INATIVO
  -l, --lock                    trava a conta indicada
  -n, --mindays MIN_DIAS        define número mínimo de dias antes da troca
                                de senhas para MIN_DIAS
  -q, --quiet                   modo silencioso
  -r, --repository REPOSITÓRIO  mudar senha no repositório REPOSITÓRIO
  -R, --root CHROOT_DIR         diretório a ser transformado em root
  -S, --status                  reportar estado de senha para a conta indicada
  -u, --unlock                  destravar a conta indicada
  -w, --warndays DIAS_AVISO     define dias de aviso de expiração para
                                DIAS_AVISO
  -x, --maxdays MAX_DIAS        define número máximo de dias antes da troca
                                de senhas para MAX_DIAS
```

Para os testes, vamos inicialmente criar um usuário, modificá-lo e removê-lo:

```
$ sudo useradd teste1
$ sudo useradd -G adm teste2
$ sudo useradd -m -G sudo -s /bin/bash -c "Usuario de Lab"lab
$ sudo useradd -m -r -s /bin/nologin -c "Usuario de Sistema"teste3
$ grep teste /etc/passwd
testes:x:1002:1002::/home/testes:/bin/sh
teste1:x:1006:1007::/home/teste1:/bin/sh
teste2:x:1007:1008::/home/teste2:/bin/sh
teste3:x:998:997:Usuario de Sistema:/home/teste3:/bin/nologin
```

```
$ sudo usermod -c "Teste 2"teste2
$ grep teste /etc/passwd
testes:x:1002:1002::/home/testes:/bin/sh
teste1:x:1006:1007::/home/teste1:/bin/sh
teste2:x:1007:1008:Teste 2 :/home/teste2:/bin/sh
teste3:x:998:997:Usuario de Sistema:/home/teste3:/bin/nologin
$ sudo userdel teste1
$ sudo userdel -r teste2
$ grep teste /etc/passwd
testes:x:1002:1002::/home/testes:/bin/sh
teste3:x:998:997:Usuario de Sistema:/home/teste3:/bin/nologin
```

```
$ sudo usermod -m -d $PWD/acesso_ext/lab lab
$ ls -la $PWD/acesso_ext/
total 24
drwxr-xr-x 4 root root 4096 out 12 16:57 .
drwxrwxr-x 4 edson curso 62 out 10 23:11 ..
drwxr-xr-x 3 lab lab 4096 out 12 16:40 lab
drwx------ 2 root root 16384 out 12 16:55 lost+found
$ ls -la $PWD/acesso_ext/lab
total 52
drwxr-xr-x 3 lab lab 4096 out 12 16:40 .
drwxr-xr-x 4 root root 4096 out 12 16:57 ..
-rw-r--r-- 1 lab lab 220 abr 4 2019 .bash_logout
-rw-r--r-- 1 lab lab 3771 abr 4 2019 .bashrc
drwxr-xr-x 2 lab lab 4096 abr 7 2020 .config
-rw-r--r-- 1 lab lab 8980 abr 16 2018 examples.desktop
-rw-r--r-- 1 lab lab 14965 fev 16 2019 .face
lrwxrwxrwx 1 lab lab 5 set 29 2019 .face.icon -> .face
-rw-r--r-- 1 lab lab 807 abr 4 2019 .profile
```

```
$ sudo grep lab /etc/shadow
lab:!:18547::::::
$ sudo passwd lab
Nova senha:
Redigite a nova senha:
passwd: senha atualizada com sucesso
$ su - lab
Senha:
lab@proteu: $ pwd
/Curso/Laboratorio/acesso_ext/lab
lab@proteu: $ exit
sair
$
```

4.1.2 Gerenciando grupos

Lidar com grupos no GNU/Linux é tarefa muito semelhante ao trabalho com usuários. Veja os principais comandos disponíveis:

- ***groupadd*** grupo: cria grupos locais. Em Debian e Ubuntu também existe o helper ***addgroup***, que chama rotinas de criação com comportamento mais amigável; em Rocky/RHEL, o comando usual é ***groupadd***;
- ***groupdel*** grupo: serve para eliminar grupos do sistema;
- ***groups*** usuário: mostra os grupos dos quais um usuário faz parte.

```
$ groupadd -h
Uso: groupadd [opções] GRUPO

Opções:
  -f, --force                   sai com sucesso se o grupo já existe,
                                e cancela -g se o GID já está em uso
  -g, --gid GID                 usa GID para o novo grupo
  -h, --help                    mostrar esta mensagem de ajuda e sair
  -K, --key CHAVE=VALOR         sobreescreve os padrões de /etc/login.defs
  -o, --non-unique              permite criar grupos com GID duplicado
                                (não-único)
  -p, --password SENHA          usa a senha criptografada para o novo grupo
  -r, --system                  cria uma conta de sistema
  -R, --root CHROOT_DIR         diretório a ser transformado em root
      --extrausers              Use the extra users database

$ groupdel -h
Uso: groupdel [opções] GRUPO

Opções:
  -h, --help                    mostrar esta mensagem de ajuda e sair
  -R, --root CHROOT_DIR         diretório a ser transformado em root
  -f, --force                   delete group even if it is the primary group of a user
      --extrausers              Use the extra users database
```

```
$ sudo groupadd teste_lab
$ sudo groupadd Laboratorio
$ grep -E '(teste_lab|Laboratorio):' /etc/group
teste_lab:x:1010:
Laboratorio:x:1011:
$ sudo groupdel teste_lab
$ sudo usermod -a -G Laboratorio lab
$ su - lab
Senha:
lab@proteu: $ groups
lab sudo Laboratorio
lab@proteu: $ exit
sair
```

4.2 Permissões e Donos de Arquivos

Cada arquivo no GNU/Linux possui um conjunto de privilégios para ler, escrever e executar. Essas permissões são atribuídas de forma independente para o dono, para o grupo e para os demais usuários do sistema.

Tais permissões podem ser concedidas individualmente para usuários, para grupos e para todos os usuários do sistema. Este modelo de permissões de arquivo é a parte mais importante do modelo tradicional de segurança.

Quando você executa *ls -l*, como em:

```
$ ls -l disco_teste -rw-rw-r 1 linux curso 1048576 out 9 23:26 disco_teste
```

Temos:

Tipo	Dono	Grupo	Dono	Grupo	Tamanho	Data	Arquivo
-rw-	rw-	r- -	linux	curso	1048576	out 9 23:26	disco_teste

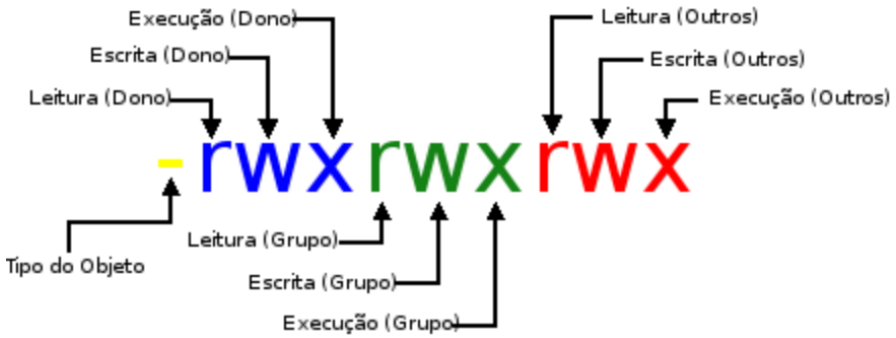
Onde:

- r Significa permissão de leitura (read);
- w Significa permissão de gravação (write);
- x Significa permissão de execução (execution);
- \- Significa permissão desabilitada.

Ou seja, para melhor entendimento, o primeiro caractere determina o tipo de arquivo, os três seguintes indicam as permissões do dono, os três do meio indicam as permissões do grupo

ao qual o arquivo pertence e os três últimos indicam as permissões dos demais usuários. A regra para as permissões sempre será:

- - - Nenhuma permissão;
r- - Permissão de leitura;
r-x Leitura e execução;
rw- Leitura e gravação;
rwx Leitura, gravação e execução.



Após a primeira letra (que indica o tipo de arquivo), existem mais nove que indicam os direitos de acesso concedidos aos usuários em potencial do arquivo. Estes direitos são divididos em três grupos de três letras:

A primeira letra representa o tipo de arquivo / objeto e pode ser :

- **-** : Arquivo convencional
- **d** : Diretório
- **l** : Link Simbólico
- **c** : Dispositivo especial de caracteres.
- **p** : Pipe - Canal fifo
- **s** : Socket

Dono: o usuário que possui o arquivo (também chamado de usuário).
Grupo: o grupo de usuários que têm acesso.
Outros: todos excluindo o dono e o grupo (também chamado mundo).

Na listagem acima, o usuário é linux e o grupo é curso.

Cada um dos trigêmeos pode ter combinações do seguinte conjunto:

- **r** : o acesso de leitura é permitido.
- **w** : acesso de escrita é permitido.
- **x** : executar é permitido.

Se a ação não é permitida, um hífen (-) aparece em vez de uma dessas letras.

Além disso, permissões especializadas existem para cada categoria, como as permissões setuid / setgid.

Assim, no exemplo anterior, o usuário linux e membros do grupo curso têm acesso de leitura e escrita, enquanto os outros têm apenas acesso de leitura.

As permissões de acesso a arquivos são uma parte importante do sistema de segurança que o Linux usa. Qualquer pedido de acesso a um arquivo requer a comparação das credenciais e da identidade do usuário que solicitou o acesso com as credenciais do arquivo.

A autenticação é concedida, dependendo de um destes três conjuntos de permissões, na seguinte ordem:

- Se o solicitante é o dono do arquivo, as permissões do dono do arquivo são usadas.
- Caso contrário, se o solicitante está no grupo que possui os arquivos, as permissões de grupo são examinadas;
- Se isso não for possível, as permissões para os outros usuários são usadas.

4.3 Comandos de manipulação de permissões

4.3.1 chmod, chown e chgrp

Alterar permissões de arquivos é feito com o comando ***chmod***. Você só pode alterar as permissões dos seus arquivos, ou seja aqueles que são associados ao seu usuário, a menos que você tenha privilégios administrativos.

Há maneiras diferentes de utilizar chmod. Por exemplo, para dar ao dono e aos outros a permissão de execução, e ao mesmo tempo remover a permissão de gravação para o grupo:

```
$ ls -l disco_teste
-rw-rw-r- - 1 linux curso 1048576 out 9 23:26 disco_teste
$ chmod uo+x,g-w disco_teste
$ ls -l disco_teste
-rwxr- -r-x 1 linux curso 1048576 out 9 23:26 disco_teste
```

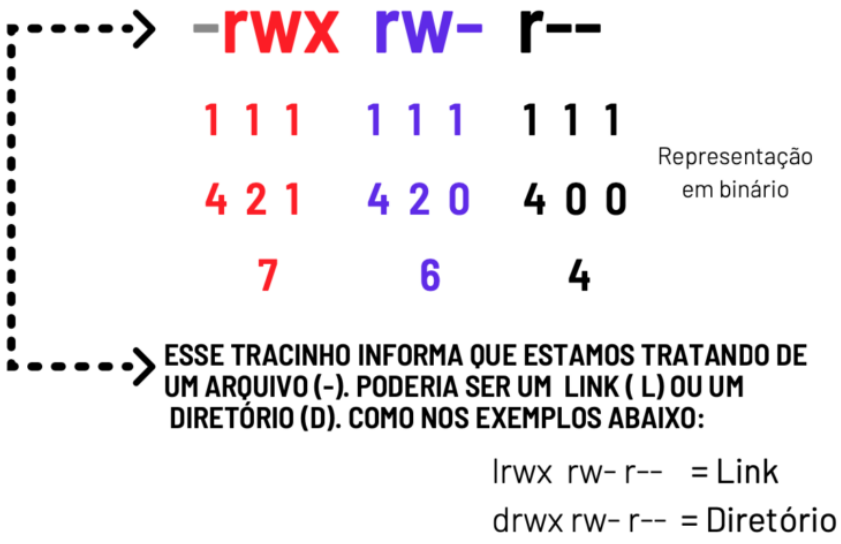
onde **u** representa o usuário (dono), **o** significa outros (ou mundo), e **g** representa o grupo.

As permissões podem ser representadas tanto na forma simbólica quanto na forma octal. A representação octal usa valores como 0755, enquanto a representação simbólica usa expressões como u+rwx,g+rwx,o+rx.

A sintaxe simbólica pode ser difícil de digitar e lembrar, e por isso a abreviação octal é popular por permitir que você defina todas as permissões em uma única etapa. Isto é feito com um algoritmo simples, e um único número é suficiente para especificar os três bits de permissão para cada entidade. Este número é a soma de:

4 se a permissão de leitura é desejada.
2 se a permissão de gravação é desejada.
1 se a permissão de execução é desejada.

Logo 7 significa ler/gravar/executar, 6 significa ler/escrever, e 5 significa ler/executar.



Ao usar o ***chmod*** você deve fornecer um número para cada um dos grupos como em:

```
$ chmod 755 disco_teste
$ ls -l disco_teste
-rwxr-xr-x 1 linux curso 1048576 out 9 23:26 disco_teste
```

Mudar o dono de um arquivo é feito com a ferramenta ***chown*** e mudar o grupo de um arquivo é feito com a ferramenta ***chgrp***.

Somente o superusuário pode alterar as propriedades dos arquivos. Da mesma forma, você só poderá mudar as propriedades do grupo do qual você é um membro.

Alterar o grupo de um arquivo é simples:

```
$ chgrp projeto arquivo
```

e mudar o dono também é simples

```
$ chown coop arquivo
```

você pode mudar os dois ao mesmo tempo com:

```
$ chown coop:projeto arquivo
```

em que você separa o dono do grupo com dois-pontos.

Estes programas aceitam a opção -R, que significa recursivo. Por exemplo:

```
$ chown -R coop:projeto ./
$ chown -R coop:projeto subdir
```

o primeiro comando vai mudar o dono e o grupo de todos os arquivos no diretório atual e em todos os seus subdiretórios, e apenas no diretório subdir e todos os seus subdiretórios no segundo comando.

4.3.2 umask

As permissões-base usadas na criação de arquivos são 0666, pois arquivos comuns não nascem executáveis por padrão. Para diretórios, a base é 0777, pois a permissão de execução é necessária para entrar e atravessar o diretório.

```
$ touch afile
$ mkdir adir
$ ls -l | grep -e afile -e adir
drwxrwxr-x 2 coop coop 4096 Sep 16 11:18 adir
-rw-rw-r- - 1 coop coop 0 Sep 16 11:17 afile
```

Contudo, quando criamos arquivos e diretórios, as permissões finais podem aparecer como 664 para o arquivo e 775 para o diretório. Isso ocorre porque a ***umask*** define quais bits de permissão devem ser removidos das permissões-base. O valor atual pode ser mostrado por:

```
$ umask
0002
```

Este valor é combinado com as permissões de criação padrão para obter o resultado final:

```
0666 & ~0002 = 0664        & = E lógico; ~= negação bit a bit
ou seja:
-rw-rw-r- -
```

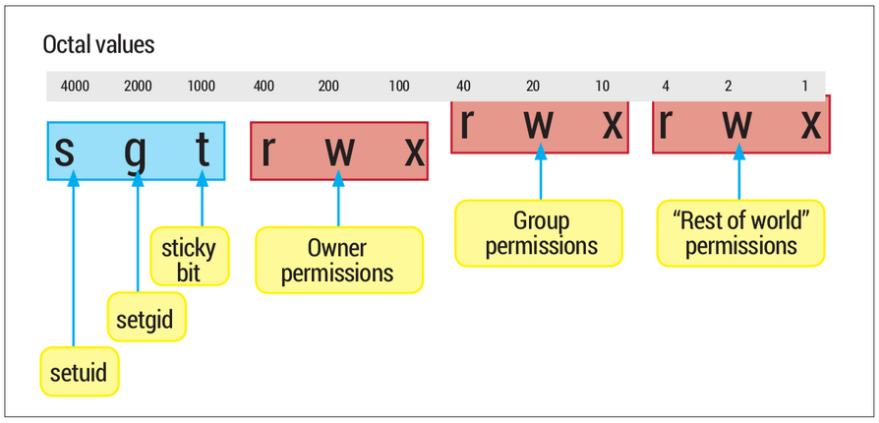
Você pode alterar o umask a qualquer momento com o comando ***umask***:

```
$ umask 0022
```

4.4 Permissões de Acesso Especiais

4.4.1 *setuid, setgid, sticky bit*

Há permissões de acesso especiais que afetam arquivos e diretórios. No formato octal, elas ocupam a quarta posição à esquerda, antes das permissões tradicionais de dono, grupo e outros. Os valores principais são 4 para *setuid*, 2 para *setgid* e 1 para *sticky bit*.



São eles:

- **s** - Quando é usado na permissão de acesso do Dono, ajusta a identificação efetiva do usuário do processo durante a execução de um programa, também chamado de ***bit setuid***. Não tem efeito em diretórios. Quando **s** é usado na permissão de acesso do Grupo, ajusta a identificação efetiva do grupo do processo durante a execução de um programa, chamado de ***bit setgid***. É identificado pela letra *s* no lugar da permissão de execução do grupo do arquivo/diretório. Em diretórios, força que os arquivos criados dentro dele pertençam ao mesmo grupo do diretório, ao invés do grupo primário que o usuário pertence. Ambos **setgid** e **setuid** podem aparecer ao mesmo tempo no mesmo arquivo/diretório. A permissão de acesso especial **s** somente pode aparecer no campo Dono e Grupo.

- **S** - Idêntico a **s**. Significa que não existe a permissão "x" (execução ou entrar no diretório) naquela posição.

- **t** - Em sistemas Unix antigos, o *sticky bit* em arquivos executáveis tinha relação com cache de texto do programa. No Linux atual, o uso relevante para administração está em diretórios: ele impede que usuários removam ou renomeiem arquivos dos quais não são donos, mesmo quando o diretório permite escrita para vários usuários. O exemplo

clássico é o */tmp*: todos podem criar arquivos temporários, mas um usuário comum não deve apagar arquivos de outro usuário. A permissão especial *t* aparece no campo de outros usuários;

- **T** - Idêntico a *t*. Significa que não existe a permissão "x" naquela posição
- **X** - Se você usar **X** ao invés de **x**, a permissão de execução somente é aplicada se o arquivo já tiver permissões de execução. Em diretórios ela tem o mesmo efeito que a permissão de execução x.

Explicação de permissão de acesso especial **X**:

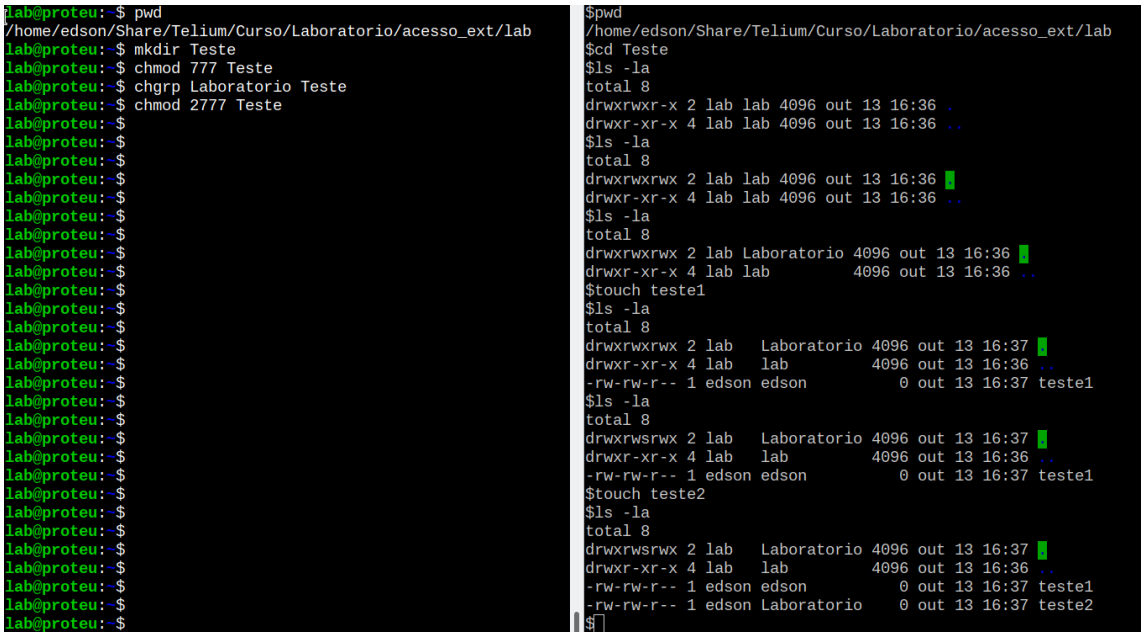
1. Crie um arquivo teste (digitando *touch teste*) e defina sua permissão para rw-rw-r - - (*chmod ug=rw,o=r teste* ou *chmod 664 teste*).
2. Agora use o comando *chmod a+X* teste
3. digite *ls -l*
4. Veja que as permissões do arquivo não foram afetadas.
5. agora digite *chmod o+x teste*
6. digite *ls -l*, você colocou a permissão de execução para os outros usuários.
7. Agora use novamente o comando *chmod a+X teste*
8. digite *ls -l*
9. Veja que agora a permissão de execução foi concedida a todos os usuários, pois foi verificado que o arquivo era executável (tinha permissão de execução para outros usuários).
10. Agora use o comando *chmod a-X teste*
11. Ele também funcionará e removerá as permissões de execução de todos os usuários, porque o arquivo teste tem permissão de execução (confira digitando *ls -l*).
12. Agora tente novamente o *chmod a+X teste*
13. Você deve ter reparado que a permissão de acesso especial **X** é semelhante a **x**, mas somente faz efeito quando o arquivo já tem permissão de execução para o dono, grupo ou outros usuários.

Em diretórios, a permissão de acesso especial **X** funciona da mesma forma que **x**, até mesmo se o diretório não tiver nenhuma permissão de acesso (**x**).

Obs: Ao utilizar valores em octal, a opção especial **X** não faz sentido, pois os valores são atribuídos diretamente a cada campo da permissão!

O ***setuid*** em scripts interpretados normalmente é ignorado pelo Linux por razões de segurança. Quando necessário, esse tipo de privilégio deve ser implementado com programas compilados, bem auditados e com escopo mínimo.

Para teste do ***setgid*** vamos utilizar de dois usuários, um que será o dono do diretório e outro manipulando o diretório:



Resumo do capítulo

Usuários e grupos formam a base da administração de acesso no GNU/Linux. O arquivo */etc/passwd* identifica as contas locais, enquanto as senhas protegidas ficam em */etc/shadow*. As permissões tradicionais são avaliadas na ordem dono, grupo e outros, podendo ser manipuladas com *chmod*, *chown* e *chgrp*. Permissões especiais, como *setuid*, *setgid* e *sticky bit*, devem ser usadas com cuidado, pois alteram o comportamento normal de execução ou de criação e remoção de arquivos.

Perguntas de revisão

1. Qual é a diferença entre UID e GID?
2. Por que o uso diário da conta *root* não é recomendado?
3. Qual é a função do arquivo */etc/shadow*?
4. O que significa a permissão *rwxr-x—*?
5. Qual é a diferença entre *chmod*, *chown* e *chgrp*?

6. Por que o *sticky bit* é usado em diretórios como */tmp*?

Desafio opcional

Crie dois usuários de laboratório e um grupo chamado *projeto*. Configure um diretório compartilhado em que os dois usuários consigam criar arquivos, mas os arquivos novos permaneçam associados ao grupo *projeto*. Depois, registre os comandos usados e explique o papel do *setgid* no diretório.

5 Processos Linux

Um sistema Linux em execução é formado por diversos processos trabalhando ao mesmo tempo. Alguns processos pertencem ao próprio sistema operacional, outros a serviços de rede, tarefas agendadas, sessões de usuários e comandos executados no terminal. Para administrar um servidor, não basta saber iniciar programas: é necessário observar processos, entender seus estados e agir com cuidado quando algum deles consome recursos ou deixa de responder.

Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- diferenciar programa, processo e serviço;
- identificar PID, PPID, usuário dono e estado de um processo;
- listar processos com *ps*, *top* e comandos auxiliares;
- enviar sinais com *kill*, *pkill* e *killall*;
- executar comandos em primeiro e segundo plano;
- compreender a prioridade de processos com *nice* e *renice*;
- aplicar cuidados básicos antes de finalizar processos em servidores.

Pré-requisitos

Para acompanhar este capítulo, é recomendável conhecer comandos básicos do shell, redirecionamento, permissões de usuários e o uso de *sudo*. Os exemplos devem ser executados em ambiente de laboratório, pois encerrar processos de forma indevida pode interromper sessões, serviços e aplicações.

5.1 Conceito de processo

Um programa é um arquivo executável armazenado em disco. Um processo é uma instância desse programa em execução. Quando o usuário executa um comando, o kernel cria um processo, associa recursos a ele e passa a controlá-lo por meio de um identificador numérico chamado *PID*.

Além do PID, cada processo possui um *PPID*, que identifica o processo pai. Em sistemas Linux modernos, o primeiro processo de espaço de usuário normalmente é o *systemd*, com PID 1. Ele inicia e supervisiona várias partes do ambiente do sistema.

```
$ ps -p 1 -o pid,ppid,user,stat,comm
PID PPID USER STAT COMMAND
1 0 root Ss systemd
```

No exemplo, o processo de PID 1 pertence ao usuário *root*, não possui um processo pai comum e está associado ao comando *systemd*.

5.2 Listando processos

O comando *ps* mostra uma fotografia dos processos no momento da execução. Sem opções, ele lista apenas os processos associados ao terminal atual.

```
$ ps
PID TTY TIME CMD
2140 pts/0 00:00:00 bash
2195 pts/0 00:00:00 ps
```

Para uma visão mais completa, uma forma comum é usar:

```
$ ps aux
$ ps -ef
```

As duas formas são muito usadas em administração Linux. A saída do *ps aux* segue estilo BSD, enquanto *ps -ef* segue estilo Unix tradicional. Em ambos os casos, o objetivo é enxergar processos de todo o sistema.

Também é possível escolher exatamente as colunas desejadas:

```
$ ps -eo pid,ppid,user,stat,comm –sort=pid | head
```

Colunas importantes:

- *PID*: identificador do processo;
- *PPID*: identificador do processo pai;
- *USER*: usuário dono do processo;
- *STAT*: estado do processo;
- *COMMAND* ou *CMD*: comando associado ao processo.

5.3 Estados comuns de processos

O estado de um processo indica como ele está se comportando naquele momento. Os estados mais comuns vistos no *ps* são:

- *R*: em execução ou pronto para executar;
- *S*: dormindo, aguardando algum evento;
- *D*: espera não interrompível, normalmente relacionada a I/O;
- *T*: parado ou suspenso;
- *Z*: processo zumbi, já finalizado, mas ainda não coletado pelo processo pai.

Um processo zumbi não consome CPU como um processo normal, mas indica que o processo pai ainda não tratou corretamente o encerramento do filho. Muitos zumbis podem apontar problema em uma aplicação ou serviço.

5.4 Monitoramento interativo

O comando *top* mostra processos de forma interativa, atualizando a tela periodicamente. Ele permite observar consumo de CPU, memória, carga média e processos mais ativos.

```
$ top
```

Dentro do *top*, algumas teclas úteis são:

- *q*: sair;
- *P*: ordenar por uso de CPU;
- *M*: ordenar por uso de memória;
- *k*: enviar sinal para um processo.

Em muitos ambientes também se usa *htop*, que oferece uma interface mais amigável. Ele pode não estar instalado por padrão, portanto deve ser tratado como ferramenta complementar.

5.5 Localizando processos

Para localizar processos pelo nome, é comum usar *pgrep*, *pidof* ou combinar *ps* com *grep*.

```
$ pgrep ssh
$ pidof systemd
$ ps aux | grep ssh
```

O comando *pgrep* é geralmente mais limpo para scripts e diagnósticos, pois retorna diretamente os PIDs encontrados.

5.6 Sinais e finalização de processos

Sinais são mensagens enviadas a processos para solicitar alguma ação. O comando *kill* não significa necessariamente matar um processo; ele envia um sinal. O sinal padrão é o *TERM*, que solicita encerramento ordenado.

```
$ kill PID
```

Sinais comuns:

- *SIGTERM* ou 15: solicita encerramento normal;
- *SIGKILL* ou 9: força o encerramento e não pode ser ignorado;
- *SIGHUP* ou 1: usado por alguns serviços para recarregar configuração;
- *SIGSTOP*: pausa um processo;
- *SIGCONT*: continua um processo pausado.

A sequência administrativa mais segura é tentar primeiro o encerramento normal e usar *SIGKILL* apenas quando necessário.

```
$ kill -15 1234
$ kill -9 1234
```

Atenção: nunca finalize processos críticos em produção sem entender o impacto. Encerrar banco de dados, serviços de autenticação, processos de armazenamento ou sessões administrativas pode causar indisponibilidade ou perda de dados.

Também existem comandos que localizam e sinalizam processos pelo nome:

```
$ pkill nome_do_processo
$ killall nome_do_processo
```

Esses comandos devem ser usados com cuidado, pois podem afetar mais de um processo ao mesmo tempo.

5.7 Primeiro plano e segundo plano

Um processo em primeiro plano ocupa o terminal até terminar. Um processo em segundo plano permite que o shell continue disponível.

```
$ sleep 300 &
[1] 2450
```

O caractere *&* envia o comando para segundo plano. Para listar tarefas do shell atual:

```
$ jobs
```

Para trazer uma tarefa para primeiro plano:

```
$ fg %1
```

Para continuar em segundo plano uma tarefa suspensa:

```
$ bg %1
```

Em uma sessão interativa, *Ctrl+Z* suspende o processo em primeiro plano. Depois disso, *bg* pode continuar a execução em segundo plano ou *fg* pode trazê-lo de volta.

5.8 Prioridade com nice e renice

O Linux permite influenciar a prioridade de CPU de processos por meio do valor *nice*. Quanto maior o valor de *nice*, mais gentil o processo é com os demais, ou seja, menor tende a ser sua prioridade de CPU. Valores menores, especialmente negativos, aumentam a prioridade e normalmente exigem privilégios administrativos.

Executando um comando com prioridade mais baixa:

```
$ nice -n 10 comando
```

Alterando a prioridade de um processo já em execução:

```
$ renice 10 -p 1234
```

Esse recurso não corrige aplicações mal projetadas, mas pode ajudar a reduzir o impacto de tarefas pesadas, como compactações, relatórios e rotinas de backup.

5.9 Laboratório guiado

Neste laboratório, o aluno irá criar processos simples, observá-los e encerrá-los com segurança.

1. Inicie um processo em segundo plano:

```
$ sleep 600 &
```

2. Liste as tarefas do shell:

```
$ jobs
```

3. Localize o processo:

```
$ pgrep sleep
$ ps -o pid,ppid,user,stat,cmd -p PID
```

4. Encerre o processo de forma ordenada:

```
$ kill -15 PID
```

5. Confirme se ele foi encerrado:

```
$ pgrep sleep
```

6. Repita o teste usando *nice*:

```
$ nice -n 10 sleep 600 &
$ ps -o pid,ni,cmd -p PID
```

Resumo do capítulo

Processos são programas em execução, identificados por PID e associados a um usuário, a um processo pai e a um estado. O administrador Linux deve saber listar, localizar, monitorar e sinalizar processos. Comandos como *ps*, *top*, *pgrep*, *kill*, *pkill*, *jobs*, *fg*, *bg*, *nice* e *renice* fazem parte da rotina básica de diagnóstico e operação.

Perguntas de revisão

1. Qual é a diferença entre programa e processo?

2. O que representam PID e PPID?
3. Qual é a diferença entre *SIGTERM* e *SIGKILL*?
4. Por que *kill -9* não deve ser a primeira opção?
5. Para que servem *jobs*, *fg* e *bg*?
6. O que significa aumentar o valor de *nice* de um processo?

Desafio opcional

Crie três processos *sleep* em segundo plano, localize seus PIDs com *pgrep* e exiba suas colunas PID, PPID, usuário, estado, prioridade e comando. Depois, encerre apenas um deles com *SIGTERM*, confirme o resultado e finalize os demais com segurança.

6 Comandos Básicos

Este capítulo apresenta os comandos essenciais para operação diária em sistemas Linux. A intenção não é decorar opções, mas compreender a lógica de uso da linha de comando, a navegação pelo sistema de arquivos e a manipulação segura de arquivos, diretórios e textos.

6.1 Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a sintaxe básica dos comandos Linux;
- navegar pelo sistema de arquivos usando caminhos absolutos e relativos;
- criar, copiar, mover, renomear e remover arquivos e diretórios;
- consultar manuais e interpretar opções de comandos;
- visualizar, filtrar e redirecionar conteúdo de arquivos texto;
- reconhecer comandos que exigem cuidado operacional, como *rm*, *mv* e redirecionamentos destrutivos.

6.2 Pré-requisitos

Para acompanhar este capítulo, recomenda-se que o aluno já tenha noções básicas de terminal, diretórios, arquivos, usuário comum, usuário administrativo e estrutura geral do sistema de arquivos Linux.

6.3 Sintaxe dos comandos

É importante lembrar que a linha de comando é *case sensitive*, isto é, diferencia letras maiúsculas de minúsculas. Portanto, "echo"é diferente de "Echo"e de "ECHO". Isso também vale para nomes de diretórios e arquivos. Os comandos são, em geral, escritos em letras minúsculas. Muitos deles aceitam argumentos iniciados com um ou dois hífens; esses argumentos são as opções.

comando -opção1 -opção2 –opção3 argumentos

Quando os argumentos forem arquivos ou diretórios, tanto o caminho absoluto como o relativo poderão ser usados. Outro ponto importante é que você pode digitar os comandos e

nomes de arquivos ou diretórios pela metade e depois pressionar $\ll$ ***TAB*** $\gg$. O shell "tentará completar"o que falta. Se houver mais de uma opção para completar o que foi digitado, as alternativas possíveis serão mostradas. Este é um recurso que facilita muito o uso da linha de comando.

6.3.1 **man** (manual)

Mostra o manual de um comando.

```
man [comando]
```

6.3.2 **pwd** (print working directory)

Mostra o nome e o caminho do diretório atual (diretório em que o usuário está).

```
curso@curso-desktop:~$ pwd
/home/curso
```

6.3.3 **ls** (list)

Lista os arquivos e subdiretórios de um ou mais diretórios.
Sintaxe básica:

```
ls [opções] [diretório1] [diretório2] ...
```

Exemplos:

- Lista os diretórios e arquivos do /.

  ```
  $ ls /
  ```

- Lista os diretórios e arquivos do */etc* mostrando os detalhes do arquivo no formato de uma lista.

  ```
  $ ls -l /etc
  ```

- Lista o conteúdo do / e do /etc, de uma só vez, use:

  ```
  $ ls / /etc
  ```

- Lista o conteúdo do /etc e seus subdiretórios:

```
$ ls -R /etc
```

- Lista o conteúdo do /tmp incluindo arquivos ocultos :

```
$ ls -a /tmp
```

- Lista o conteúdo do /tmp incluindo arquivos ocultos no formato de uma lista :

```
$ ls -la /tmp
$ ls -l -a /tmp
```

6.3.4 **cd** (change directory)

Altera de diretório.
Sintaxe básica:

```
cd [diretório]
```

Exemplos:

- Para mudar para o diretório raiz (raiz do sistema, o / , não confundir com diretório do usuário root).

```
$ cd /
```

- Para mudar para o diretório home do usuário

```
$ cd
```

- Para voltar um diretório, use:

```
$ cd ..
```

- Para voltar dois diretórios, use:

```
$ cd ../../
```

- Para voltar ao diretório imediatamente anterior ao acessado, basta usar:

```
$ cd -
```

6.3.5 **mkdir** (make directory)

Cria novos diretórios (vazios).

Sintaxe básica:

```
mkdir [caminho1/diretório1] [caminho2/diretório2] ...
```

Exemplos:

- Cria os diretórios "dir1" e "dir2" dentro do diretório */tmp* .

```
$ mkdir /tmp/dir1 /tmp/dir2
```

- Cria os diretórios, criando os diretórios intermediários, não gera erro caso algum diretório já exista, cria os diretórios pais à medida que forem necessários.

```
$ mkdir -p /tmp/dir3/dir4/dir5/dir6/dir7
```

6.3.6 **rmdir** (remove directory)

Remove um ou mais diretórios **vazios**.

Sintaxe básica:

```
rmdir [caminho1/diretório1] [caminho2/diretório2] ...
```

Exemplos:

- Remove os diretórios "dir1" e "dir2" dentro do diretório */tmp* .

```
$ rmdir /tmp/dir1 /tmp/dir2
```

- Remove o diretório e seus subdiretórios.

```
$ rmdir -p /tmp/dir3
```

6.3.7 **touch**

Cria novos arquivos vazios e também para mudar a data e a hora de criação de arquivos existentes.

Sintaxe básica:

```
touch [opções] [arquivo1] [arquivo2] ...
```

-t [[YY]YY]MMDDhhmm[.ss] - Altera a data e hora do arquivo para o ano YYYY (nesse caso, pode-se usar os quatro dígitos ou apenas dois), para o mês MM, para o dia DD, para a hora hh, para o minuto mm e para o segundo ss. As opções de ano e segundo são opcionais.

Exemplos:

- Para criar um arquivo vazio chamado "arquivonovo" no diretório atual.

```
$ touch arquivonovo
```

- Alterar a data do arquivo "arquivonovo" para o dia 16/11 (16 de novembro), e o horário para 16h11min

```
$ touch -t 11161611 arquivonovo
```

6.3.8 **rm** (remove)

Remove arquivos e diretórios.

Sintaxe básica:

```
rm [opções] [arquivo1] [arquivo2] ...
```

Exemplos:

- Remove o "arquivo_teste" no diretório atual.

```
$ touch arquivo_teste
$ rm arquivo_teste
```

- Remove recursivamente diretórios e seu conteúdo.

```
$ mkdir -p dir_teste/dir1
$ touch dir_teste/dir1/arquivo_teste
$ rm -r dir_teste
```

Atenção: em linha de comando, o comando rm remove arquivos sem enviá-los para uma lixeira. Antes de remover diretórios recursivamente, confirme o caminho com pwd e ls. Em ambientes reais, recuperação normalmente depende de backup, snapshot ou ferramentas específicas do sistema de arquivos.

6.3.9 **cp** (copy)

Efetua cópia de arquivos e diretórios.

Sintaxe básica:

```
cp [opções] [origem] [destino]
```

> Importante: O comando cp sempre precisará da origem e do destino, caso contrário acusará um erro.

Exemplos:

- Para copiar o arquivo "teste" do */tmp* para o diretório home do usuário:

```
$ touch /tmp/teste
$ cp /tmp/teste ~
```

- Copiar o diretório e seu conteúdo, para o diretório atual

```
$ mkdir -p /tmp/dir_teste/dir1
$ touch /tmp/dir_teste/dir1/arquivo_teste
$ cp -r /tmp/dir_teste .
```

- Copiar um ou mais arquivos para um outro diretório

```
$ touch teste1 teste2 teste3
$ cp teste1 teste2 teste3 /tmp/dir_teste
```

6.3.10 **mv** (move)

Efetua o movimento de arquivos e diretórios para outro diretório, ou renomeia arquivos e diretórios.

Sintaxe básica:

```
mv [opções] [origem] [destino]
```

Exemplos:

- Renomear arquivo ou diretório:

```
$ touch teste
$ mkdir dir_teste
$ mv teste teste1
$ mv dir_teste dir_teste_1
```

- Mover arquivo e diretório:

```
$ mv dir_teste_1 /tmp
$ mv teste1 /tmp/dir_teste_1
```

- Mover um ou mais arquivos para um outro diretório

```
$ touch teste2 teste3 teste4
$ mv teste2 teste3 teste4 /tmp/dir_teste_1
```

6.3.11 **cat** (concatenate)

Concatena ou mostra o conteúdo de arquivos.

Sintaxe básica:

```
cat [arquivo1] [arquivo2] ...
```

Exemplos:

- Mostra conteúdo do arquivo:

```
$ cat /etc/hosts
```

- Mostra os conteúdos dos arquivos, ou concatena as saídas dos arquivos:

```
$ cat /etc/hosts /etc/passwd
```

6.3.12 **find**

O comando find é usado para procurar por diretórios e arquivos no disco. Possui muitas opções, é recomendável ler o man dele, será mostrado apenas alguns exemplos simples.

Sintaxe básica:

```
find [local_busca] [opções] [motivo_busca]
```

Exemplos:

- Procura por um arquivo ou diretório com o nome "Documentos"a partir do */home*

```
$ find /home -name Documentos
```

- Procura por um arquivo ou diretório com o nome "documentos"independente se é maiúsculo ou minúsculo ou suas combinações a partir do */home* limitando a busca no máximo 2 subdiretórios de profundidade

```
$ find /home -maxdepth 2 -iname documentos
```

- Procura por somente arquivos a partir do */home* limitando a busca no máximo 2 subdiretórios de profundidade

```
$ find /home -maxdepth 2 -type f
```

- Procura por somente diretórios a partir do */home* limitando a busca no máximo 2 subdiretórios de profundidade

```
$ find /home -maxdepth 2 -type d
```

6.3.13 clear

O comando clear limpa o console:
Sintaxe básica:

```
clear
```

6.3.14 exit

Sai do shell (interpretador), faz o logout do usuário.
Sintaxe básica:

```
logout
```

6.3.15 echo

Mostra um texto. É utilizado em scripts para exibir mensagens ao usuário.
Sintaxe básica:

```
echo "mensagem"
```

6.3.16 date

Mostra ou modifica a data e o horário do sistema. Este comando possui muitas opções vale uma lida do man.

Sintaxe básica:

```
date [opções]
```

MMDDhhmm[[**YYyy**][**.ss**]] - Altera a data e hora, MM é o mês, DD é o dia, hh é a hora, mm são os minutos. Opcionalmente, podem ser usados o ano YYyy (com 2 ou 4 dígitos) e os segundos (ss).

Exemplos:

- Mostrar data e hora do sistema

```
$ date
```

- Alterar a data e a hora do sistema.

```
$ date 02011430
```

6.3.17 df

Este comando exibe informações sobre espaço, livre e ocupado, das partições do sistema.

Sintaxe básica:

```
df [opções] [partição]
```

Exemplos:

- Mostrar informações das partições

```
$ df
```

- Mostra informações sobre o espaço livre da partição /dev/sda1 em Kbytes

```
$ df -k /dev/sda1
```

6.3.18 du

Exibe informações sobre o espaço usado pelos diretórios.
Sintaxe básica:

```
du [opções] [diretório]
```

Algumas opções do comando

```
-a : mostra também o espaço ocupado pelos arquivos (de forma recursiva).
-b : mostra o espaço ocupado em bytes.
-c : mostra o total do espaço ocupado.
-h : apresenta o espaço ocupado em formato compreensível para o ser humano
     (1K, 1M, 1G, etc).
-k : mostra o espaço ocupado em Kbytes (é o padrão).
-m : mostra o espaço ocupado em Mbytes.
-s : mostra apenas o total ocupado (sumário).
-x : mostra apenas o total da partição, outras que estejam como subdiretórios
     não são mostradas.
```

Exemplos:

- Para ver o tamanho do arquivo teste.txt em bytes.

```
$ du -b teste.txt
```

- Para obter o tamanho, em Kbytes, de todos os arquivos do diretório corrente com extensão txt e o total de espaço ocupado por estes arquivos.

```
$ du -c *.txt
```

- Para verificar o total de espaço ocupado pelo diretório testes, digite

```
$ du -sh testes
```

Observações

O comando du, sem qualquer opção e sem nome de arquivo ou diretório, fornece a quantidade de espaço ocupada por cada subdiretório que se encontra hierarquicamente abaixo do diretório atual

6.3.19 cal

Mostra o calendário em linha de comando.

Sintaxe básica:

```
cal [opções] [data]
```

6.3.20 uptime

Mostra quanto tempo o sistema está operando.

Sintaxe básica:

```
uptime
```

6.3.21 time

Mede o tempo gasto para executar um processo (programa).

Sintaxe básica:

```
time [comando]
```

Exemplos:

- `$ time ls`

- `$ time find / -iname time`

6.3.22 ln (links)

Este comando cria ligações (links) entre arquivos. Há dois conceitos de ligação em sistemas Unix:

- **Ligação direta** ou **hard link** – cria uma cópia de inode para o arquivo com outro nome (um arquivo pode ter diversos nomes). O arquivo será removido do disco quando o último nome for removido. Não há algo como um nome original, todos os nomes têm o mesmo status. **Não funciona com diretório.**
- **Ligação simbólica** ou **symbolic link** – define um caminho para um arquivo / diretório. Ligações simbólicas podem apontar para arquivos / diretórios em diferentes sistemas de arquivo e não necessitam apontar para arquivos / diretório que realmente existem.

Sintaxe básica:

```
ln [opção] [origem] [destino]
```

Exemplos:

- Cria um link simbólico para um arquivo

```
 $ date > teste_arquivo
$ ln -s ./teste_arquivo ./arquivo_date
```

- Cria um hard link para um arquivo

```
$ ln  ./teste_arquivo ./teste_arquivo_2
```

6.4 Comandos manipulação texto

6.4.1 **Redirecionamento E/S**

Os redirecionadores de E/S definem qual é a entrada/saída de um comando. A tabela abaixo mostra alguns exemplos de operadores de E/S.

Operador	Significado
> ou 1>	redireciona para saída
>>	redireciona para fim de arquivo
<	redireciona para a entrada
<<	redireciona para a entrada e mantém a entrada aberta até que seja digitado algum caractere de EOF (fim de arquivo) como, por exemplo, CTRL+D
>& ou 2>	redireciona a saída de erros
|	redireciona a saída de um comando para a entrada de um outro comando
tee	redireciona o resultado para a saída padrão e para um arquivo, deve ser usado em conjunto com o "|"

Exemplos:

- O comando cat abaixo copia o conteúdo dos arquivos teste1.txt e teste2.txt para o arquivo teste. Caso o arquivo teste não exista, o arquivo é criado. Caso o arquivo teste exista, ele é sobreposto.

```
$ cat teste1.txt teste2.txt > teste
```

- O comando cat abaixo adiciona o conteúdo do arquivo teste3.txt no final do arquivo teste. Caso o arquivo teste não exista, ele é criado.

```
 $ cat teste3.txt >> teste
```

- O comando cat abaixo recebe como entrada o arquivo teste.txt e, portanto, mostra na tela o conteúdo deste arquivo.

```
$ cat < teste.txt
```

- Suponha que o diretório xxxxx não existe. Portanto, o comando *ls xxxxxx* apresenta uma mensagem de erro. Para direcionar a saída de erro deste comando para o arquivo resultado.

```
$ ls xxxxxx >& resultado
$ ls xxxxxx 2> resultado
```

- O comando tee junto com o caractere | para que o resultado de um comando seja redirecionado para a saída padrão e para um arquivo.

```
cat teste.txt | tee teste2.txt
```

Observações

Normalmente cada processo opera com pelo menos três descritores de arquivo sempre abertos: stdin (fd 0), stdout (fd 1) e stderr (fd 2).

6.4.2 tail

Este comando exibe as últimas linhas de um arquivo texto.
Sintaxe básica:

```
tail [opção] [arquivo]
```

Exemplos:

- Mostra as 10 últimas linhas de um arquivo

```
$ tail arquivo
```

- Mostra as 3 últimas linhas de um arquivo

```
$ tail -n 3 arquivo
```

6.4.3 head

Este comando exibe as primeiras linhas de um arquivo texto.

Sintaxe básica:

```
head [opção] [arquivo]
```

Exemplos:

- Mostra as 10 primeiras linhas de um arquivo

```
$ head arquivo
```

- Mostra as 3 primeiras linhas de um arquivo

```
$ head -n 3 arquivo
```

6.4.4 more

Permite fazer a paginação de arquivos ou da entrada padrão. O comando *more* pode ser usado como comando para leitura de arquivos que ocupem mais de uma tela. Quando toda a tela é ocupada, o *more* efetua uma pausa e permite que você pressione *Enter* ou *espaço* para continuar avançando no arquivo sendo visualizado. Para sair do more pressione *q*. A visualização é contínua e não tem volta.

Sintaxe básica:

```
more [arquivo]
```

Exemplos:

```
$ more /etc/services
```

```
$ cat /etc/services | more
```

6.4.5 less

Permite fazer a paginação de arquivos ou da entrada padrão. O comando *less* pode ser usado como comando para leitura de arquivos que ocupem mais de uma tela. Quando toda a tela é ocupada, o *less* efetua uma pausa e permite que você pressione Seta para Cima e Seta para Baixo ou PgUP/PgDown para fazer o rolamento da página, permite a navegação entre as páginas. Para sair do *less* pressione q. O comando less possui opção de busca no arquivo semelhante ao comando *vim*

Sintaxe básica:

```
less [arquivo]
```

Exemplos:

```
$ less /etc/services
```

```
$ cat /etc/services | less
```

6.4.6 sort

Organiza as linhas de um arquivo texto ou da entrada padrão. A organização é feita por linhas e as linhas são divididas em campos que é a ordem que as palavras aparecem na linha separadas por um delimitador (normalmente um espaço).

Sintaxe básica:

```
sort [opções] [arquivo]
```

Exemplos:

```
$ sort  texto.txt
```

```
$ cat texto | sort
```

6.4.7 grep e grep -E

Procura por texto dentro de arquivos ou no dispositivo de entrada padrão. O comando *zgrep* pesquisa diretamente em arquivos compactados com *gzip* e usa opções semelhantes às do *grep*.

Sintaxe básica:

```
grep [opções] [expressão] [arquivo]
```

Exemplos:

```
$ grep  pop3 /etc/services
```

```
$ grep -E "110|143" /etc/services
```

O poder do *grep -E* está na possibilidade de uso de *Regex* (expressões regulares estendidas), pois permite extrair dados de relevância que uma busca literal simples não consegue. O comando *egrep* ainda pode existir por compatibilidade, mas a forma preferencial em materiais atuais é *grep -E*.

6.4.8 **vim** (Vi IMproved)

Este utilitário é um editor de textos e é uma versão melhorada do antigo vi (Vi IMproved). Em algumas distribuições é necessária a instalação do complemento *enhanced* ou do *vim* para se ter as funcionalidades de *vim*.

Sintaxe básica:

```
vim [opções] [arquivo]
```

Algumas opções do comando:

```
-b : permite editar arquivo binário.
-h : exibe opções do aplicativo.
+n : inicializa o cursor na n-ésima linha.
-r : recupera a última versão do arquivo (o arquivo estava sendo editado quan-
     do o vim foi encerrado de forma não normal, por exemplo, devido a falta
     de energia elétrica).
-R : abre arquivo apenas para leitura.
-o : abrir mais de um arquivo simultaneamente , a tela será dividida automati-
     camente (para trocar de tela tecla \textit{<Ctrl>-w}
```

Formas de Trabalho

O vim possui três formas de trabalho: modo de linha, modo de edição e modo de comandos. A mudança de um modo para outro modo é feita através do uso da tecla *Esc*.

Após o arquivo ser aberto pelo vim, o modo de comandos é ativado. No modo de comandos, as teclas digitadas pelo usuário são interpretadas pelo vim como ações a serem executadas dentro do arquivo aberto.

No modo de edição, as teclas digitadas pelo usuário são ecoadas na tela. Para entrar neste modo, pode-se digitar, por exemplo, *"a"* (adicionar), *"i"* (incluir), etc. No modo de linha, o usuário define ações a serem executadas no arquivo como um todo (por exemplo, salvar, substituir caracteres, sair do aplicativo, etc). Para entrar neste modo, deve-se digitar *":"*.

Modo de comandos

0	- mover o cursor para o início da linha em que o cursor está posicionado.
a	- inserir texto após a posição atual do cursor.
A	- inserir texto no final da linha atual.
dd	- deletar linha atual.
[n]+dd	- deletar n linhas a partir da linha atual.
G	- ir para o fim do arquivo.
[n]+G	- ir para a n-ésima linha do arquivo.
h	- voltar um caractere.
H	- ir para a primeira linha exibida na tela atual.

i	- inserir texto a partir da posição atual do cursor.
I	- inserir texto no início da linha atual.
j	- descer uma linha.
J	- juntar a linha atual com a linha seguinte.
[n]+J	- juntar n linhas consecutivas a partir da linha atual.
k	- subir uma linha.
l	- avançar um caractere.
L	- ir para a última linha exibida na tela atual.
n	- procurar, a partir da posição atual do cursor, a próxima ocorrência do texto definido no último comando /.
N	- procurar, a partir da posição atual do cursor e indo em direção ao início do arquivo, a próxima ocorrência do texto definido no último comando /.
o	- inserir uma linha em branco após a linha atual.
O	- inserir uma linha em branco acima da linha atual.
p	- inserir linhas copiadas após a linha atual.
P	- inserir linhas copiadas antes da linha atual.
r	- substituir o caractere atual.
R	- substituir um conjunto de caracteres.
s	- deletar o caractere atual e inserir texto.
S	- apagar linha e inserir novo texto na linha.
u	- desfazer a última alteração feita no texto e ainda não desfeita.
U	- desfazer a última alteração feita no texto.
x	- apagar caractere onde o cursor está posicionado.
$	- mover o cursor para o fim da linha em que o cursor está posicionado.
[n]+y	- copiar n linhas a partir da linha atual.
yy	- copiar a linha atual.
[n]+Y	- copiar n linhas a partir da linha atual.
YY	- copiar a linha atual.
***<Ctrl>* + B**	- voltar uma página.
***<Ctrl>* + F**	- avançar uma página.
<F1>	- exibir tela de ajuda.
[n]+*<Enter>*	- ir para n linhas abaixo da linha atual.
[n]+.	- repetir o último comando que alterou o texto n vezes a partir da posição atual do cursor.
[n] + ~+ *<Enter>*	- inverter a caixa (case) dos n caracteres seguintes ao cursor.
/texto	- procurar pela primeira ocorrência do texto especificado a partir da posição atual do cursor.

Modo de linha

:r arquivo	- incluir arquivo a partir da linha atual do cursor.
:q + <*Enter*>	- sair da tela de ajuda.
:q!	- sair do vim sem salvar as alterações.
:w arquivo	- salvar arquivo com o nome especificado.
:wq	- sair do vim salvando as alterações.
:X	- criptografa o arquivo.
%!xxd	- Para visualizar um arquivo binário no modo hexadecimal
%!xxd -r	- Para retornar do modo hexadecimal

Para treinar o vim com um tutor <https://www.openvim.com/tutorial.html> e um jogo <https://vim-adventures.com/>.

6.4.9 **awk** ou ***gawk***

O programa *awk* (Free Software Foundation, 2020) permite a manipulação de textos a partir de uma sequência de padrões. O nome AWK vem dos sobrenomes dos três criadores da linguagem: Alfred V. Aho, Peter J. Weinberger e Brian W. Kernighan.

O Awk é uma linguagem procedural (ou imperativa) interpretada (não há necessidade de compilação), e executa linha por linha e tem como principal objetivo deixar os scripts de Shell em sistemas POSIX mais poderosos e com muito mais recursos sem utilizar muitas linhas de comando, podendo resolver infinidades de problemas do dia-a-dia do desenvolvedor nesses sistemas operacionais.

Baseada na linguagem C, é utilizada frequentemente por desenvolvedores para processar textos e manipular arquivos. Tem como os paradigmas linguagem de script, procedural e orientada a eventos.

Com awk é possível: gerenciar pequenos banco de dados pessoais; gerar relatórios; validar dados; produzir índices, e fazer outras tarefas de preparação de documentos; fazer experimentos com algoritmos que podem ser adaptados posteriormente a outras linguagens de programação.

A função básica do awk é procurar por linhas (ou outras unidades de texto) em arquivos que possuem certos padrões especificados no programa; para cada "padrão"(pattern) deve haver uma ação associada, isto é, quando uma linha corresponde a um dos padrões, awk realiza a ação especificada naquela linha; depois, awk continua processando as linhas de entrada desta maneira até encontrar o fim do arquivo de entrada; o conjunto de comandos "padrão-ação"pode aparecer literalmente como um programa ou em um arquivo específico com a opção -f "; arquivos de entrada são lidos em ordem; se não há arquivos, a entrada padrão é lida.

Programas em awk são diferentes dos programas na maioria das outras linguagens porque são *data-driven*, isto é, deve-se descrever os dados com os quais se quer trabalhar, e o que fazer quando encontrá-los. Em outras linguagens procedurais normalmente é mais difícil descrever claramente os dados que o programa irá processar. Por isso, programas awk são mais fáceis de escrever e ler.

Algumas opções do comando:

```
-f : especifica o nome do arquivo com os comandos a serem executados.
-F : define quem é o separador de campos (o padrão é o espaço).
-h ou --help : informa as opções do utilitário.
-V ou --version : informa a versão do utilitário gawk.
```

O funcionamento básico do awk é receber uma entrada, normalmente linha por linha, e separar cada linha em campos para manipulação. O separador padrão é o espaço. Para entender melhor, vamos usar a frase "A banana madura é amarela e doce". Como existem sete palavras separadas por espaço, o awk interpreta cada palavra como um campo. No exemplo abaixo, serão exibidos o primeiro, o segundo e o quinto campo:

```
$ echo "A banana madura é amarela e doce" | awk '{print $1 "  " $2 "  " $5}'
```

Como resultado:

```
A  banana  amarela
```

Um exemplo muito difundido é, imaginemos que queremos obter a lista dos usuários do sistema e o shell usado por cada um dos usuários. Estas duas informações fazem parte do arquivo */etc/passwd* onde existe uma entrada para cada usuário do sistema com as seguintes informações: nome, senha, UID, GID, informações do usuário, diretório home e o shell padrão. Por exemplo, a linha abaixo mostra a entrada para o usuário aluno que usa o shell bash como padrão.

```
aluno:x:1001:1001::/home/aluno:/bin/bash
```

Note que os campos são separados por dois pontos (":"). Podemos então, digitar o seguinte comando para obter os nomes e os shells dos usuários:

```
$ awk -F':'  '{print $1 " => " $7}'  /etc/passwd
```

A opção -F define o caractere ":"como separador dos campos. O comando *print* define a exibição de três argumentos: o primeiro campo de cada linha ($1), a string "=> "e o sétimo campo de cada linha ($7). Portanto, a saída do comando acima para a entrada do usuário aluno é

```
aluno => /bin/bash
```

Somando números :

```
$ echo 2 3 | awk 'soma = $1 + $2 { print "Soma = " soma }'
```

Obtendo o Fatorial de um número :

```
$ echo 5 | awk '{ fat = 1
for (i = 1; i <= $1; i++)
       fat = fat * i
    print "Fatorial de " $1 " é " fat
}'
Fatorial de 5 é 120
$ echo $(( 1 * 2 * 3 * 4 * 5 ))
120
```

Outra forma é criar um arquivo e torná-lo executável com o comando chmod. Por exemplo, suponha que o arquivo teste possui o conteúdo mostrado abaixo. Note que agora o separador de campos é definido na variável FS (Field Separator) e que uma frase é exibida antes de processar o arquivo de entrada (bloco BEGIN) e outra frase é exibida no final da execução (bloco END). Apenas os comandos do bloco central são executados de forma repetitiva (ler uma linha do arquivo de entrada e executa o bloco central). Portanto, neste exemplo, o número de vezes que o comando *print* do bloco central é executado depende do número de linhas do arquivo /etc/passwd usado como entrada.

```
#!/usr/bin/awk -f
BEGIN { FS=":"; print "*** inicio ***\n" }
    {
       print $1 " => " $7
    }
END { print "*** fim ***" }
```

Antes de executá-lo, devemos alterar a permissão do arquivo com o comando:

```
chmod u+x teste
```

Em seguida, basta digitar

```
./teste /etc/passwd
```

O resultado será o mesmo do segundo exemplo, exceto que aparecerão duas frases: uma no início e outra no fim da execução.

Mais Exemplos

O exemplo abaixo exibe o resultado da soma de dois valores recebidos na linha de comandos. Da mesma forma da função main( ), as variáveis ARGC e ARGV contém, respectivamente, o número de parâmetros e os argumentos passados na linha de comandos (o parâmetro ARGV[0] possui "awk"). Observe que todos os comandos estão dentro do bloco BEGIN (isto significa que os comandos serão executados uma única vez)

O exemplo abaixo calcula a raiz quadrada de um número. Note que o programa verifica se o parâmetro passado pelo usuário é realmente um número.

```
#!/usr/bin/awk -f
BEGIN {
    if (ARGC != 2 )
       print "Forneca um numero"
    else
       {
          if (ARGV[1] ~ /^[0-9]+$/)
             {
                raiz=sqrt(ARGV[1])
                print "Raiz quadrada de", ARGV[1], "=", raiz
             }
          else
             print "Valor nao numerico"
       }
}
```

Um exemplo de uso de arrays é mostrado a seguir. Note que o índice do array não precisa ser um valor numérico.

```
#!/usr/bin/awk -f
BEGIN {
    vetor[0] = 1
    vetor[1] = 2
    vetor[2] = 3
    dicionario["teste"] = "socorro"
    dicionario["AOU"] = "Linux"
    for (i = 0; i < 3; i++)
       print "vetor[" i "] = " vetor[i]
    print "teste = " dicionario["teste"]
    print "AOU = " dicionario["AOU"]
}
```

O exemplo abaixo mostra outra forma de ler o arquivo */etc/passwd* e exibir o nome do usuário (primeiro campo = $1) e o shell usado pelo usuário (sétimo campo = $7). Os nomes dos usuários são usados como índices do vetor e cada posição do vetor armazena o shell do usuário. A variável NF contém o número de campos da linha que, neste caso, é igual a 7.

```
#!/usr/bin/awk -f
BEGIN { FS=":" }
```

```
{
    usuario[$1]=$NF
}
END {
    for (i in usuario)
       print i,"=>", usuario[i]
}
```

O programa teste mostrado abaixo faz a multiplicação de duas matrizes armazenadas em arquivos distintos (os nomes dessas matrizes são fornecidos na linha de comandos). Observe que o comando *getline*, usado para ler uma linha de arquivo, retorna 1 quando é executado com sucesso e zero quando nenhuma linha é lida. Neste caso, $0 corresponde à linha toda lida, $1 corresponde ao primeiro campo da linha, $2 ao segundo campo da linha, e assim por diante.

```
#!/usr/bin/awk -f
BEGIN{
    # Ler primeira matriz
    valor = getline < ARGV[1]
    lin_A = 0
    while (valor == 1)
       {
          lin_A++
          for (i = 1; i <= NF; i++)
             A[lin_A][i] = $i
          valor = getline < ARGV[1]
       }
    col_A = NF
    # Ler segunda matriz
    valor = getline < ARGV[2]
    lin_B = 0
    while (valor == 1)
       {
          lin_B++
          for (i = 1; i <= NF; i++)
             B[lin_B][i] = $i
          valor = getline < ARGV[2]
       }
    col_B = NF
    if (col_A != lin_B)
       printf "Numero de colunas de A deve ser igual a numero de linhas de B\n"
    else
```

```
       {
          for (i = 1; i <= lin_A; i++)
             {
                for (j = 1; j <= col_B; j++)
                   {
                      C[i][j] = 0;
                       for (k = 1; k <= col_A; k++)
                           {
                              C[i][j] += A[i][k] * B[k][j]
                           }
                       printf C[i][j] " "
                   }
                printf "\n"
             }
       }
}
```

Suponha que os arquivos A e B tenham os conteúdos mostrados abaixo.

Arquivo A

```
1  3  1
2  3  2
```

Arquivo B

```
2  1
3  1
1  2
```

Ao executar o programa com

```
./teste A B
```

o AWK fornece a seguinte resposta:

```
12  6
15  9
```

Variáveis internas O AWK possui variáveis internas que podem ser usadas pelo programador. A tabela abaixo mostra algumas dessas variáveis.

Variável	Descrição
ARGC	Número de argumentos da linha de comandos
ARGV	Vetor com os argumentos da linha de comandos
ENVIRON	Vetor com as variáveis de ambiente
ERRNO	Armazena mensagem de erro
FILENAME	Nome do arquivo de entrada sendo processado no bloco central
FNR	Número de linhas já processadas de um determinado arquivo
FS	Separador dos campos da linha de entrada (o espaço é o padrão)
NF	Número de campos na linha sendo processada
NR	Número de linhas já processadas considerando todos os arquivos de entrada

Observações

A linguagem AWK é muito rica, pois permite, por exemplo, o uso de:
comandos de decisão (*if, else*);
laços (*for, while, do while*);
expressões aritméticas ($+$, $-$, $*$, $/$, $\%$, $++$, $--$, $=$, $+=$, $-=$, $>$, $>=$, $<$, $<=$, $!=$, *etc*);
funções (*exp, log, sqrt, split, substr, getline, etc*);
variáveis, constantes e vetores.
Para obter mais informações, consulte o manual on-line (digite "man awk"ou GNU Awk Manual).

6.5 Resumo

Neste capítulo foram apresentados comandos essenciais para a operação inicial de um sistema Linux. O aluno viu como consultar manuais, navegar entre diretórios, listar conteúdo, criar e remover arquivos, copiar e mover dados, consultar uso de disco, trabalhar com links e manipular textos com comandos como *cat*, *head*, *tail*, *less*, *sort*, *grep* e *awk*.

O ponto mais importante é compreender que a linha de comando trabalha de forma objetiva: o comando executa exatamente o que foi solicitado. Por isso, comandos que removem, sobrescrevem ou movem arquivos exigem conferência prévia do diretório atual, do caminho informado e das opções utilizadas.

6.6 Perguntas de revisão

1. Qual é a diferença entre caminho absoluto e caminho relativo?
2. Por que o Linux diferencia letras maiúsculas e minúsculas em comandos e nomes de arquivos?
3. Qual é a diferença prática entre *rmdir* e *rm -r*?

4. Em que situação o uso de *grep -E* é mais adequado que uma busca simples com *grep*?

5. Qual é a diferença entre um *hard link* e um *symbolic link*?

6. Por que comandos como *rm*, *mv* e redirecionamento com > exigem atenção operacional?

6.7 Desafio opcional

Crie um diretório de laboratório em */tmp*, gere alguns arquivos de teste com *touch* e *echo*, copie parte deles para outro diretório, mova outros arquivos, crie um link simbólico e use *grep*, *sort*, *head*, *tail* e *awk* para consultar o conteúdo produzido. Ao final, remova somente o diretório de laboratório criado, conferindo o caminho antes da remoção.

7 Pipes e Redirecionamento

Pipes e redirecionamentos são recursos fundamentais do shell. Eles permitem conectar comandos, gravar saídas em arquivos, ler dados de arquivos e separar mensagens normais de mensagens de erro. Na prática, esses recursos transformam comandos simples em pequenas sequências de processamento.

7.1 Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a diferença entre entrada padrão, saída padrão e saída de erro;
- usar o pipe para conectar a saída de um comando à entrada de outro;
- redirecionar saídas para arquivos usando > e >>;
- usar redirecionamento de entrada com <;
- compreender o uso básico de << em blocos de texto;
- evitar sobrescrita acidental de arquivos durante operações em linha de comando.

7.2 Pré-requisitos

Para acompanhar este capítulo, recomenda-se que o aluno já conheça comandos básicos como *ls*, *cat*, *grep*, *sort*, *head*, *tail*, *wc* e *less*.

7.3 Entrada e saída padrão

Todo processo em sistemas Unix e Linux normalmente inicia com três canais abertos:

- **stdin** ou entrada padrão, identificada pelo descritor 0;
- **stdout** ou saída padrão, identificada pelo descritor 1;
- **stderr** ou saída de erro, identificada pelo descritor 2.

Quando um comando lê dados do teclado, ele normalmente está usando a entrada padrão. Quando mostra uma resposta na tela, normalmente está usando a saída padrão. Quando mostra uma mensagem de erro, normalmente está usando a saída de erro.

7.4 Pipe

O pipe, representado pelo caractere |, envia a saída padrão de um comando para a entrada padrão de outro comando.

```
comando1 | comando2
```

Um exemplo simples é contar a quantidade de entradas exibidas pelo comando *ls*:

```
$ ls -1 | wc -l
```

Neste exemplo, o comando *ls -1* lista um item por linha. Em vez de essa saída ser exibida diretamente na tela, ela é enviada para o comando *wc -l*, que conta a quantidade de linhas recebidas.

Outro exemplo comum é filtrar processos:

```
$ ps aux | grep ssh
```

O comando *ps aux* lista processos. O comando *grep ssh* recebe essa lista e mostra apenas as linhas que contêm o texto *ssh*.

7.5 Redirecionamento de saída com >

O operador > redireciona a saída padrão de um comando para um arquivo. Se o arquivo não existir, ele será criado. Se já existir, seu conteúdo será sobrescrito.

```
$ echo "primeira linha" > arquivo.txt
$ cat arquivo.txt
primeira linha
```

Atenção

O operador > sobrescreve o arquivo de destino. Antes de executar comandos desse tipo em ambientes reais, confirme o nome do arquivo e o diretório atual com *pwd* e *ls*.

7.6 Redirecionamento de saída com >>

O operador >> também redireciona a saída padrão para um arquivo, mas acrescenta o conteúdo ao final, sem apagar o que já existia.

```
$ echo "segunda linha" >> arquivo.txt
$ cat arquivo.txt
primeira linha
segunda linha
```

Esse operador é muito usado para acrescentar registros em arquivos de teste, relatórios simples e saídas de comandos que precisam ser preservadas.

7.7 Redirecionamento de entrada com <

O operador < faz um comando receber sua entrada a partir de um arquivo, em vez de receber dados digitados pelo teclado.

```
$ wc -l < arquivo.txt
2
```

Neste caso, o comando *wc -l* contou as linhas recebidas pela entrada padrão, mas a entrada veio do arquivo *arquivo.txt*.

7.8 Here document com <<

O operador << permite fornecer um bloco de texto diretamente para um comando. Esse recurso é conhecido como *here document*.

```
$ cat << FIM
linha 1
linha 2
FIM
```

O texto será enviado ao comando *cat* até que a palavra delimitadora informada, neste caso *FIM*, apareça sozinha em uma linha.

7.9 Redirecionando erros

Como a saída de erro usa o descritor 2, é possível gravar mensagens de erro em um arquivo separado.

```
$ ls /diretorio_inexistente 2> erros.txt
$ cat erros.txt
```

Também é possível gravar a saída normal e a saída de erro em arquivos diferentes:

```
$ find /etc -name "*.conf" > resultado.txt 2> erros.txt
```

Quando for necessário juntar saída normal e saída de erro no mesmo arquivo, pode-se usar:

```
$ comando > saida.txt 2>&1
```

Nesse exemplo, *stdout* é enviado para *saida.txt* e, em seguida, *stderr* é redirecionado para o mesmo destino.

7.10 Usando tee

O comando *tee* permite enviar a saída de um comando para a tela e para um arquivo ao mesmo tempo.

```
$ ls -l /etc | tee listagem_etc.txt
```

Para acrescentar ao final de um arquivo existente, use a opção *-a*:

```
$ date | tee -a comandos_executados.txt
```

7.11 Laboratório guiado

Crie um diretório temporário para praticar:

```
$ mkdir -p /tmp/lab_pipe
$ cd /tmp/lab_pipe
```

Crie arquivos de teste:

```
$ echo "linux" > sistemas.txt
$ echo "freebsd" >> sistemas.txt
$ echo "openbsd" >> sistemas.txt
$ echo "linux" >> sistemas.txt
```

Liste, ordene e conte ocorrências:

```
$ cat sistemas.txt | sort
$ cat sistemas.txt | sort | uniq -c
```

Grave a saída em um relatório:

```
$ cat sistemas.txt | sort | uniq -c > relatorio.txt
$ cat relatorio.txt
```

Teste a separação entre saída normal e erro:

```
$ ls sistemas.txt arquivo_inexistente > saida.txt 2> erro.txt
$ cat saida.txt
$ cat erro.txt
```

7.12 Resumo

Pipes e redirecionamentos permitem controlar o fluxo de dados entre comandos, arquivos e mensagens de erro. O pipe conecta comandos. O operador > grava e sobrescreve arquivos. O operador >> acrescenta conteúdo ao final. O operador < usa um arquivo como entrada. O descritor 2 permite tratar erros de forma separada.

7.13 Perguntas de revisão

1. Qual é a diferença entre *stdout* e *stderr*?
2. O que o operador > faz quando o arquivo de destino já existe?
3. Em que situação o operador >> é mais seguro que >?
4. Qual é a função do pipe entre dois comandos?
5. Para que serve o comando *tee*?

7.14 Desafio opcional

Crie um arquivo com nomes repetidos de serviços, ordene o conteúdo, conte as ocorrências de cada nome, grave o resultado em um relatório e separe possíveis erros em outro arquivo. Use pelo menos um pipe, um redirecionamento com >, um redirecionamento com >> e um redirecionamento de erro com *2>*.

8 Expressões Regulares

Expressões regulares são padrões usados para localizar, filtrar e validar texto. Em administração Linux, elas aparecem em comandos como *grep*, *sed*, *awk*, *find*, editores de texto e ferramentas de análise de logs.

O objetivo deste capítulo não é esgotar o assunto, mas apresentar os metacaracteres mais importantes para tarefas rotineiras de um administrador Linux júnior.

8.1 Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender o que é uma expressão regular;
- diferenciar busca literal de busca por padrão;
- usar metacaracteres básicos como ponto, lista, âncoras e quantificadores;
- aplicar expressões regulares com *grep* e *grep -E*;
- evitar erros comuns de interpretação pelo shell.

8.2 Pré-requisitos

Para acompanhar este capítulo, recomenda-se que o aluno já conheça comandos básicos de manipulação de texto, especialmente *cat*, *grep*, *sort*, *head*, *tail* e pipes.

8.3 Conceito

Uma expressão regular, também chamada de *regex*, é uma forma de descrever um padrão de texto. Em vez de procurar apenas uma palavra exata, a expressão regular permite procurar estruturas.

Por exemplo, uma busca literal por *erro* encontra apenas o texto *erro*. Já uma expressão regular pode procurar linhas que começam com uma palavra, linhas que terminam com um número, endereços IP em um log ou nomes que seguem determinado formato.

Observação

Nem toda ferramenta implementa exatamente o mesmo conjunto de expressões regulares. O *grep* tradicional usa expressões regulares básicas. O *grep -E* usa expressões regulares estendidas, nas quais metacaracteres como *+*, *?*, | e agrupamentos são mais naturais de usar.

8.4 Arquivo de laboratório

Para reproduzir os exemplos, crie um arquivo simples:

```
$ cat > /tmp/regex_lab.txt << FIM
linux
Linux
LINUX
teclado
Teclado
telnet
ssh
sshd
http
https
192.168.0.10
192.168.0.200
erro: disco cheio
aviso: memoria alta
FIM
```

Nos exemplos seguintes, usaremos esse arquivo como entrada.

8.5 Busca literal

Antes dos metacaracteres, veja uma busca literal simples:

```
$ grep linux /tmp/regex_lab.txt
linux
```

Por padrão, essa busca diferencia maiúsculas e minúsculas. Para ignorar essa diferença, use *-i*:

```
$ grep -i linux /tmp/regex_lab.txt
linux
Linux
LINUX
```

8.6 Metacaractere ponto

O ponto representa qualquer caractere individual.

```
$ grep "te.lado" /tmp/regex_lab.txt
teclado
```

Nesse exemplo, o ponto ocupa a posição de um único caractere entre *te* e *lado*.

8.7 Listas de caracteres

Colchetes definem uma lista de caracteres aceitos em uma posição.

```
$ grep "[tT]eclado" /tmp/regex_lab.txt
teclado
Teclado
```

A expressão *[tT]* significa: nesta posição, aceite t ou T.

Também é possível usar intervalos:

```
$ grep "[0-9]" /tmp/regex_lab.txt
192.168.0.10
192.168.0.200
```

8.8 Lista negada

Uma lista iniciada com ^ dentro dos colchetes nega os caracteres indicados.

```
$ grep "[^0-9]" /tmp/regex_lab.txt
```

A expressão acima casa linhas que possuem ao menos um caractere que não seja número. Em arquivos reais, isso costuma casar muitas linhas. Por isso, listas negadas devem ser usadas com cuidado.

Para procurar linhas que começam com algo que não é número:

```
$ grep "^[^0-9]" /tmp/regex_lab.txt
linux
Linux
LINUX
teclado
Teclado
telnet
ssh
sshd
http
https
erro: disco cheio
aviso: memoria alta
```

8.9 Âncoras de início e fim

O circunflexo fora dos colchetes indica início da linha. O cifrão indica fim da linha.

```
$ grep "^ssh" /tmp/regex_lab.txt
ssh
sshd
```

A expressão acima encontra linhas que começam com *ssh*.

Para encontrar apenas a linha que termina exatamente com *ssh*:

```
$ grep "^ssh$" /tmp/regex_lab.txt
ssh
```

8.10 Asterisco

O asterisco indica zero ou mais repetições do item anterior.

```
$ grep "sshd*" /tmp/regex_lab.txt
ssh
sshd
```

Neste caso, o *d** significa: zero ou mais letras *d* após *ssh*. Por isso, tanto *ssh* como *sshd* são encontrados.

8.11 Expressões regulares estendidas

Com *grep -E*, alguns operadores ficam mais simples de usar.

O sinal de mais indica uma ou mais repetições:

```
$ grep -E "[0-9]+" /tmp/regex_lab.txt
192.168.0.10
192.168.0.200
```

O ponto de interrogação indica item opcional:

```
$ grep -E "https?" /tmp/regex_lab.txt
http
https
```

A barra vertical funciona como OU:

```
$ grep -E "erro|aviso" /tmp/regex_lab.txt
erro: disco cheio
aviso: memoria alta
```

8.12 Escapando metacaracteres

Quando é necessário procurar um metacaractere como texto comum, usa-se a barra invertida.

Para procurar o ponto literal em endereços IP:

```
$ grep "192\.168" /tmp/regex_lab.txt
192.168.0.10
192.168.0.200
```

Sem escape, o ponto significa qualquer caractere. Com escape, \. significa ponto literal.

8.13 Cuidados com aspas

É boa prática colocar expressões regulares entre aspas. Isso evita que o shell interprete caracteres especiais antes que o comando receba a expressão.

Prefira:

```
$ grep -E "erro|aviso" /tmp/regex_lab.txt
```

Em vez de escrever a expressão sem aspas.

8.14 Laboratório guiado

Crie um arquivo de exemplo com serviços e portas:

```
$ cat > /tmp/servicos_regex.txt << FIM
ssh 22/tcp
http 80/tcp
https 443/tcp
dns 53/udp
ntp 123/udp
mysql 3306/tcp
postgresql 5432/tcp
FIM
```

Liste apenas linhas que começam com *http*:

```
$ grep "^http" /tmp/servicos_regex.txt
```

Liste serviços TCP:

```
$ grep "tcp$" /tmp/servicos_regex.txt
```

Liste serviços cujas portas têm quatro dígitos:

```
$ grep -E " [0-9]{4}/" /tmp/servicos_regex.txt
```

Liste *ssh* ou *dns*:

```
$ grep -E "^(ssh|dns)" /tmp/servicos_regex.txt
```

8.15 Resumo

Expressões regulares descrevem padrões de texto. O ponto casa um caractere qualquer. Colchetes definem listas. Âncoras marcam início e fim de linha. O asterisco, o mais e a interrogação controlam repetições. O *grep -E* facilita o uso de expressões estendidas. Em linha de comando, expressões devem ser protegidas com aspas para evitar interpretação indevida pelo shell.

8.16 Perguntas de revisão

1. Qual é a diferença entre busca literal e busca por expressão regular?
2. O que o ponto representa em uma expressão regular?
3. Qual é a diferença entre `^ssh` e `^ssh$`?
4. Por que é recomendável usar aspas ao passar uma expressão regular para o shell?
5. Em que situação *grep -E* é mais prático que *grep*?

8.17 Desafio opcional

Crie um arquivo com linhas simulando logs de serviço. Inclua linhas com as palavras *erro*, *aviso*, *info*, endereços IP e nomes de serviços. Use *grep* e *grep -E* para separar linhas por severidade, localizar IPs e encontrar somente entradas relacionadas a serviços específicos.

9 Redes no Linux

9.1 Objetivos

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender os conceitos básicos de interface de rede, endereço IP, máscara, rede, broadcast, gateway e rota;
- identificar o estado das interfaces de rede no Linux;
- configurar endereços IPv4 temporariamente com o comando ***ip***;
- consultar rotas e vizinhança ARP/NDISC;
- diferenciar configuração temporária e configuração persistente;
- seguir uma sequência básica de diagnóstico de conectividade.

9.2 Pré-requisitos

Para acompanhar este capítulo, é recomendável que o aluno já saiba:

- navegar no sistema de arquivos;
- executar comandos como usuário comum e com privilégios administrativos;
- interpretar saídas simples de comandos no terminal;
- compreender a diferença entre arquivo, processo e serviço.

9.3 Conceitos fundamentais

Antes de usar comandos, é importante compreender o que está sendo configurado. Uma interface de rede é o ponto de conexão do sistema operacional com uma rede. Ela pode representar uma placa Ethernet, uma interface Wi-Fi, uma interface virtual de máquina virtual, uma VLAN, uma bridge ou outro tipo de dispositivo lógico.

Um endereço IPv4 identifica a máquina dentro de uma rede. A máscara, normalmente representada por um prefixo como **/24**, informa qual parte do endereço identifica a rede e qual parte identifica o host. No exemplo **192.168.1.10/24**, a rede é **192.168.1.0/24**, o broadcast é **192.168.1.255** e os hosts utilizáveis ficam, em regra, entre **192.168.1.1** e **192.168.1.254**.

O gateway padrão é o roteador usado quando o destino não está na mesma rede local. Se a máquina precisa acessar outra rede, ela consulta a tabela de roteamento. Se não houver uma rota mais específica, a rota **default** será utilizada.

Atenção

Configurações feitas diretamente com o comando ***ip*** são temporárias. Elas normalmente são perdidas após reiniciar a máquina ou reiniciar o serviço de rede. A configuração persistente depende da distribuição e do gerenciador de rede utilizado.

9.4 O comando ip

Nas distribuições Linux modernas, o comando principal para consultar e alterar informações de rede é o ***ip***, do conjunto **iproute2**. Ele substitui comandos antigos como ***ifconfig***, ***route*** e ***arp***, que ainda podem aparecer em materiais legados ou sistemas antigos.

Este comando é usado para os seguintes fins:

- mostrar quais interfaces existem no sistema;
- mostrar o estado operacional das interfaces;
- mostrar, adicionar e remover endereços IP;
- ativar ou desativar interfaces;
- consultar e modificar a tabela de roteamento;
- consultar a vizinhança ARP, no IPv4, e NDISC, no IPv6;
- gerenciar interfaces virtuais, VLANs, túneis e outros recursos.

Sintaxe básica

Observações

```
ip OBJECT COMMAND
ip [options] OBJECT COMMAND
ip OBJECT help
```

Os objetos mais úteis para este módulo são apresentados na tabela 6.

Objeto	Forma abreviada	Objetivo
address	a addr	Consultar, adicionar ou remover endereços IPv4 e IPv6 em uma interface.
link	link	Consultar ou alterar o estado de uma interface de rede.
neighbour	n neigh	Consultar ou alterar entradas de vizinhança ARP/N-DISC.
route	r	Consultar, adicionar ou remover rotas.
maddress	maddr	Consultar endereços multicast.
monitor	monitor	Acompanhar eventos de rede via netlink.
netns	netns	Gerenciar namespaces de rede, recurso comum em containers e laboratórios avançados.

Tabela 6 – Objetos mais usados do comando ip neste módulo

Observações

Para obter ajuda sobre cada objeto, use:

```
ip address help
ip link help
ip route help
ip neigh help
```

9.5 Interfaces de rede

9.5.1 Listando interfaces

Para listar interfaces e seus estados operacionais:

```
ip link show
ip link show up
```

Uma saída típica mostra o nome da interface, flags, MTU, estado e endereço MAC:

```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
    fq_codel state UP
    link/ether 08:00:27:12:34:56 brd ff:ff:ff:ff:ff:ff
```

Os nomes **eth0**, **eth1** ainda aparecem em muitos exemplos e sistemas legados. Em instalações modernas é comum encontrar nomes como **enp0s3**, **ens160**, **eno1** ou **wlp2s0**.

9.5.2 Ativando e desativando interfaces

Sintaxe

```
ip link set dev <interface> up
ip link set dev <interface> down
```

Exemplo:

```
ip link set dev enp0s3 up
ip link set dev enp0s3 down
```

9.5.3 Alterando MTU

A MTU define o tamanho máximo do quadro que a interface transmite sem fragmentação. Em redes Ethernet comuns, o valor típico é **1500**.

```
ip link set dev enp0s3 mtu 1500
```

Alterar MTU sem conhecer a rede pode causar perda de conectividade ou problemas intermitentes. Em um primeiro diagnóstico, apenas consulte o valor:

```
ip link show dev enp0s3
```

9.6 Endereçamento IP

9.6.1 Exibindo endereços

Digite o seguinte comando para listar todos os endereços IP associados às interfaces:

```
ip address show
ip addr
ip a
```

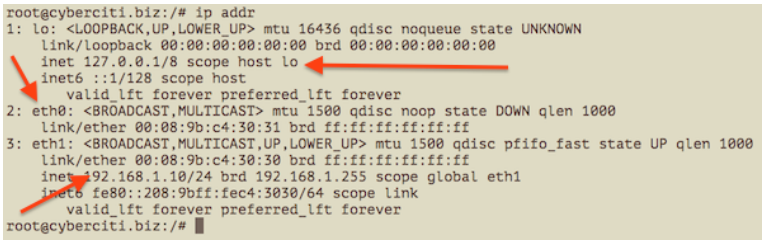


Figura 4 – Mostra os endereços associados à interface, *inet* para IPv4 e *inet6* para IPv6.

Para filtrar por família de endereços:

```
ip -4 address show
ip -6 address show
```

Para consultar uma interface específica:

```
ip address show dev enp0s3
```

9.6.2 Atribuindo endereço IP temporário

Sintaxe

```
ip address add <ip/prefixo> dev <interface>
```

Exemplo:

```
ip address add 192.168.1.200/24 dev enp0s3
```

É possível atribuir mais de um endereço IP à mesma interface, mas isso deve ser feito com critério para evitar conflito de endereços e confusão na origem dos pacotes:

```
ip address add 192.168.1.201/24 dev enp0s3
ip address add 192.168.1.202/24 dev enp0s3
```

9.6.3 Removendo endereço IP temporário

Sintaxe

```
ip address del <ip/prefixo> dev <interface>
```

Exemplo:

```
ip address del 192.168.1.200/24 dev enp0s3
```

Para remover vários endereços de uma interface, use **flush** com cuidado:

```
ip address flush dev enp0s3
```

Cuidado operacional

O comando ***ip address flush*** remove endereços da interface indicada. Se executado em uma interface usada para acesso remoto, como SSH, a sessão pode ser interrompida.

9.6.4 Broadcast

Em redes IPv4, o broadcast é o endereço usado para alcançar todos os hosts daquela rede local. Em uma rede **192.168.1.0/24**, o broadcast normalmente é **192.168.1.255**. O comando ***ip*** costuma calcular esse valor automaticamente quando o prefixo está correto.

Também é possível definir explicitamente:

```
ip address add 192.168.1.50/24 brd + dev enp0s3
```

Vendo o broadcast na interface:

```
ip address show dev enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
    fq_codel state UP
    link/ether 08:00:27:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.50/24 brd 192.168.1.255 scope global enp0s3
```

9.7 Roteamento

9.7.1 Consultando rotas

Para exibir a tabela de roteamento principal:

```
ip route show
ip r
```

Resultado típico:

```
default via 192.168.1.1 dev enp0s3
192.168.1.0/24 dev enp0s3 proto kernel scope link src 192.168.1.10
```

Neste exemplo:

- **192.168.1.0/24 dev enp0s3** indica a rede diretamente conectada;
- **default via 192.168.1.1** indica o gateway padrão;
- **src 192.168.1.10** indica o endereço de origem preferencial para aquela rota.

Para verificar qual rota será usada para alcançar um destino:

```
ip route get 8.8.8.8
ip route get 192.168.1.20
```

9.7.2 Acrescentando rota temporária

Sintaxe

```
ip route add <rede/prefixo> via <gateway>
ip route add <rede/prefixo> dev <interface>
ip route add default via <gateway>
```

Exemplos:

```
ip route add default via 192.168.1.1
ip route add 10.20.30.0/24 via 192.168.1.254
ip route add 10.100.201.0/24 dev enp0s8
```

9.7.3 Removendo rota temporária

Sintaxe

```
ip route del default
ip route del <rede/prefixo> via <gateway>
ip route del <rede/prefixo> dev <interface>
```

Exemplo:

```
ip route del 10.20.30.0/24 via 192.168.1.254
```

9.8 ARP e vizinhança

Em IPv4, antes de enviar um pacote para outro host na mesma rede local, o sistema precisa descobrir o endereço MAC correspondente ao IP de destino. Esse processo usa ARP. No IPv6, a função equivalente é feita pelo Neighbor Discovery Protocol, também conhecido como NDISC.

9.8.1 Mostrando a tabela de vizinhança

Sintaxe

```
ip neigh show
ip n show
```

Exemplo:

```
192.168.1.1 dev enp0s3 lladdr 08:00:27:aa:bb:cc REACHABLE
192.168.1.20 dev enp0s3 lladdr 08:00:27:40:dd:d4 STALE
```

Estados comuns:

- **REACHABLE**: o vizinho foi validado recentemente;
- **STALE**: a entrada existe, mas precisa ser confirmada novamente;
- **DELAY**: o kernel enviou tráfego e está aguardando confirmação;
- **FAILED**: a resolução falhou.

9.8.2 Removendo uma entrada de vizinhança

Sintaxe

```
ip neigh del <ip> dev <interface>
```

Exemplo:

```
ip neigh del 192.168.1.5 dev enp0s3
```

Adicionar entradas ARP permanentes é uma tarefa administrativa específica e não é necessária na maior parte dos ambientes de administração inicial.

9.9 Configuração persistente

O comando ***ip*** é ideal para consulta, teste, laboratório e diagnóstico imediato. Para que a configuração sobreviva ao boot, é necessário usar o mecanismo persistente da distribuição.

9.9.1 Debian e Ubuntu

Em servidores Ubuntu 24.04 LTS, é comum encontrar o **Netplan**. Em Debian, pode-se encontrar **/etc/network/interfaces**, **NetworkManager**, **systemd-networkd** ou outro padrão definido durante a instalação.

Exemplo simples de Netplan:

```
network:
  version: 2
  ethernets:
    enp0s3:
      addresses:
        - 192.168.1.10/24
      routes:
        - to: default
          via: 192.168.1.1
```

```
    nameservers:
      addresses:
        - 192.168.1.1
        - 8.8.8.8
```

Após editar o arquivo, a aplicação geralmente é feita com:

```
netplan try
netplan apply
```

9.9.2 Rocky Linux, Red Hat e derivados

Em Rocky Linux 9 e Red Hat Enterprise Linux 9, o padrão é usar o **NetworkManager**, normalmente administrado pelo comando ***nmcli***.

Exemplo:

```
nmcli connection show
nmcli connection modify "System enp0s3" ipv4.addresses 192.168.1.10/24
nmcli connection modify "System enp0s3" ipv4.gateway 192.168.1.1
nmcli connection modify "System enp0s3" ipv4.dns "192.168.1.1 8.8.8.8"
nmcli connection modify "System enp0s3" ipv4.method manual
nmcli connection up "System enp0s3"
```

Arquivos antigos em **/etc/sysconfig/network-scripts/ifcfg-*** ainda aparecem em materiais de CentOS 7 e ambientes legados. Eles devem ser compreendidos como contexto histórico ou compatibilidade, não como primeira escolha em Rocky Linux 9.

9.10 Sequência de diagnóstico

Ao diagnosticar conectividade, evite começar pelo DNS ou pelo firewall. Primeiro confirme se a base da rede está correta.

1. Verificar o link

```
ip link show
ip link show dev enp0s3
```

Procure sinais como **UP**, **LOWER_UP**, **NO-CARRIER** ou **DOWN**. Se a interface está sem link, IP e rota não resolverão o problema.

2. Verificar o endereço IP

```
ip address show dev enp0s3
```

Confirme se o IP, o prefixo e a interface estão corretos.

3. Verificar a rota

```
ip route show
ip route get 8.8.8.8
```

Se não houver rota default, a máquina pode falar com a rede local, mas não com outras redes.

4. Testar conectividade local

```
ping -c 4 192.168.1.1
```

Neste ponto, o alvo deve ser o gateway ou outro host da mesma rede.

5. Testar conectividade externa por IP

```
ping -c 4 8.8.8.8
```

Se o teste por IP funciona e o teste por nome não funciona, o problema provável está em resolução de nomes.

6. Verificar DNS do cliente

```
resolvectl status
cat /etc/resolv.conf
```

O estudo detalhado de DNS será feito em capítulo próprio. Neste capítulo, o foco é identificar se a máquina possui servidores DNS configurados.

9.11 Comandos legados e equivalentes atuais

9.12 Laboratório

O laboratório pode ser feito com máquinas virtuais no VirtualBox. Para o Módulo 1, use máquinas Linux sem interface gráfica, com pelo menos 512 MB de RAM e uma instalação mínima. As redes utilizadas entre VMs serão redes internas do VirtualBox, sem DHCP.

Comando antigo	Comando atual
ifconfig -a	ip address show
ifconfig enp0s3 down	ip link set dev enp0s3 down
ifconfig enp0s3 up	ip link set dev enp0s3 up
ifconfig enp0s3 192.168.2.24 netmask 255.255.255.0	ip address add 192.168.2.24/24 dev enp0s3
ifconfig enp0s3 mtu 9000	ip link set dev enp0s3 mtu 9000
route	ip route show
route add default gw 192.168.2.254	ip route add default via 192.168.2.254
arp -a	ip neigh show
arp -d 192.168.2.254	ip neigh del 192.168.2.254 dev enp0s3
netstat -tulpn	ss -tulpn

Tabela 7 – Equivalência entre comandos legados e comandos atuais

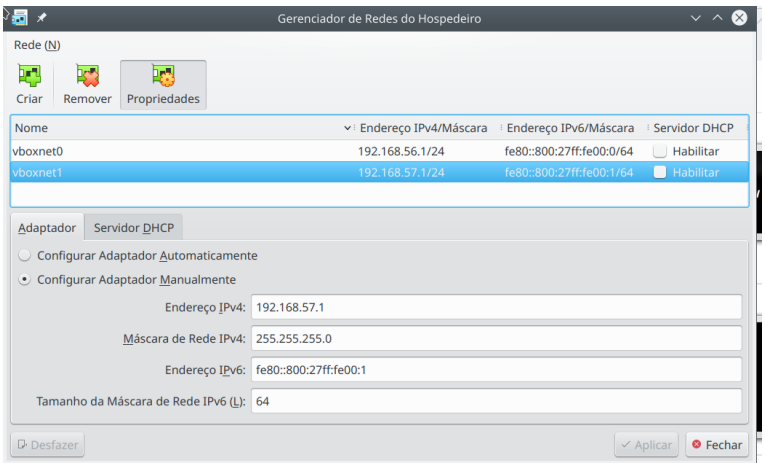


Figura 5 – Interface do gerenciador de redes do VirtualBox.

Figura 6 – Configuração da interface da VM para rede interna no VirtualBox.

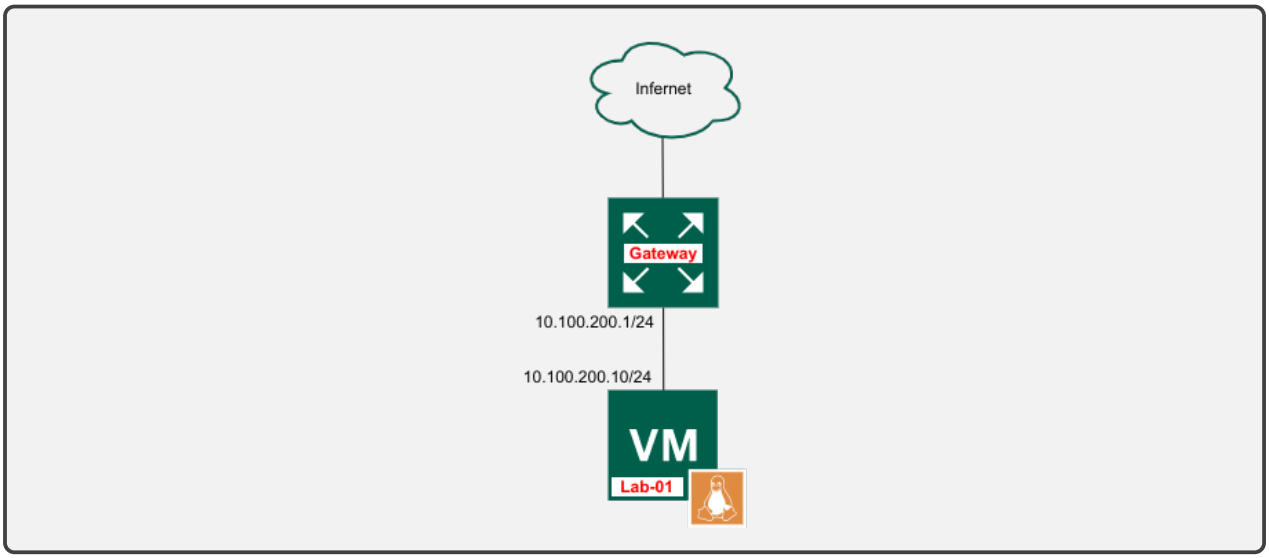
Cuidados antes do laboratório

Combine o último octeto, representado por **X**, para evitar IP duplicado. Não desabilite firewall ou serviços de rede sem necessidade. Primeiro valide link, IP, rota e gateway. Firewall será tratado em outro momento do curso.

9.12.1 Prática 01: configuração básica de uma interface

Configure a interface principal da máquina Lab-01:

IP:	10.100.200.X
Prefixo:	24
Gateway:	10.100.200.1



Exemplo, ajustando o nome da interface ao seu ambiente:

```
ip link set dev enp0s3 up
ip address add 10.100.200.X/24 dev enp0s3
ip route add default via 10.100.200.1
```

Objetivo: configurar interface e rota. Para verificação, execute:

```
ip address show dev enp0s3
ip route show
ping -c 4 10.100.200.1
```

Não deve haver perda de pacotes no teste ICMP para o gateway.

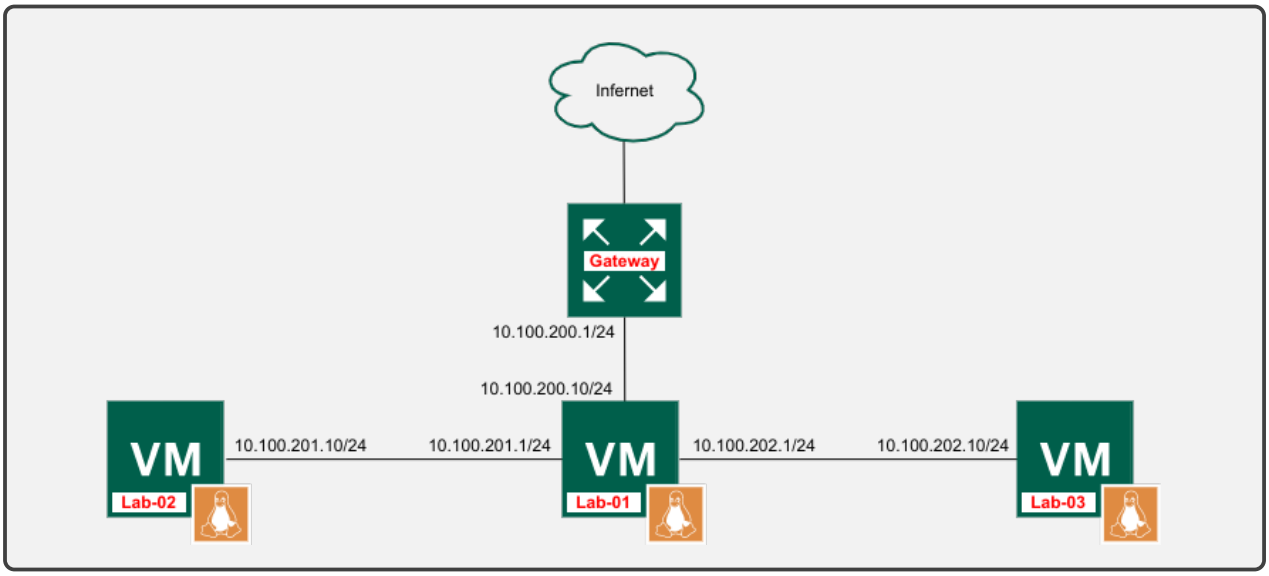
9.12.2 Prática 02: redes diretamente conectadas

Configure a máquina Lab-01 com três interfaces:
Configure a máquina Lab-02:
Configure a máquina Lab-03:

Interface 1:
IP: 10.100.200.X
Prefixo: 24
Gateway: 10.100.200.1
Interface 2:
IP: 10.100.201.1
Prefixo: 24
Interface 3:
IP: 10.100.202.1
Prefixo: 24

IP: 10.100.201.10
Prefixo: 24
Gateway: 10.100.201.1

IP: 10.100.202.10
Prefixo: 24
Gateway: 10.100.202.1



Objetivo: configurar IPs e rotas nas máquinas. Teste a conectividade entre cada host e seu gateway. Depois consulte:

```
ip route show
ip neigh show
```

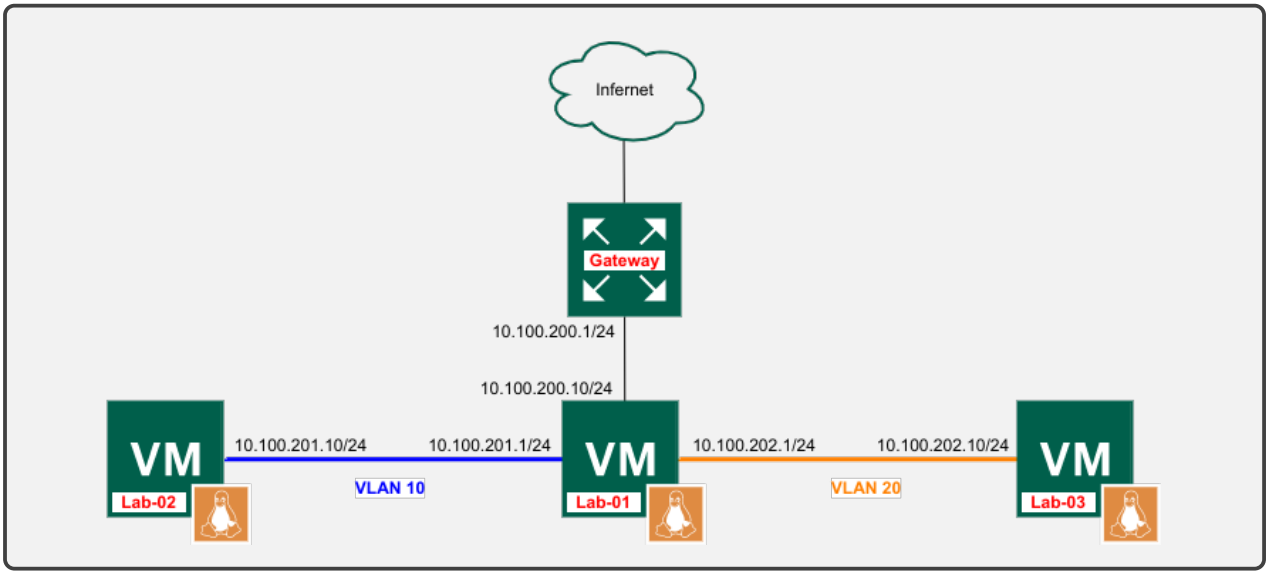
9.12.3 Prática 03: VLANs em laboratório

Configure a máquina Lab-01 com:
Configure a máquina Lab-02:
Configure a máquina Lab-03:

Interface 1:
IP: 10.100.200.X
Prefixo: 24
Gateway: 10.100.200.1
Interface 2:
IP: 10.100.201.1
Prefixo: 24
VLAN: 10
Interface 3:
IP: 10.100.202.1
Prefixo: 24
VLAN: 20

IP: 10.100.201.10
Prefixo: 24
Gateway: 10.100.201.1
VLAN: 10

IP: 10.100.202.10
Prefixo: 24
Gateway: 10.100.202.1
VLAN: 20



Exemplo de criação temporária de uma VLAN:

```
ip link add link enp0s8 name enp0s8.10 type vlan id 10
ip link set dev enp0s8.10 up
ip address add 10.100.201.1/24 dev enp0s8.10
```

Objetivo: configurar interfaces, VLANs e rotas nas máquinas. Para verificação, teste a conectividade entre hosts da mesma VLAN e consulte a tabela de vizinhança.

9.12.4 Prática 04: roteamento entre redes

Utilizando a estrutura da Prática 03, configure a máquina Lab-01 para rotear o tráfego de rede entre as interfaces:

```
sysctl -w net.ipv4.ip_forward=1
```

Para tornar o encaminhamento persistente em laboratório:

```
echo "net.ipv4.ip_forward=1" >> /etc/sysctl.conf
sysctl -p
```

Objetivo: permitir comunicação entre Lab-02 e Lab-03 passando pela Lab-01. Para verificação, execute testes de conectividade entre as máquinas e confirme as rotas configuradas.

Limite do escopo

NAT, mascaramento e regras de firewall não são tratados neste capítulo. Esses temas dependem de conceitos adicionais e serão estudados em etapa própria. Neste momento, o foco é link, IP, rota, gateway e diagnóstico básico.

9.13 Resumo

Neste capítulo foram apresentados os fundamentos de rede necessários para operar uma máquina Linux em ambiente real. O comando ***ip*** foi usado para consultar interfaces, endereços, rotas e vizinhos. Também foi reforçada a diferença entre configuração temporária e persistente, ponto essencial para evitar que uma correção feita em laboratório desapareça após o boot.

9.14 Perguntas de revisão

1. Qual é a diferença entre endereço IP, máscara e gateway?
2. Por que uma máquina pode acessar a rede local, mas não acessar a Internet?
3. O que significa uma rota **default**?
4. Qual comando mostra a tabela de roteamento atual?
5. Qual comando mostra a tabela ARP/NDISC no Linux moderno?
6. Por que comandos como ***ifconfig*** e ***route*** devem ser tratados como legados?
7. Qual é a diferença entre configurar IP com ***ip address add*** e configurar IP no Netplan ou NetworkManager?

9.15 Desafio opcional

Monte duas máquinas virtuais em uma rede interna sem DHCP. Configure endereços IP manuais, teste conectividade entre elas, remova o endereço de uma das máquinas e explique, com base na saída dos comandos ***ip address***, ***ip route*** e ***ip neigh***, por que a comunicação parou de funcionar.

10 Níveis de execução

Nas versões antigas do Linux, a inicialização seguia o modelo SysV init, que organizava o sistema em níveis de execução (*runlevels*). Cada nível definia quais serviços seriam iniciados e qual seria o estado esperado da máquina. Em distribuições atuais, esse modelo foi substituído pelo *systemd*, que trabalha com unidades de destino (*targets*).

Os *runlevels* ainda aparecem como contexto histórico. Em sistemas com *systemd*, eles são mapeados para *targets* equivalentes, o que ajuda na migração de quem aprendeu administração Linux em ambientes mais antigos.

Observação

Em distribuições modernas, o comando abaixo mostra os *targets* de compatibilidade ainda instalados:

```
# find / -name "runlevel*.target"
/usr/lib/systemd/system/runlevel5.target
/usr/lib/systemd/system/runlevel0.target
/usr/lib/systemd/system/runlevel6.target
/usr/lib/systemd/system/runlevel1.target
/usr/lib/systemd/system/runlevel2.target
/usr/lib/systemd/system/runlevel3.target
/usr/lib/systemd/system/runlevel4.target
```

Comparação de níveis de execução de SysV e unidades de destino

Run level	Target units	Descrição
0	`runlevel0.target`, `poweroff.target`	Desliga o sistema
1	`runlevel1.target`, `rescue.target`	Modo de resgate, com acesso mínimo
2, 3, 4	`runlevel[234].target`, `multi-user.target`	Modo multiusuário, sem interface gráfica obrigatória
5	`runlevel5.target`, `graphical.target`	Modo multiusuário com interface gráfica
6	`runlevel6.target`, `reboot.target`	Reinicia o sistema

Os arquivos `runlevel[0123456].target` são links simbólicos para os *targets* atuais do *systemd*. A verificação abaixo ajuda o aluno a entender essa compatibilidade:

```
# cd /usr/lib/systemd/system
# ls -l runlevel*
lrwxrwxrwx. 1 root root 15 Set 23 19:52 runlevel0.target -> poweroff.target
lrwxrwxrwx. 1 root root 13 Set 23 19:52 runlevel1.target -> rescue.target
lrwxrwxrwx. 1 root root 17 Set 23 19:52 runlevel2.target -> multi-user.target
lrwxrwxrwx. 1 root root 17 Set 23 19:52 runlevel3.target -> multi-user.target
lrwxrwxrwx. 1 root root 17 Set 23 19:52 runlevel4.target -> multi-user.target
lrwxrwxrwx. 1 root root 16 Set 23 19:52 runlevel5.target -> graphical.target
lrwxrwxrwx. 1 root root 13 Set 23 19:52 runlevel6.target -> reboot.target
```

10.1 systemd e systemctl

O comando `systemctl` é a principal ferramenta administrativa do *systemd*. Ele permite consultar o estado do sistema, controlar serviços e alterar o *target* padrão.

10.1.1 Verificar o destino padrão

```
# systemctl get-default
graphical.target
```

O `graphical.target` indica que o sistema foi configurado para subir em modo multiusuário com interface gráfica. Em termos históricos, isso se aproxima do antigo nível de execução 5.

O *target* padrão é representado por `/etc/systemd/system/default.target`, que normalmente é um link simbólico para o destino selecionado:

```
# ls -l /etc/systemd/system/default.target
lrwxrwxrwx. 1 root root 41 Set 24 11:58 /etc/systemd/system/default.target ->
/usr/lib/systemd/system/multi-user.target
```

10.1.2 Listar *targets* ativos

Para visualizar os *targets* atualmente carregados no sistema, use:

```
# systemctl list-units --type target
```

Essa saída ajuda a perceber quais etapas da inicialização já foram concluídas. O administrador pode usar essa informação para diagnosticar uma máquina que subiu em modo de recuperação ou sem determinados serviços.

10.1.3 Alterar o destino padrão

Para mudar o *target* padrão, use `set-default`. O exemplo abaixo altera o sistema para o modo multiusuário:

```
# systemctl set-default multi-user.target
```

Esse comando altera apenas o próximo boot. Ele não muda o estado atual da máquina.

10.1.4 Alterar o estado atual

Para mudar imediatamente o estado do sistema em execução, use `isolate`. O exemplo abaixo leva o sistema para o modo multiusuário:

```
# systemctl isolate multi-user.target
```

Em ambientes SysV, esse comportamento era obtido com:

```
# init 3
```

Em sistemas atuais, `init 3` continua funcionando como compatibilidade, mas o comando recomendado é `systemctl isolate multi-user.target`.

10.1.5 Controlar serviços

Além de *targets*, o *systemd* controla serviços. O administrador precisa distinguir dois conceitos:

- **ativo**: o serviço está em execução agora;
- **habilitado**: o serviço será iniciado no boot.

Exemplo com um serviço já instalado no sistema:

```
# systemctl status nome-do-servico.service
# systemctl start nome-do-servico.service
# systemctl stop nome-do-servico.service
# systemctl restart nome-do-servico.service
# systemctl reload nome-do-servico.service
# systemctl enable nome-do-servico.service
# systemctl disable nome-do-servico.service
# systemctl is-enabled nome-do-servico.service
# systemctl list-unit-files --type service
```

O comando `status` mostra se a unidade está carregada, ativa e se houve falhas recentes. Já `enable` e `disable` tratam do comportamento no boot, não da execução imediata.

10.1.6 Diagnóstico com journalctl

Em sistemas com *systemd*, o diagnóstico inicial pode ser feito com o `journalctl`. Um exemplo útil é consultar os registros de um serviço:

```
# journalctl -u nome-do-servico.service
# journalctl -u nome-do-servico.service --since today
# journalctl -xeu nome-do-servico.service
```

Esse comando é importante quando o serviço não sobe, sobe parcialmente ou apresenta mensagens de erro que não aparecem no `status`.

10.2 Desligamento

10.2.1 Shutdown

O comando `shutdown` agenda o desligamento do sistema. Ele pode desligar, travar ou reiniciar a máquina, conforme a opção utilizada.

O primeiro argumento pode indicar o momento da execução, normalmente `now` ou um horário no formato `hh:mm`. Também é possível informar uma mensagem aos usuários conectados.

Importante: quando há um agendamento, o arquivo `/run/nologin` é criado antes do desligamento para impedir novos logins.

```
# shutdown
# shutdown now
# shutdown 13:20
# shutdown 13:20 "Manutenção programada"
# shutdown -p now
# shutdown -H now
```

Opcionalmente, pode-se usar o comando `init` para compatibilidade com ambientes antigos:

```
# init 0
```

10.2.2 Halt

O comando `halt` interrompe a CPU e finaliza as funções do sistema. Em algumas plataformas ele apenas para a máquina; em outras, também provoca o desligamento.

```
# halt
# halt -p
# halt --reboot
```

10.2.3 Power off

O comando `poweroff` solicita o desligamento do sistema.

```
# poweroff
# poweroff --halt
# poweroff --reboot
```

10.2.4 Reboot

O comando `reboot` reinicia o sistema.

```
# reboot
# reboot --halt
# reboot -p
```

Opcionalmente, também é possível usar:

```
# init 6
```

11 Gerenciadores de Pacotes

Gerenciar software em um servidor Linux não significa apenas executar um comando de instalação. O administrador precisa saber de onde o pacote vem, que dependências serão instaladas, que arquivos serão alterados, como consultar o estado do sistema e quais riscos existem ao atualizar componentes em produção.

No início, muitos programas eram distribuídos como código-fonte compactado, normalmente em arquivos *.tar.gz*. O usuário baixava o código, compilava o programa e instalava manualmente os arquivos no sistema. Esse processo dava controle, mas também criava problemas: cada programa podia exigir bibliotecas específicas, versões diferentes de compiladores, caminhos próprios de instalação e procedimentos diferentes de remoção.

Para resolver esse tipo de dificuldade surgiram os pacotes. Um pacote é um arquivo preparado pela distribuição contendo o programa, seus metadados, scripts de instalação, dependências e informações de controle. Em vez de cada administrador compilar tudo manualmente, a distribuição mantém repositórios com pacotes testados para aquela versão do sistema.

Os gerenciadores de pacotes vieram para organizar esse processo. Com eles é possível pesquisar pacotes, instalar programas, remover softwares, atualizar o sistema, validar assinaturas, resolver dependências e manter um histórico mais controlado do que foi instalado.

11.1 Objetivos

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a diferença entre pacote, repositório, cache e dependência;
- usar comandos básicos do APT em Debian e Ubuntu;
- usar comandos básicos do DNF em Rocky Linux, Red Hat Enterprise Linux e derivados;
- reconhecer o YUM como ferramenta histórica e compatível em algumas distribuições;
- diferenciar atualização de lista de pacotes, atualização pontual e atualização ampla do sistema;
- executar consultas e diagnósticos simples usando *dpkg* e *rpm*;
- identificar riscos de misturar repositórios sem critério técnico.

11.2 Pré-requisitos

Para acompanhar este capítulo, o aluno deve saber:

- navegar pelo sistema de arquivos;
- executar comandos no shell;
- usar privilégios administrativos com *root* ou *sudo*;
- compreender a diferença entre arquivo, diretório e permissão básica.

11.3 Famílias de distribuições e formatos de pacote

As distribuições Linux não usam todas o mesmo formato de pacote. No contexto deste módulo, as duas famílias mais importantes são:

- **Debian/Ubuntu**: usam pacotes *.deb*. A ferramenta de baixo nível é o *dpkg*. A ferramenta de uso diário para resolver dependências e trabalhar com repositórios é o *apt*.
- **Red Hat/Rocky/Alma/Fedora**: usam pacotes *.rpm*. A ferramenta de baixo nível é o *rpm*. A ferramenta atual de uso diário é o *dnf*. O *yum* foi muito usado historicamente e ainda pode existir como comando compatível em algumas versões.

Outras distribuições possuem ferramentas próprias, como *zypper* no SUSE, *pacman* no Arch Linux e formatos autocontidos como *snap* e *flatpak*. Eles são importantes em seus contextos, mas o foco deste capítulo é a administração básica com APT e DNF, pois cobrem as famílias mais comuns em ambientes corporativos.

11.4 Conceitos importantes

Antes de decorar comandos, é importante entender os elementos envolvidos.

- **Pacote**: arquivo que contém o software e os metadados necessários para instalação, remoção e consulta.
- **Repositório**: local, normalmente remoto, onde a distribuição publica pacotes organizados por versão, arquitetura e categoria.
- **Cache local**: cópia local de metadados ou pacotes baixados. O cache evita consultas desnecessárias à rede e acelera operações repetidas.
- **Dependência**: outro pacote necessário para que o programa instalado funcione corretamente.
- **Assinatura**: mecanismo criptográfico usado para validar que o pacote veio de uma fonte confiável e não foi alterado no caminho.

Em servidores, a origem do pacote é tão importante quanto o comando usado para instalá-lo. Misturar repositórios de versões diferentes, usar repositórios de terceiros sem validação ou instalar pacotes baixados manualmente pode quebrar dependências e dificultar futuras atualizações.

11.5 Configurações básicas

11.5.1 APT

No Debian e no Ubuntu, as configurações do APT ficam no diretório */etc/apt*. As fontes de repositório podem estar no arquivo */etc/apt/sources.list* ou em arquivos dentro de */etc/apt/sources.list.d/*.

Uma entrada tradicional de repositório pode ser representada assim:

```
deb http://host/debian distribuicao secao1 secao2 secao3
deb-src http://host/debian distribuicao secao1 secao2 secao3
```

As linhas acima são apenas exemplos. A palavra *deb* indica um repositório de pacotes binários, que são os pacotes normalmente instalados no sistema. A palavra *deb-src* indica um repositório de pacotes fonte, usado quando se deseja consultar ou reconstruir o código-fonte do pacote.

Em versões atuais do Debian e do Ubuntu também é comum encontrar arquivos no formato *.sources*. O princípio é o mesmo: o sistema precisa saber quais repositórios deve consultar e quais chaves usar para validar os pacotes.

11.5.2 DNF e YUM

Nas distribuições da família Red Hat, como Rocky Linux 9, AlmaLinux 9, Fedora e Red Hat Enterprise Linux 9, a ferramenta atual de uso diário é o *dnf*. As configurações principais ficam associadas aos arquivos de repositório em */etc/yum.repos.d/*, normalmente com extensão *.repo*.

Um arquivo de repositório possui uma estrutura semelhante a esta:

```
[baseos]
name=Rocky Linux $releasever - BaseOS
baseurl=https://mirror.example.org/rocky/$releasever/BaseOS/$basearch/os/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-rockyofficial
```

A primeira linha identifica o repositório. A opção *name* define um nome descritivo. A opção *baseurl* aponta para o local onde os pacotes estão publicados. A opção *enabled* informa se o

repositório está ativo. A opção *gpgcheck* exige validação de assinatura, prática recomendada em ambientes administrados. A opção *gpgkey* informa a chave usada nessa validação.

O comando *yum* ainda aparece em muitos materiais e sistemas antigos. Em distribuições atuais, ele pode ser apenas um atalho compatível para o *dnf*. Por isso, neste capítulo, os exemplos principais usam *dnf*; quando aparecer *yum*, ele deve ser entendido como contexto histórico ou compatibilidade.

11.6 Atualizando a lista de pacotes

Os repositórios mudam com o tempo. Novas versões são publicadas, correções de segurança aparecem e pacotes podem ser substituídos. Antes de instalar ou atualizar pacotes no Debian e no Ubuntu, o administrador deve atualizar a lista local de pacotes disponíveis:

```
apt update
```

Esse comando não atualiza os programas instalados. Ele apenas atualiza a visão local do APT sobre os repositórios configurados.

No DNF, a atualização dos metadados costuma ocorrer automaticamente quando necessário, mas é possível forçar a criação ou atualização do cache:

```
dnf makecache
```

Em sistemas antigos ou compatíveis com YUM:

```
yum makecache
```

11.7 Procurando pacotes e consultando informações

Para procurar pacotes no Debian ou Ubuntu:

```
apt search <PACOTE>
```

Para exibir informações mais detalhadas sobre um pacote:

```
apt show <PACOTE>
```

Em distribuições Red Hat-like atuais:

```
dnf search <PACOTE>
dnf info <PACOTE>
```

Esses comandos devem ser usados antes da instalação quando o administrador ainda não tem certeza do nome do pacote ou precisa confirmar versão, descrição, repositório de origem e dependências principais.

11.8 Instalando pacotes

Para instalar pacotes no Debian ou Ubuntu:

```
apt install <PACOTE>
```

Para instalar pacotes em Rocky Linux, RHEL, AlmaLinux ou Fedora:

```
dnf install <PACOTE>
```

Em ambos os casos, o gerenciador calcula as dependências necessárias e apresenta um resumo antes da instalação. O administrador deve ler esse resumo. Em servidores, uma instalação aparentemente simples pode trazer bibliotecas, serviços, usuários de sistema ou alterações de configuração.

11.9 Removendo pacotes

No Debian e no Ubuntu, existem duas remoções comuns:

```
apt remove <PACOTE>
apt purge <PACOTE>
```

O *remove* remove o pacote, mas pode preservar arquivos de configuração. O *purge* remove o pacote e também os arquivos de configuração gerenciados pelo pacote. Em um servidor, essa diferença é importante. Remover um serviço preservando sua configuração pode ser útil quando ele será reinstalado depois. Usar *purge* é mais adequado quando se deseja limpar a instalação.

Em distribuições Red Hat-like:

```
dnf remove <PACOTE>
```

Antes de confirmar a remoção, verifique se outros pacotes importantes serão removidos por dependência.

11.10 Atualizando pacotes instalados

No Debian e no Ubuntu, o fluxo básico é:

```
apt update
apt upgrade
```

O primeiro comando atualiza a lista local de pacotes disponíveis. O segundo atualiza os pacotes instalados quando há versões novas compatíveis com a versão atual da distribuição.

Em distribuições Red Hat-like:

```
dnf upgrade
```

O comando abaixo também é encontrado em muitos ambientes:

```
dnf update
```

No DNF atual, *update* é tratado como equivalente a *upgrade*. Mesmo assim, é recomendável padronizar o uso de *dnf upgrade* em materiais novos.

11.11 Atualização pontual e atualização ampla

Atualizar um pacote específico é diferente de atualizar todo o sistema. Para atualizar apenas um pacote no Debian ou Ubuntu:

```
apt install --only-upgrade <PACOTE>
```

Em distribuições Red Hat-like:

```
dnf upgrade <PACOTE>
```

Já a atualização ampla do sistema envolve todos os pacotes instalados que têm novas versões disponíveis. Em servidores de produção, esse tipo de atualização deve ser planejado, testado e preferencialmente executado em janela de manutenção.

No Debian e no Ubuntu, também existe:

```
apt full-upgrade
```

O *full-upgrade* pode instalar novos pacotes ou remover pacotes existentes para resolver mudanças de dependência. Ele não deve ser tratado como um comando rotineiro sem leitura cuidadosa do que será alterado.

Atualizar de uma versão da distribuição para outra, por exemplo de uma versão do Ubuntu LTS para a próxima, é outro procedimento. Esse processo costuma envolver ferramentas e documentação próprias da distribuição e não deve ser confundido com a atualização comum de pacotes.

11.12 Limpando cache e pacotes não utilizados

Quando pacotes são baixados, metadados e arquivos podem ficar armazenados em cache. No Debian e no Ubuntu, é comum usar:

```
apt clean
apt autoclean
apt autoremove
```

O *clean* remove arquivos baixados do cache. O *autoclean* remove apenas arquivos antigos que não são mais úteis. O *autoremove* remove pacotes instalados automaticamente como dependências e que não são mais necessários.

Em distribuições Red Hat-like:

```
dnf clean all
dnf autoremove
```

Esses comandos ajudam a liberar espaço e corrigir situações em que metadados locais ficam inconsistentes. Ainda assim, *autoremove* deve ser usado com atenção, sempre lendo a lista de pacotes que será removida.

11.13 Cuidados com repositórios

Repositórios facilitam a administração, mas também podem ser fonte de problemas. Alguns cuidados básicos:

- não misturar repositórios de versões diferentes da distribuição;
- evitar repositórios de terceiros sem necessidade clara;
- validar a origem da chave GPG;
- registrar alterações feitas em arquivos de repositório;
- testar atualizações relevantes em ambiente de homologação quando possível;
- evitar baixar pacotes avulsos da internet para instalar manualmente em servidores.

Em caso de dúvida, prefira os repositórios oficiais da distribuição. Quando um repositório de terceiro for realmente necessário, documente o motivo, a origem e o impacto esperado.

11.14 Manuseando pacotes Debian com dpkg

O *dpkg* é a ferramenta de baixo nível para lidar com pacotes em sistemas Debian e Ubuntu. Ele sabe o que está instalado no sistema e consegue instalar ou remover arquivos *.deb* locais. Porém, o *dpkg* não consulta repositórios nem resolve dependências sozinho como o APT.

Por isso, no uso diário, prefira *apt*. Use *dpkg* para consultas locais, diagnóstico ou situações específicas envolvendo arquivos *.deb*.

11.14.1 Instalando um arquivo .deb local

```
dpkg -i <ARQUIVO.deb>
```

Se houver dependências ausentes, pode ser necessário corrigir a instalação com:

```
apt install -f
```

11.14.2 Removendo pacote

```
dpkg -r <PACOTE>
```

11.14.3 Consultando arquivos de um pacote instalado

```
dpkg -L <PACOTE>
```

Exemplo:

```
dpkg -L base-passwd
```

11.14.4 Descobrindo qual pacote instalou um arquivo

```
dpkg -S <CAMINHO>
```

Exemplo:

```
dpkg -S /bin/date
```

11.14.5 Exibindo informações de um pacote instalado

```
dpkg -s <PACOTE>
```

11.14.6 Listando pacotes instalados

```
dpkg -l
dpkg -l <PACOTE>
```

11.14.7 Listando arquivos dentro de um pacote .deb

```
dpkg -c <ARQUIVO.deb>
```

11.15 Manuseando pacotes Red Hat com rpm

O *rpm* é a ferramenta de baixo nível para pacotes em distribuições da família Red Hat. Assim como o *dpkg*, ele não deve ser confundido com o gerenciador de repositórios. O *rpm* instala, remove e consulta pacotes locais, mas não resolve dependências de repositório como o DNF.

No uso diário, prefira *dnf*. Use *rpm* para consulta local, diagnóstico e situações específicas envolvendo arquivos *.rpm*.

11.15.1 Instalando um arquivo .rpm local

```
rpm -i <ARQUIVO.rpm>
```

Em muitos casos, é melhor instalar um arquivo local com DNF, pois ele tenta resolver dependências nos repositórios configurados:

```
dnf install ./<ARQUIVO.rpm>
```

11.15.2 Atualizando pacote local

```
rpm -U <ARQUIVO.rpm>
```

11.15.3 Removendo pacote

```
rpm -e <PACOTE>
```

11.15.4 Consultando arquivos de um pacote instalado

```
rpm -ql <PACOTE>
```

11.15.5 Descobrindo qual pacote instalou um arquivo

```
rpm -qf <CAMINHO>
```

Exemplo:

```
rpm -qf /etc/passwd
```

11.15.6 Listando pacotes instalados

```
rpm -qa
```

11.15.7 Exibindo informações de um pacote

```
rpm -qi <PACOTE>
```

11.16 Diagnóstico básico

Alguns problemas comuns em gerenciamento de pacotes:

- falha de rede ao acessar o repositório;
- repositório desabilitado;
- chave GPG ausente ou inválida;
- pacote inexistente naquela versão da distribuição;
- dependência quebrada;
- conflito entre pacotes de repositórios diferentes;
- cache local desatualizado ou corrompido.

Em Debian e Ubuntu, comandos úteis para iniciar o diagnóstico:

```
apt update
apt show <PACOTE>
apt policy <PACOTE>
dpkg -l <PACOTE>
```

Em Rocky Linux, RHEL e derivados:

```
dnf repolist
dnf info <PACOTE>
dnf provides <CAMINHO>
rpm -qa | grep <PACOTE>
```

Ao diagnosticar, observe sempre a mensagem de erro completa. Ela normalmente indica se o problema é rede, repositório, chave, dependência ou conflito.

11.17 Laboratório guiado

11.17.1 Objetivo do laboratório

Instalar, consultar, remover e limpar pacotes usando uma VM Debian ou Ubuntu. Ao final, comparar conceitualmente os comandos equivalentes em Rocky Linux.

11.17.2 Ambiente

- uma máquina virtual Debian 12 ou Ubuntu 24.04 LTS;
- acesso administrativo com *sudo*;
- conectividade com a internet;
- preferencialmente um snapshot da VM antes do laboratório.

11.17.3 Etapas em Debian ou Ubuntu

Atualize a lista local de pacotes:

```
sudo apt update
```

Pesquise um pacote simples:

```
apt search tree
apt show tree
```

Instale o pacote:

```
sudo apt install tree
```

Teste o comando instalado:

```
tree /etc/apt
```

Descubra a qual pacote pertence o programa:

```
dpkg -S /usr/bin/tree
```

Liste os arquivos instalados pelo pacote:

```
dpkg -L tree
```

Remova o pacote:

```
sudo apt remove tree
```

Limpe pacotes não utilizados, se houver:

```
sudo apt autoremove
```

11.17.4 Equivalência conceitual em Rocky Linux

Em uma VM Rocky Linux 9, o fluxo equivalente seria:

```
sudo dnf makecache
dnf search tree
dnf info tree
sudo dnf install tree
rpm -qf /usr/bin/tree
rpm -ql tree
sudo dnf remove tree
```

11.18 Exercícios de revisão

1. Explique a diferença entre *apt update* e *apt upgrade*.
2. Em que situação o comando *apt purge* pode ser mais adequado que *apt remove*?
3. Por que não é recomendável misturar repositórios de distribuições ou versões diferentes?
4. Qual é a diferença de responsabilidade entre *dnf* e *rpm*?
5. Um pacote não foi encontrado pelo gerenciador. Quais verificações você faria antes de concluir que ele não existe?

11.19 Resumo

Gerenciadores de pacotes são parte central da administração Linux. Eles reduzem o trabalho manual, resolvem dependências, validam pacotes e permitem manter o sistema atualizado de forma controlada. No Debian e no Ubuntu, a ferramenta principal é o *apt*; em Rocky Linux, RHEL e derivados atuais, a ferramenta principal é o *dnf*. As ferramentas *dpkg* e *rpm* continuam importantes para consultas e diagnósticos locais, mas não substituem o gerenciador de repositórios no uso diário.

Em ambientes reais, o cuidado mais importante é tratar instalação e atualização como atividade administrativa controlada. Antes de confirmar alterações, leia o resumo do gerenciador, confira os repositórios envolvidos e avalie o impacto no serviço.

12 Logs do Sistema

Logs são registros produzidos pelo sistema operacional, por serviços e por aplicações. Eles mostram eventos que ocorreram na máquina: inicialização, autenticação de usuários, falhas de serviços, mensagens do kernel, erros de permissão, tentativas de acesso e muitas outras situações. Para um administrador Linux, saber ler logs é tão importante quanto saber executar comandos.

Quando um serviço não sobe, quando um usuário não consegue autenticar ou quando um processo termina sem explicação aparente, o log costuma ser o primeiro lugar onde o diagnóstico começa. O objetivo deste capítulo é apresentar os registros do Linux como uma ferramenta prática de investigação para o dia a dia de um SysAdmin Júnior.

Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a finalidade dos logs em sistemas Linux;
- localizar arquivos importantes em `/var/log`;
- diferenciar logs de sistema, autenticação, kernel e serviços;
- consultar eventos com `journalctl`;
- acompanhar logs em tempo real com `tail -f`;
- filtrar mensagens usando `grep`, `less` e expressões regulares simples;
- aplicar uma sequência básica de diagnóstico para falhas de serviço, login, permissão e disco cheio.

Pré-requisitos

Para acompanhar este capítulo, é recomendável conhecer comandos básicos do shell, navegação no sistema de arquivos, pipes, redirecionamentos, permissões e noções iniciais de processos e serviços. Os exemplos devem ser executados em uma máquina de laboratório, preferencialmente Debian 12, Ubuntu 24.04 LTS ou Rocky Linux 9.

12.1 Conceito de log

Um log é uma linha de registro sobre algo que aconteceu no sistema. Em geral, essa linha informa uma data, um horário, o nome da máquina, o programa que gerou o evento e a mensagem registrada.

```
Jun 11 10:15:42 servidor01 sshd[2145]: Failed password for aluno
from 192.168.10.50 port 51234 ssh2
```

No exemplo, é possível identificar:

- quando o evento ocorreu;
- em qual servidor ele ocorreu;
- qual serviço gerou a mensagem;
- qual foi o problema registrado.

Essa informação não resolve o problema sozinha, mas orienta a investigação. Em vez de tentar adivinhar a causa, o administrador passa a procurar evidências.

12.2 Onde os logs ficam

Historicamente, muitos logs do Linux ficam em `/var/log`. Esse diretório contém arquivos de texto, subdiretórios de serviços e arquivos rotacionados de dias anteriores.

```
$ ls -lh /var/log
```

Arquivos comuns nesse diretório:

Arquivo ou diretório	**Finalidade**
`/var/log/syslog`	Mensagens gerais do sistema em Debian e Ubuntu
`/var/log/messages`	Mensagens gerais do sistema em Rocky Linux e derivados RHEL
`/var/log/auth.log`	Eventos de autenticação em Debian e Ubuntu
`/var/log/secure`	Eventos de autenticação em Rocky Linux e derivados RHEL
`/var/log/kern.log`	Mensagens do kernel em algumas distribuições
`/var/log/dmesg`	Mensagens do kernel registradas durante a inicialização
`/var/log/apt/`	Histórico e registros do APT em Debian e Ubuntu
`/var/log/dnf.log`	Registro do DNF em Rocky Linux e derivados RHEL

A existência e o nome exato de cada arquivo variam conforme a distribuição e a configuração do sistema. Em servidores com *systemd*, parte importante dos eventos também fica disponível pelo `journalctl`.

12.3 Logs clássicos e o syslog

O modelo clássico de logs em Linux usa um serviço de syslog para receber mensagens e gravá-las em arquivos. Em muitas distribuições esse papel é realizado pelo `rsyslog`.

```
$ systemctl status rsyslog
```

O `rsyslog` pode registrar mensagens locais, receber logs de outros equipamentos e separar eventos por origem e severidade. Para o Módulo 1, o mais importante é entender que ele é um dos responsáveis por popular arquivos como `/var/log/syslog`, `/var/log/messages`, `/var/log/auth.log` e `/var/log/secure`.

> Em servidores antigos, a investigação de logs era quase sempre baseada apenas nos arquivos em `/var/log`. Em distribuições modernas, o administrador deve conhecer também o `journalctl`, pois muitos serviços integrados ao *systemd* registram eventos no journal.

12.4 systemd-journald e journalctl

O `systemd-journald` é o componente do *systemd* responsável por coletar logs do sistema, do kernel, de serviços e de unidades gerenciadas pelo *systemd*. O comando usado para consultar esses registros é o `journalctl`.

```
$ journalctl
```

Sem opções, o comando mostra muitos registros. Por isso, no trabalho diário, é comum filtrar por período, serviço, prioridade ou inicialização.

12.4.1 Consultar logs de um serviço

Para investigar um serviço específico:

```
$ journalctl -u ssh.service
$ journalctl -u sshd.service
```

Em Debian e Ubuntu, o serviço costuma aparecer como `ssh.service`. Em Rocky Linux e derivados RHEL, é comum encontrar `sshd.service`. Essa diferença de nome é normal e deve ser observada durante o diagnóstico.

```
$ systemctl status ssh
$ systemctl status sshd
```

12.4.2 Filtrar por período

Para consultar eventos recentes:

```
$ journalctl --since today
$ journalctl --since "1 hour ago"
$ journalctl --since "2026-06-11 08:00" --until "2026-06-11 12:00"
```

O filtro por período reduz ruído. Em uma ocorrência real, normalmente o administrador sabe uma janela aproximada: "o erro começou depois do reboot", "o usuário tentou acessar às 10h" ou "o serviço caiu durante a madrugada".

12.4.3 Consultar a inicialização atual

Para ver registros do boot atual:

```
$ journalctl -b
```

Para listar boots conhecidos pelo journal:

```
$ journalctl --list-boots
```

Quando o problema ocorre logo após reiniciar a máquina, `journalctl -b` é uma das consultas mais úteis.

12.4.4 Acompanhar eventos em tempo real

Para acompanhar o journal continuamente:

```
$ journalctl -f
```

Para acompanhar apenas um serviço:

```
$ journalctl -u ssh.service -f
```

Esse modo é útil durante testes controlados. Por exemplo: abrir um terminal acompanhando o log e, em outro, tentar iniciar um serviço ou realizar uma autenticação.

12.5 Lendo arquivos de log

Arquivos de log podem ser lidos com ferramentas já conhecidas do shell:

```
$ less /var/log/syslog
$ tail /var/log/syslog
$ tail -f /var/log/syslog
$ grep -i error /var/log/syslog
```

Em Rocky Linux, o arquivo equivalente para mensagens gerais costuma ser:

```
$ less /var/log/messages
```

Para autenticação:

```
$ sudo less /var/log/auth.log
$ sudo less /var/log/secure
```

Alguns arquivos exigem permissão de administrador porque podem conter dados sensíveis, como nomes de usuários, origem de acessos, comandos executados com `sudo` e falhas de autenticação.

12.6 Tipos comuns de eventos

Nem todo log tem a mesma finalidade. Em diagnóstico, é importante procurar no lugar certo.

12.6.1 Sistema

Mensagens gerais do sistema incluem inicialização, serviços, unidades do *systemd*, tarefas periódicas e eventos de componentes internos.

```
$ journalctl --since today
$ grep -i failed /var/log/syslog
```

12.6.2 Autenticação

Logs de autenticação ajudam a investigar login, `sudo`, SSH, bloqueios e tentativas de acesso inválidas.

```
$ sudo grep -i "failed password" /var/log/auth.log
$ sudo grep -i "sudo" /var/log/auth.log
```

Em Rocky Linux:

```
$ sudo grep -i "failed password" /var/log/secure
$ sudo grep -i "sudo" /var/log/secure
```

12.6.3 Kernel

Mensagens do kernel são importantes para problemas de hardware, disco, módulos, interfaces de rede e sistemas de arquivos.

```
$ dmesg | less
$ journalctl -k
```

O `dmesg` mostra o buffer de mensagens do kernel. O `journalctl -k` consulta mensagens de kernel registradas no journal.

12.6.4 Serviços

Serviços gerenciados pelo *systemd* devem ser investigados com `systemctl status` e `journalctl -u`.

```
$ systemctl status nome-do-servico.service
$ journalctl -u nome-do-servico.service --since today
```

O `status` mostra um resumo. O `journalctl` permite aprofundar a consulta.

12.7 Rotação de logs

Logs crescem com o tempo. Para evitar que ocupem todo o disco, sistemas Linux usam mecanismos de rotação. O mais comum é o `logrotate`.

Quando um arquivo é rotacionado, ele pode ser renomeado, compactado e substituído por um arquivo novo. Por isso, é comum encontrar arquivos como:

```
/var/log/syslog
/var/log/syslog.1
/var/log/syslog.2.gz
```

Para ler um arquivo compactado:

```
$ zless /var/log/syslog.2.gz
$ zgrep -i error /var/log/syslog.2.gz
```

A rotação não elimina a necessidade de monitorar espaço em disco. Se `/var` ficar cheio, serviços podem falhar ao gravar dados temporários, spools, bancos locais ou novos logs.

12.8 Sequência básica de diagnóstico

Uma investigação simples deve seguir uma ordem. Isso evita conclusões apressadas.

12.8.1 Serviço que não inicia

1. verificar o estado do serviço;
2. consultar logs recentes da unidade;
3. procurar mensagens de erro explícitas;
4. validar arquivo de configuração;
5. corrigir a causa e reiniciar o serviço.

```
$ systemctl status nome-do-servico.service
$ journalctl -u nome-do-servico.service --since "30 minutes ago"
$ sudo systemctl restart nome-do-servico.service
```

12.8.2 Falha de login

1. confirmar se o usuário existe;
2. verificar se a senha ou chave está correta;
3. consultar log de autenticação;
4. verificar bloqueios, permissões e origem do acesso.

```
$ id aluno
$ sudo grep -i aluno /var/log/auth.log
$ sudo journalctl _COMM=sshd --since today
```

Em Rocky Linux, substituir `/var/log/auth.log` por `/var/log/secure` quando necessário.

12.8.3 Disco cheio

1. verificar uso dos sistemas de arquivos;
2. identificar diretórios que mais crescem;
3. verificar se logs cresceram de forma anormal;
4. corrigir a causa antes de apagar arquivos.

```
$ df -h
$ sudo du -sh /var/log/*
$ sudo journalctl --disk-usage
```

Atenção: não apague logs de produção sem avaliar necessidade de auditoria, retenção e investigação. Em muitos ambientes, logs são evidência operacional e podem ser exigidos em processos internos de segurança.

12.8.4 Erro de permissão

Erros de permissão podem aparecer no terminal, no log do serviço ou no journal. O diagnóstico deve relacionar usuário, arquivo, diretório e processo envolvido.

```
$ journalctl --since "10 minutes ago" | grep -i denied
$ journalctl --since "10 minutes ago" | grep -i permission
```

Depois de localizar a mensagem, o administrador deve verificar dono, grupo, permissões e contexto do serviço.

12.9 Laboratório guiado

Este laboratório gera eventos simples e mostra como encontrá-los. Execute em ambiente de teste.

Parte 1: gerar mensagem manual no log

O comando `logger` envia uma mensagem para o sistema de logs.

```
$ logger "LAB-LOGS: teste de mensagem do aluno"
```

Localize a mensagem no journal:

```
$ journalctl --since "5 minutes ago" | grep "LAB-LOGS"
```

Em Debian ou Ubuntu, também procure no syslog:

```
$ grep "LAB-LOGS" /var/log/syslog
```

Em Rocky Linux:

```
$ grep "LAB-LOGS" /var/log/messages
```

Parte 2: acompanhar logs em tempo real

Em um terminal, execute:

```
$ journalctl -f
```

Em outro terminal, gere uma nova mensagem:

```
$ logger "LAB-LOGS: evento acompanhado em tempo real"
```

Observe se a mensagem aparece no primeiro terminal.

Parte 3: investigar uma falha controlada de serviço

Escolha um serviço existente no laboratório. Em Debian e Ubuntu, pode ser `ssh`. Em Rocky Linux, pode ser `sshd`, se o pacote estiver instalado.

```
$ systemctl status ssh
$ journalctl -u ssh.service --since today
```

ou:

```
$ systemctl status sshd
$ journalctl -u sshd.service --since today
```

O objetivo não é provocar uma indisponibilidade real, mas aprender onde consultar as mensagens do serviço. Em ambientes compartilhados, não pare serviços sem autorização do instrutor.

12.10 Resumo

Logs registram evidências do comportamento do sistema. Em Linux, o administrador deve conhecer tanto arquivos clássicos em `/var/log` quanto o journal do *systemd*. Para diagnóstico inicial, uma boa prática é combinar `systemctl status`, `journalctl`, `less`, `tail`, `grep`, `df` e `du`.

O ponto mais importante é trabalhar com evidências. Antes de reiniciar serviços, alterar permissões ou apagar arquivos, leia os logs e procure a mensagem que explica o erro.

12.11 Perguntas de revisão

1. Qual é a finalidade de um log em um sistema Linux?
2. Qual é a diferença prática entre consultar `/var/log/syslog` e usar `journalctl`?
3. Em quais arquivos ou comandos você procuraria falhas de autenticação em Debian, Ubuntu e Rocky Linux?
4. Por que `tail -f` e `journalctl -f` são úteis durante um teste de diagnóstico?
5. O que deve ser verificado antes de apagar arquivos antigos de log?

12.12 Desafio opcional

Crie uma mensagem de teste com `logger`, localize essa mensagem no journal e no arquivo clássico de log da sua distribuição. Em seguida, registre em um pequeno relatório:

- comando usado para gerar a mensagem;
- comando usado para localizar a mensagem no journal;
- arquivo de log onde a mensagem apareceu;
- diferença observada entre Debian/Ubuntu e Rocky Linux, se houver.

13 Agendamento de Tarefas com Cron

Em servidores Linux, muitas atividades precisam ocorrer sem intervenção manual: gerar relatórios, executar cópias de segurança, limpar arquivos temporários, sincronizar diretórios, verificar serviços e registrar informações de diagnóstico. Quando essas tarefas são repetitivas, o administrador deve automatizá-las de forma simples, previsível e verificável.

O *cron* é um dos mecanismos tradicionais do Linux para executar comandos em horários definidos. Ele é antigo, estável e ainda muito usado em ambientes de produção. Mesmo em sistemas modernos com *systemd*, conhecer o *cron* continua sendo essencial para um SysAdmin Júnior, pois muitos servidores, scripts legados e rotinas administrativas dependem dele.

Objetivos do capítulo

Ao final deste capítulo, o aluno deverá ser capaz de:

- compreender a finalidade do agendamento de tarefas;
- diferenciar *crontab* de usuário, `/etc/crontab` e diretórios periódicos do sistema;
- interpretar a sintaxe de minuto, hora, dia do mês, mês e dia da semana;
- criar agendamentos simples com `crontab -e`;
- usar caminhos absolutos e redirecionamento de saída em tarefas agendadas;
- consultar logs para verificar se uma tarefa foi executada;
- reconhecer *systemd timers* como alternativa moderna em alguns cenários;
- montar uma rotina simples de backup agendado em laboratório.

Pré-requisitos

Para acompanhar este capítulo, é recomendável conhecer comandos básicos do shell, permissões, redirecionamento de entrada e saída, edição de arquivos de texto e noções iniciais sobre logs do sistema. Os exemplos devem ser executados em uma máquina de laboratório, preferencialmente Debian 12, Ubuntu 24.04 LTS ou Rocky Linux 9.

13.1 Conceito de agendamento

Agendar uma tarefa significa informar ao sistema que determinado comando deve ser executado em um momento específico ou em uma recorrência definida. Em vez de depender da memória do administrador, o próprio sistema passa a executar a rotina no horário configurado.

Exemplos comuns:

- executar um backup todos os dias durante a madrugada;
- limpar arquivos temporários uma vez por semana;
- gerar um relatório no primeiro dia do mês;
- testar a disponibilidade de um serviço a cada cinco minutos;
- sincronizar um diretório com outro servidor em horários definidos.

Automatizar não elimina a responsabilidade do administrador. Toda tarefa agendada precisa ser testada manualmente antes, registrar sua saída de forma adequada e ter algum meio de verificação posterior. Uma automação silenciosa que falha sem deixar evidência pode ser pior do que uma tarefa manual.

13.2 O serviço cron

O *cron* normalmente é executado como um serviço do sistema. O nome exato da unidade pode variar conforme a distribuição.

Em Debian e Ubuntu:

```
$ systemctl status cron
```

Em Rocky Linux e derivados RHEL, é comum encontrar o serviço como `crond`:

```
$ systemctl status crond
```

Se o serviço não estiver em execução, as tarefas agendadas não serão processadas. Por isso, diante de uma rotina que não executou, a primeira verificação deve ser o estado do serviço.

> Em servidores atuais, o *cron* costuma conviver com o *systemd*. Isso não significa que o *cron* esteja obsoleto para todas as situações. Ele continua adequado para agendamentos simples e scripts administrativos pequenos. Para tarefas mais integradas ao gerenciamento de serviços, *systemd timers* podem ser uma alternativa melhor.

13.3 Crontab de usuário

Cada usuário pode possuir sua própria tabela de agendamentos, chamada *crontab*. Para editar o *crontab* do usuário atual:

```
$ crontab -e
```

Para listar os agendamentos do usuário atual:

```
$ crontab -l
```

Para remover o *crontab* do usuário atual:

```
$ crontab -r
```

O comando `crontab -r` deve ser usado com cuidado, pois remove todos os agendamentos daquele usuário. Em ambientes administrados, é recomendável listar e registrar o conteúdo antes de qualquer remoção.

```
$ crontab -l
```

Quando o administrador precisa editar o *crontab* de outro usuário, pode usar a opção `-u`, normalmente com privilégios administrativos:

```
$ sudo crontab -u aluno -e
$ sudo crontab -u aluno -l
```

13.4 Sintaxe do cron

Uma linha comum de *crontab* possui cinco campos de tempo, seguidos pelo comando a ser executado:

```
minuto hora dia_do_mes mes dia_da_semana comando
```

Os campos são:

Campo	**Valores**	**Descrição**
minuto	0–59	Minuto em que o comando será executado
hora	0–23	Hora do dia
dia do mês	1–31	Dia do mês
mês	1–12	Mês do ano
dia da semana	0–7	Dia da semana, normalmente 0 ou 7 para domingo

O asterisco significa “qualquer valor”. Assim, uma tarefa com asterisco no campo de minuto será avaliada em todos os minutos.

13.5 Exemplos de agendamento

Executar um comando todos os dias às 02:30:

```
30 2 * * * /usr/local/bin/rotina.sh
```

Executar de segunda a sexta às 08:00:

```
0 8 * * 1-5 /usr/local/bin/relatorio.sh
```

Executar a cada cinco minutos:

```
*/5 * * * * /usr/local/bin/verifica_servico.sh
```

Executar no primeiro dia de cada mês às 03:00:

```
0 3 1 * * /usr/local/bin/fechamento_mensal.sh
```

Executar aos domingos às 23:00:

```
0 23 * * 0 /usr/local/bin/limpeza_semanal.sh
```

Esses exemplos mostram a ideia geral, mas não dispensam teste. Antes de colocar um script no *cron*, execute-o manualmente com o mesmo usuário que fará o agendamento.

13.6 Atalhos especiais

Algumas implementações de *cron* aceitam atalhos para períodos comuns:

Atalho	**Significado**
`@hourly`	Executa uma vez por hora
`@daily`	Executa uma vez por dia
`@weekly`	Executa uma vez por semana
`@monthly`	Executa uma vez por mês
`@yearly`	Executa uma vez por ano
`@reboot`	Executa após a inicialização do sistema

Exemplo:

```
@daily /usr/local/bin/backup_diario.sh
```

O atalho `@reboot` deve ser usado com critério. Ele executa após a inicialização, mas isso não garante que rede, discos remotos, banco de dados ou outros serviços já estejam prontos no momento exato da execução.

13.7 Crontab do sistema

Além dos *crontabs* de usuários, existe o arquivo `/etc/crontab`. Ele segue uma sintaxe parecida, mas possui um campo adicional indicando qual usuário executará o comando.

```
minuto hora dia_do_mes mes dia_da_semana usuario comando
```

Exemplo:

```
15 4 * * * root /usr/local/sbin/rotina_sistema.sh
```

Como esse arquivo permite definir o usuário de execução, ele é usado para rotinas do próprio sistema ou tarefas administrativas mantidas pelo *root*. Em geral, tarefas pessoais ou de aplicação devem ficar no *crontab* do usuário apropriado.

13.8 Diretórios /etc/cron.*

Muitas distribuições também possuem diretórios para agendamentos periódicos:

- `/etc/cron.hourly`;
- `/etc/cron.daily`;
- `/etc/cron.weekly`;
- `/etc/cron.monthly`.

Scripts colocados nesses diretórios podem ser executados automaticamente pela configuração do sistema, normalmente com apoio de `run-parts`. A periodicidade exata deve ser conferida na própria distribuição, pois pode haver diferenças entre Debian, Ubuntu, Rocky Linux e ambientes personalizados.

```
$ ls -l /etc/cron.daily
$ cat /etc/crontab
```

Esses diretórios são úteis para tarefas simples do sistema, mas exigem atenção a permissões, nome do arquivo e comportamento do script. Um script sem permissão de execução, com nome não aceito pelo `run-parts` ou dependente de variáveis do shell interativo pode não executar como esperado.

13.9 Ambiente reduzido do cron

Um erro comum é testar um comando no terminal, colocar a mesma linha no *cron* e descobrir que ela não funciona. Isso ocorre porque o ambiente do *cron* é mais reduzido do que uma sessão interativa.

Diferenças importantes:

- o `PATH` pode ser menor;
- aliases do shell não são carregados;
- funções definidas no perfil do usuário não existem;
- variáveis de ambiente podem estar ausentes;
- o diretório inicial pode não ser o esperado para o script;
- comandos que dependem de terminal interativo podem falhar.

Por isso, em tarefas agendadas, prefira caminhos absolutos:

```
0 2 * * * /usr/bin/tar -czf /backup/etc.tar.gz /etc
```

Em scripts, também é recomendável declarar o interpretador na primeira linha:

```
#!/bin/bash
```

E usar caminhos completos para arquivos importantes:

```
/usr/bin/rsync -a /home/aluno/ /backup/aluno/
```

13.10 Saída, erro e logs

Toda tarefa agendada deve registrar sua saída de alguma forma. Caso contrário, um erro pode passar despercebido.

Redirecionando saída padrão e erro para um arquivo:

```
0 2 * * * /usr/local/bin/backup.sh >> /tmp/backup.log 2>&1
```

Separando saída normal e erro:

```
LOG=/tmp/backup.log
ERR=/tmp/backup.err
0 2 * * * /usr/local/bin/backup.sh >> $LOG 2>> $ERR
```

Em muitas distribuições, eventos do *cron* aparecem em logs do sistema. Em Debian e Ubuntu, consulte:

```
$ grep CRON /var/log/syslog
$ journalctl -u cron
```

Em Rocky Linux e derivados RHEL:

```
$ grep CRON /var/log/cron
$ journalctl -u crond
```

O arquivo exato pode variar conforme a configuração de logs da distribuição. Quando houver dúvida, use `journalctl`, verifique o serviço do *cron* e procure mensagens recentes em `/var/log`.

13.11 Cuidados administrativos

Algumas práticas reduzem falhas em tarefas agendadas:

- testar o comando manualmente antes de agendar;
- executar o teste com o mesmo usuário do agendamento;
- usar caminhos absolutos para comandos e arquivos;
- registrar saída e erro;
- evitar comandos destrutivos sem validação prévia;
- documentar o objetivo da tarefa com comentários no *crontab*;
- revisar permissões de scripts e diretórios de destino;
- monitorar espaço em disco quando a tarefa gera arquivos.

Um agendamento mal construído pode consumir disco, sobrescrever dados, enviar relatórios incorretos ou mascarar uma falha operacional. Automatizar exige mais disciplina, não menos.

13.12 Diagnóstico de tarefas que não executam

Quando uma tarefa de *cron* não executa, investigue em ordem:

1. O serviço *cron* ou *crond* está ativo?
2. A linha do *crontab* está no usuário correto?
3. A sintaxe dos campos de tempo está correta?
4. O comando funciona manualmente com o mesmo usuário?

5. O script possui permissão de execução?
6. Os caminhos usados são absolutos?
7. O script depende de variável de ambiente ausente?
8. Há registro de execução nos logs?
9. A saída e o erro foram redirecionados para algum arquivo?

Comandos úteis:

```
$ systemctl status cron
$ systemctl status crond
$ crontab -l
$ sudo crontab -u usuario -l
$ ls -l /usr/local/bin/rotina.sh
$ journalctl --since "1 hour ago"
```

Em diagnóstico, evite alterar vários pontos ao mesmo tempo. Corrija uma hipótese, teste novamente e observe o log.

13.13 Systemd timers

Em sistemas baseados em *systemd*, os *timers* permitem agendar a execução de unidades de serviço. Eles oferecem integração com `systemctl`, `journalctl`, dependências, calendário e estado da unidade.

Para listar timers ativos:

```
$ systemctl list-timers
```

Comparação inicial:

Cron	**Systemd timer**
Simples e tradicional	Integrado ao gerenciamento de serviços
Muito usado em scripts legados	Melhor rastreabilidade com unidades
Sintaxe compacta	Configuração mais explícita
Bom para rotinas pequenas	Bom para tarefas com dependências do sistema

Neste módulo, o foco principal é o *cron*. Os *systemd timers* devem ser reconhecidos como alternativa atual, mas sua criação detalhada pode ser aprofundada em módulos posteriores.

13.14 Laboratório guiado: backup simples com cron

O objetivo deste laboratório é criar uma rotina simples de backup local usando `tar` e agendá-la com *cron*. Execute em uma máquina de laboratório.

13.14.1 1. Preparar diretórios

```
$ mkdir -p ~/lab-cron/dados
$ mkdir -p ~/lab-cron/backups
$ echo "arquivo de teste" > ~/lab-cron/dados/arquivo1.txt
$ date > ~/lab-cron/dados/gerado_em.txt
```

13.14.2 2. Criar o script de backup

Crie o arquivo `$HOME/lab-cron/backup.sh` com o conteúdo:

```
#!/bin/bash

ORIGEM="$HOME/lab-cron/dados"
DESTINO="$HOME/lab-cron/backups"
DATA="$(/usr/bin/date +%Y%m%d-%H%M%S)"

/usr/bin/mkdir -p "$DESTINO"
/usr/bin/tar -czf "$DESTINO/backup-$DATA.tar.gz" "$ORIGEM"
```

Ajuste a permissão:

```
$ chmod +x ~/lab-cron/backup.sh
```

13.14.3 3. Testar manualmente

```
$ ~/lab-cron/backup.sh
$ ls -lh ~/lab-cron/backups
```

Antes de agendar, o script precisa funcionar manualmente.

13.14.4 4. Agendar execução

Edite o *crontab* do usuário:

```
$ crontab -e
```

Para teste em laboratório, agende a execução a cada cinco minutos:

```
BASE=/home/aluno/lab-cron
*/5 * * * * $BASE/backup.sh >> $BASE/backup.log 2>&1
```

Substitua `/home/aluno/lab-cron` pelo caminho real criado no laboratório. Em um ambiente real, um backup a cada cinco minutos quase nunca é adequado; aqui ele serve apenas para observar rapidamente o funcionamento.

13.14.5 5. Verificar execução

Aguarde alguns minutos e verifique:

```
$ ls -lh ~/lab-cron/backups
$ tail -n 20 ~/lab-cron/backup.log
$ crontab -l
```

Se nada for criado, consulte os logs:

```
$ journalctl --since "10 minutes ago"
$ grep CRON /var/log/syslog
```

Em Rocky Linux:

```
$ grep CRON /var/log/cron
```

13.14.6 6. Encerrar o teste

Após validar o laboratório, remova ou comente a linha do *crontab* para evitar criação contínua de arquivos:

```
$ crontab -e
```

Também é possível remover os arquivos de laboratório:

```
$ rm -rf ~/lab-cron
```

Atenção: use `rm -rf` apenas em diretórios de laboratório que você criou e conferiu cuidadosamente. Em servidores reais, um caminho errado pode remover dados importantes.

13.15 Resumo

O *cron* permite executar tarefas recorrentes de forma automática. Para usá-lo com segurança, o administrador precisa entender a sintaxe de tempo, escolher o local correto do agendamento, testar comandos manualmente, considerar o ambiente reduzido do *cron* e registrar saída e erro.

Em sistemas modernos, *systemd timers* também podem ser usados para agendamentos, principalmente quando a tarefa precisa se integrar ao ciclo de vida de serviços. Ainda assim, o *cron* permanece uma ferramenta básica e necessária para administração Linux.

13.16 Perguntas de revisão

1. Qual é a diferença entre executar uma tarefa manualmente e agendá-la no *cron*?
2. Para que serve o comando `crontab -e`?
3. O que significa a linha `0 2 * * * /usr/local/bin/backup.sh`?
4. Por que é recomendado usar caminhos absolutos em tarefas agendadas?
5. Como consultar os agendamentos do usuário atual?
6. Qual é a diferença entre um *crontab* de usuário e `/etc/crontab`?
7. Por que a saída padrão e o erro devem ser redirecionados para um arquivo de log?
8. Que verificações devem ser feitas quando uma tarefa agendada não executa?
9. Em que situação um *systemd timer* pode ser mais adequado que uma linha simples de *cron*?

13.17 Desafio opcional

Adapte o laboratório para manter apenas os cinco backups mais recentes no diretório `backups`. Antes de remover qualquer arquivo automaticamente, faça o script apenas listar quais arquivos seriam removidos. Depois, acrescente a remoção real e registre a saída em um arquivo de log.

O objetivo do desafio é praticar automação com cuidado: primeiro observar, depois validar e somente então executar ações destrutivas.

14 Ferramentas de Redes

14.1 Objetivos

Ao final deste capítulo, o aluno deverá ser capaz de:

- identificar interfaces, endereços IP, rotas e conexões ativas;
- testar conectividade de camada 3 com *ping* e caminho de rede com *traceroute*;
- consultar resolução de nomes com *dig*;
- testar portas TCP/UDP em ambiente autorizado com *nc*;
- capturar e interpretar pacotes básicos com *tcpdump*;
- organizar um diagnóstico simples antes de alterar a configuração do sistema.

14.2 Pré-requisitos

Este capítulo pressupõe que o aluno já compreende os conceitos básicos de interface de rede, endereço IP, máscara, gateway, rota e DNS apresentados no capítulo de redes. As ferramentas mostradas aqui não substituem esses conceitos: elas ajudam a confirmar, negar ou localizar uma hipótese durante o diagnóstico.

14.3 Conceito: ferramenta não é diagnóstico

Em administração de sistemas, uma ferramenta de rede deve ser usada para responder uma pergunta objetiva. Antes de executar comandos aleatoriamente, o administrador deve formular hipóteses simples:

- a interface está ativa?
- o endereço IP está correto?
- existe rota para o destino?
- o DNS resolve o nome esperado?
- o serviço está escutando na porta correta?
- há pacotes saindo ou chegando na interface?

Essa ordem evita um erro comum em iniciantes: testar uma aplicação antes de confirmar a conectividade básica do sistema.

14.4 Sequência prática de diagnóstico

Uma sequência simples para diagnosticar falhas de rede em servidores Linux é:

1. Verificar interfaces e endereços com *ip address*.
2. Verificar rotas com *ip route*.
3. Testar conectividade com *ping*.
4. Verificar o caminho até o destino com *traceroute* ou *tracepath*.
5. Confirmar resolução de nomes com *dig*.
6. Verificar portas locais com *ss*.
7. Testar conexão de porta com *nc*, quando autorizado.
8. Capturar pacotes com *tcpdump*, quando os testes anteriores não explicarem a falha.

14.5 Ferramentas essenciais

14.5.1 ip

O comando *ip*, do pacote *iproute2*, é a ferramenta atual para consultar e alterar informações de rede no Linux. Ele substitui, na prática moderna, comandos legados como *ifconfig* e *route*.

Exemplos de consulta:

```
ip address show
ip link show
ip route show
```

Saída esperada em um servidor com uma interface Ethernet:

```
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.56.10/24 brd 192.168.56.255 scope global enp0s3

default via 192.168.56.1 dev enp0s3
192.168.56.0/24 dev enp0s3 proto kernel scope link src 192.168.56.10
```

Na leitura da saída, observe três pontos: a interface está *UP*, existe um endereço IP configurado e há uma rota padrão quando o servidor precisa acessar redes externas.

14.5.2 ss

O comando *ss* mostra sockets e conexões ativas. Ele substitui o uso rotineiro de *netstat* em distribuições modernas.

Exemplos:

```
ss -tulpen
ss -tan
ss -tan sport = :22
```

As opções mais usadas são:

- **-t**: mostra sockets TCP;
- **-u**: mostra sockets UDP;
- **-l**: mostra portas em escuta;
- **-p**: mostra o processo associado, quando permitido;
- **-n**: evita resolver nomes e portas, acelerando a leitura.

Se um serviço deveria estar acessível, mas não aparece em *ss -tulpen*, o problema provavelmente está no serviço, na configuração da aplicação ou no processo que deveria abrir a porta.

14.5.3 ping

O *ping* testa conectividade IP usando ICMP. Ele é útil para verificar se há resposta básica entre dois hosts, mas não prova que uma aplicação está funcionando.

```
ping -c 4 192.168.56.1
ping -c 4 8.8.8.8
```

Se o primeiro comando responde e o segundo não, a rede local pode estar funcionando, mas a saída para a Internet, a rota padrão ou o firewall precisam ser investigados.

14.5.4 traceroute e tracepath

O *traceroute* ajuda a observar o caminho até um destino. Em algumas instalações mínimas ele pode não estar instalado. O *tracepath* costuma estar disponível em ambientes Linux e pode ser usado como alternativa simples.

```
traceroute 8.8.8.8
tracepath 8.8.8.8
```

Saltos com asteriscos não significam, necessariamente, queda de rede. Muitos roteadores filtram respostas ICMP ou limitam respostas de diagnóstico. O resultado deve ser interpretado junto com os demais testes.

14.5.5 dig

O *dig* consulta DNS e permite separar problemas de conectividade de problemas de resolução de nomes.

```
dig www.debian.org
dig @8.8.8.8 www.debian.org
dig www.debian.org A
dig www.debian.org AAAA
```

Se o acesso por IP funciona, mas o acesso por nome falha, investigue DNS, */etc/resolv.conf*, configuração do resolvedor local e regras de firewall envolvendo a porta 53/UDP e 53/TCP.

14.6 Netcat e tcpdump no diagnóstico

O *netcat* e o *tcpdump* são ferramentas poderosas. Por isso, devem ser usadas apenas em ambientes próprios, laboratórios ou redes nas quais exista autorização explícita.

No contexto deste módulo:

- o *netcat* deve ser tratado como ferramenta de teste de portas, cliente TCP/UDP simples e laboratório local;
- qualquer exemplo envolvendo execução remota de shell deve ser entendido como risco de segurança e não como procedimento operacional aceitável;
- o *tcpdump* deve ser usado para observar pacotes, confirmar tráfego e apoiar o diagnóstico, respeitando políticas de privacidade e segurança da organização.

14.7 Netcat

14.7.1 Descrição

O *netcat*, normalmente chamado pelo comando *nc*, é uma ferramenta capaz de abrir conexões TCP e UDP, escutar portas e enviar dados simples pela rede. Para um administrador Linux, ele é útil principalmente para testar conectividade de aplicação: a porta responde? o serviço aceita conexão? o tráfego chega até o servidor?

O *netcat* não substitui ferramentas especializadas como *nmap*, *curl*, *telnet*, *ssh* ou clientes próprios de cada serviço. Seu valor está na simplicidade: ele permite testar rapidamente uma porta ou simular uma comunicação cliente-servidor em laboratório.

14.7.2 Cuidados de uso

Use *nc* apenas em sistemas próprios, laboratórios ou ambientes com autorização explícita. Testar portas de terceiros sem autorização pode violar políticas de segurança e legislação local.

Algumas versões antigas ou variantes do *netcat* foram usadas para criar canais remotos de comando. Esse uso deve ser entendido como risco de segurança, não como prática administrativa. Neste módulo, o foco é diagnóstico, teste de porta e laboratório controlado.

14.7.3 Variantes do netcat

Distribuições diferentes podem entregar implementações diferentes:

- *openbsd-netcat*, comum em Debian e Ubuntu;
- *ncat*, fornecido pelo projeto Nmap;
- *netcat-traditional*, mantido por compatibilidade.

Por isso, pequenas diferenças de sintaxe podem aparecer. Em caso de dúvida, consulte:

```
nc -h
man nc
```

14.7.4 Testando se uma porta TCP responde

O teste mais comum é verificar se uma porta está acessível:

```
nc -vz -w 2 <ENDERECO> <PORTA>
```

Exemplo em rede local:

```
nc -vz -w 2 192.168.56.10 22
```

Uma resposta de sucesso indica que a conexão TCP foi estabelecida. Uma falha pode indicar serviço parado, porta fechada, firewall, rota incorreta ou host indisponível.

Também é possível testar um pequeno intervalo de portas em laboratório:

```
nc -vz -w 2 192.168.56.10 20-25
```

Esse teste deve ser usado com moderação. Para varreduras amplas, utilize uma ferramenta própria e somente em ambiente autorizado.

14.7.5 Conectando manualmente a um serviço TCP

O *nc* pode abrir uma conexão simples com um serviço e enviar texto pela entrada padrão. O exemplo abaixo mostra uma consulta HTTP mínima:

```
nc www.example.org 80
GET / HTTP/1.0
Host: www.example.org
```

A linha em branco ao final é importante: ela encerra o cabeçalho HTTP/1.0 e faz o servidor responder. Esse tipo de teste ajuda a mostrar que muitos protocolos de rede são conversas estruturadas entre cliente e servidor.

14.7.6 Criando um serviço simples de laboratório

Em uma máquina do laboratório, deixe o *netcat* escutando em uma porta:

```
nc -l -p 9000
```

Em outra máquina, conecte nessa porta:

```
nc <IP_DO_SERVIDOR> 9000
```

Tudo que for digitado no cliente será enviado ao processo que está escutando. Esse laboratório é útil para demonstrar o modelo cliente-servidor sem instalar uma aplicação adicional.

Em algumas variantes do *nc*, a escuta pode ser feita sem a opção *-p*:

```
nc -l 9000
```

14.7.7 Testando UDP em laboratório

Para UDP, use a opção *-u*. Como UDP não estabelece conexão como TCP, a interpretação do teste exige mais cuidado.

Servidor:

```
nc -u -l -p 9001
```

Cliente:

```
nc -u <IP_DO_SERVIDOR> 9001
```

Digite uma mensagem no cliente e observe se ela chega ao servidor. Para confirmar o tráfego, combine o teste com *tcpdump*:

```
sudo tcpdump -i <INTERFACE> -nn udp port 9001
```

14.7.8 Transferência simples de arquivo em laboratório

O *netcat* também pode demonstrar fluxo de dados entre processos. Em uma máquina, escute em uma porta e grave o que chegar em arquivo:

```
nc -l -p 9002 > arquivo-recebido.txt
```

Na outra máquina, envie um arquivo:

```
nc <IP_DO_SERVIDOR> 9002 < arquivo-original.txt
```

Esse exemplo é didático. Em produção, prefira ferramentas com autenticação, criptografia, verificação de integridade e retomada de transferência, como *scp*, *sftp*, *rsync* sobre SSH ou soluções corporativas de backup.

14.7.9 Diagnóstico com netcat

Ao investigar uma falha, use o resultado do *nc* junto com outras ferramentas:

- se *ss -tulpen* não mostra a porta local, o serviço pode não estar escutando;
- se a porta escuta localmente, mas *nc* falha de outra máquina, investigue firewall, rota e endereço de escuta;
- se *nc* conecta, mas a aplicação falha, o problema pode estar no protocolo ou na camada de aplicação;
- se o teste é UDP, confirme com *tcpdump*, pois não há handshake de conexão.

14.7.10 Laboratório guiado

1. Em uma máquina, identifique o endereço IP com *ip address show*.
2. Na mesma máquina, abra uma porta com *nc -l -p 9000*.
3. Em outra máquina da mesma rede, conecte com *nc <IP> 9000*.
4. Envie uma mensagem simples e confirme o recebimento.
5. Repita o teste observando os pacotes com *tcpdump*.
6. Pare o *nc* e repita o teste de conexão para observar a diferença entre porta aberta e porta fechada.

14.7.11 Resumo

O *netcat* é uma ferramenta simples e poderosa para testar portas, simular conexões e apoiar diagnósticos. No Módulo 1, seu uso deve permanecer limitado a laboratórios e verificações autorizadas de conectividade.

14.8 tcpdump

14.8.1 Descrição

O *tcpdump* é uma ferramenta de captura de pacotes em linha de comando. Ele permite observar o tráfego que entra e sai por uma interface de rede, aplicar filtros e gravar capturas no formato PCAP para análise posterior.

Para o administrador Linux, o *tcpdump* responde perguntas práticas:

- o pacote saiu da máquina?
- a resposta voltou?
- o tráfego chegou na interface correta?
- o cliente está tentando acessar a porta esperada?
- há consulta DNS, tentativa TCP ou tráfego ICMP visível?

Capturar pacotes pode exigir privilégios de administrador. Em redes corporativas, respeite as políticas de segurança e privacidade antes de iniciar uma captura.

14.8.2 Identificando interfaces

Antes de capturar, identifique as interfaces disponíveis:

```
ip link show
tcpdump -D
```

Em servidores Linux atuais, nomes como *enp0s3*, *ens160*, *eth0*, *wlan0* e *lo* podem aparecer. A interface *lo* representa o loopback local.

14.8.3 Captura básica

Para capturar pacotes em uma interface:

```
sudo tcpdump -i <INTERFACE>
```

Na prática, quase sempre é melhor evitar resolução de nomes e portas, pois isso acelera a captura e reduz ruído:

```
sudo tcpdump -i <INTERFACE> -nn
```

Use *Ctrl+C* para encerrar. Ao final, o *tcpdump* mostra uma contagem de pacotes capturados, recebidos pelo filtro e descartados pelo kernel.

14.8.4 Limitando a quantidade de pacotes

Para capturar apenas uma quantidade pequena:

```
sudo tcpdump -i <INTERFACE> -nn -c 10
```

Essa opção é útil em aula e em diagnóstico inicial, pois evita capturas longas sem necessidade.

14.8.5 Filtros essenciais

Capturar todo o tráfego de uma interface pode gerar muita informação. Filtros simples tornam a análise mais objetiva.

Por host:

```
sudo tcpdump -i <INTERFACE> -nn host 192.168.56.10
```

Por origem:

```
sudo tcpdump -i <INTERFACE> -nn src host 192.168.56.10
```

Por destino:

```
sudo tcpdump -i <INTERFACE> -nn dst host 192.168.56.10
```

Por porta:

```
sudo tcpdump -i <INTERFACE> -nn port 22
```

Por protocolo:

```
sudo tcpdump -i <INTERFACE> -nn icmp
sudo tcpdump -i <INTERFACE> -nn tcp
sudo tcpdump -i <INTERFACE> -nn udp
```

Combinando filtros:

```
sudo tcpdump -i <INTERFACE> -nn tcp and port 22
sudo tcpdump -i <INTERFACE> -nn host 192.168.56.10 and port 53
sudo tcpdump -i <INTERFACE> -nn not port 22
```

14.8.6 Capturando tráfego DNS

DNS usa principalmente UDP/53, mas também pode usar TCP/53 em algumas situações. Para observar consultas DNS:

```
sudo tcpdump -i <INTERFACE> -nn port 53
```

Em outro terminal, execute:

```
dig www.debian.org
```

Se a consulta aparece, mas não há resposta, investigue conectividade com o servidor DNS, firewall e rota. Se nem a consulta aparece, investigue o resolvedor local e o comando usado para gerar a consulta.

14.8.7 Capturando tráfego ICMP

Para observar *ping*:

```
sudo tcpdump -i <INTERFACE> -nn icmp
```

Em outro terminal:

```
ping -c 3 <DESTINO>
```

Uma leitura típica mostra requisições *echo request* e respostas *echo reply*. Se apenas as requisições aparecem, o pacote sai, mas a resposta não volta ou está sendo filtrada.

14.8.8 Lendo uma linha de saída

Uma linha TCP simplificada pode se parecer com:

```
10:15:20 IP 192.168.56.20.51544 > 192.168.56.10.22: Flags [S]
```

Leitura básica:

- **10:15:20**: horário da captura;
- **192.168.56.20.51544**: IP e porta de origem;
- **192.168.56.10.22**: IP e porta de destino;
- **Flags [S]**: pacote TCP com flag SYN, início de tentativa de conexão.

Uma resposta com *Flags [S.]* indica SYN/ACK. Um pacote com *Flags [R]* indica reset, comum quando a porta está fechada ou a aplicação recusou a conexão.

14.8.9 Salvando e lendo capturas

Para salvar uma captura:

```
sudo tcpdump -i <INTERFACE> -nn -w captura.pcap
```

Para ler depois:

```
tcpdump -nn -r captura.pcap
```

Arquivos PCAP podem conter dados sensíveis. Antes de compartilhar uma captura, verifique se ela não contém senhas, tokens, endereços internos ou informações de clientes.

14.8.10 Laboratório guiado

1. Identifique a interface principal com *ip link show*.
2. Capture ICMP com *sudo tcpdump -i <INTERFACE> -nn icmp*.
3. Em outro terminal, execute *ping -c 3 <GATEWAY>*.
4. Capture DNS com *sudo tcpdump -i <INTERFACE> -nn port 53*.
5. Em outro terminal, execute *dig www.debian.org*.
6. Salve uma captura curta com *-w* e leia o arquivo com *-r*.

14.8.11 Diagnóstico com tcpdump

Use *tcpdump* quando os comandos anteriores não forem suficientes. Uma sequência prática é:

1. Confirmar endereço e rota com *ip address* e *ip route*.
2. Confirmar portas locais com *ss -tulpen*.
3. Testar conectividade com *ping*, *dig* ou *nc*.
4. Capturar pacotes específicos com *tcpdump* para confirmar o que realmente passa pela interface.

14.8.12 Resumo

O *tcpdump* mostra o tráfego real observado pela máquina. Para iniciantes, o domínio de *-i*, *-nn*, *-c*, *-w*, *-r*, *host*, *port*, *tcp*, *udp* e *icmp* já resolve grande parte dos diagnósticos básicos de rede.

14.9 Laboratório guiado: diagnóstico em rede local

14.9.1 Cenário

Utilize duas máquinas virtuais na mesma rede interna ou uma máquina Linux com serviços locais. Para evitar interferência em terceiros, execute os testes em endereços privados ou em *localhost*.

14.9.2 Parte 1: identificar a configuração da máquina

```
ip address show
ip route show
```

Registre:

- nome da interface;
- endereço IPv4;
- máscara;
- gateway padrão, se existir.

14.9.3 Parte 2: verificar portas locais

```
ss -tulpen
```

Identifique se há serviços escutando em portas TCP ou UDP. Em um servidor com SSH ativo, a porta 22/TCP deve aparecer em estado *LISTEN*.

14.9.4 Parte 3: testar conectividade e DNS

```
ping -c 4 <IP_DO_GATEWAY>
dig www.debian.org
```

Se o *ping* para o gateway funciona, mas o *dig* falha, o problema provavelmente não está na interface local. A próxima investigação deve focar DNS, rota externa ou firewall.

14.9.5 Parte 4: testar uma porta com netcat

Em uma máquina, abra uma porta de laboratório:

```
nc -l -p 9000
```

Em outra máquina, conecte:

```
nc <IP_DO_SERVIDOR> 9000
```

Digite uma mensagem simples e confirme se ela aparece no lado que está escutando. Esse teste demonstra o modelo cliente-servidor de forma simples, sem depender de uma aplicação real.

14.9.6 Parte 5: observar pacotes com tcpdump

Em um terminal, capture pacotes ICMP:

```
sudo tcpdump -i <INTERFACE> -nn icmp
```

Em outro terminal, execute:

```
ping -c 2 <IP_DO_GATEWAY>
```

Observe na captura o endereço de origem, o endereço de destino e o tipo de pacote. Interrompa a captura com *Ctrl+C*.

14.10 Resumo

Ferramentas de rede devem ser usadas com método. Primeiro confirma-se a configuração local, depois a rota, depois a resolução de nomes, depois portas e, por último, captura de pacotes. Essa sequência reduz conclusões precipitadas e aproxima o aluno da rotina real de um administrador Linux.

14.11 Perguntas de revisão

1. Qual é a diferença entre testar conectividade com *ping* e testar uma porta com *nc*?
2. Por que o comando *ip route show* deve ser verificado antes de investigar uma aplicação?
3. Em que situação o *dig @8.8.8.8 nome* ajuda no diagnóstico?
4. Por que *ss -tulpen* é útil ao investigar um serviço que não aceita conexão?
5. Quais cuidados devem ser tomados antes de executar *tcpdump* em uma rede corporativa?

14.12 Desafio opcional

Monte um relatório curto de diagnóstico contendo:

- configuração da interface;
- tabela de rotas;

- resultado de um teste de DNS;
- portas em escuta;
- uma captura curta de ICMP ou TCP com *tcpdump*;
- conclusão indicando se o problema observado está na rede, no DNS, no serviço ou na aplicação.

15 Introdução ao Modelo Cliente-Servidor

Antes de estudar serviços como DNS, SSH, HTTP ou compartilhamento de arquivos, é necessário compreender como dois programas conversam pela rede. Em uma rede IP, normalmente não é apenas uma máquina falando com outra máquina. Na prática, um processo cliente envia uma solicitação para um processo servidor, que escuta em uma porta, interpreta a solicitação e devolve uma resposta.

Esse modelo é chamado de cliente-servidor. Ele aparece em quase todas as rotinas de administração de sistemas: acesso remoto por SSH, resolução de nomes por DNS, consulta a páginas web, sincronização de pacotes, acesso a bancos de dados e integração entre aplicações.

15.1 Objetivos

Ao final deste capítulo, o aluno deverá ser capaz de:

- diferenciar cliente, servidor, requisição e resposta;
- compreender a função de portas, protocolos e sockets;
- relacionar o modelo cliente-servidor com as camadas TCP/IP e OSI;
- identificar serviços escutando em portas locais;
- testar uma comunicação simples usando *netcat*;
- preparar a base conceitual para os capítulos de DNS e SSH.

15.2 Pré-requisitos

Para acompanhar este capítulo, é recomendável que o aluno já tenha estudado:

- endereçamento IP, máscara, gateway e rota;
- uso básico de *ip*, *ss*, *ping* e *tcpdump*;
- pipes e redirecionamentos no shell;
- leitura básica de mensagens de erro no terminal.

15.3 Conceito de cliente e servidor

No modelo cliente-servidor, o **servidor** é o processo que oferece um serviço. Ele fica aguardando conexões ou consultas em uma porta conhecida. O **cliente** é o processo que inicia a comunicação para solicitar esse serviço.

Em uma conexão SSH, por exemplo:

- o servidor SSH normalmente escuta na porta TCP 22;
- o comando *ssh usuario@servidor* atua como cliente;
- o cliente inicia a conexão;
- o servidor responde, negocia criptografia e autenticação;
- depois da autenticação, os comandos passam pelo canal estabelecido.

Em uma consulta DNS, a ideia é semelhante, embora o serviço costume usar UDP:

- o resolvedor da máquina cliente envia uma consulta;
- o servidor DNS recebe a pergunta na porta 53;
- o servidor responde com o endereço IP ou com uma informação de erro;
- a aplicação cliente usa a resposta para continuar a comunicação.

Portanto, cliente e servidor não são obrigatoriamente máquinas diferentes. Eles podem ser processos diferentes na mesma máquina. Um notebook pode ser cliente ao acessar um servidor web, mas também pode ser servidor ao executar um serviço SSH para acesso remoto autorizado.

15.4 Requisição, resposta e estado da comunicação

Uma comunicação cliente-servidor normalmente segue uma sequência simples:

1. o servidor inicia e fica escutando em uma porta;
2. o cliente descobre o endereço do servidor;
3. o cliente envia uma requisição;
4. o servidor processa a requisição;
5. o servidor envia uma resposta;
6. cliente e servidor encerram ou mantêm a comunicação aberta.

Nem todo protocolo mantém uma conexão aberta. O DNS, em consultas simples via UDP, envia uma pergunta e recebe uma resposta sem estabelecer uma conexão TCP tradicional. O SSH, por outro lado, mantém uma sessão TCP aberta enquanto o usuário trabalha remotamente.

15.5 Portas, protocolos e sockets

O endereço IP identifica a máquina ou interface de rede. A porta identifica o serviço dentro daquela máquina. Sem a porta, o sistema operacional saberia para qual host entregar o pacote, mas não saberia qual processo deve receber os dados.

Alguns exemplos comuns:

Serviço	**Protocolo de transporte**	**Porta comum**
SSH	TCP	22
DNS	UDP/TCP	53
HTTP	TCP	80
HTTPS	TCP	443
SMTP	TCP	25

Um **socket** representa um ponto de comunicação entre processos. Em redes TCP/IP, ele normalmente combina:

- endereço IP local;
- porta local;
- endereço IP remoto;
- porta remota;
- protocolo de transporte, como TCP ou UDP.

Quando um administrador executa:

```
ss -ltn
```

o sistema lista sockets TCP em estado de escuta. Isso ajuda a responder uma pergunta operacional importante: “o serviço está realmente escutando na porta esperada?”.

15.6 Relação com OSI e TCP/IP

O modelo ISO/OSI é uma referência conceitual com sete camadas. Ele ajuda a separar responsabilidades: meio físico, enlace, rede, transporte, sessão, apresentação e aplicação. Na

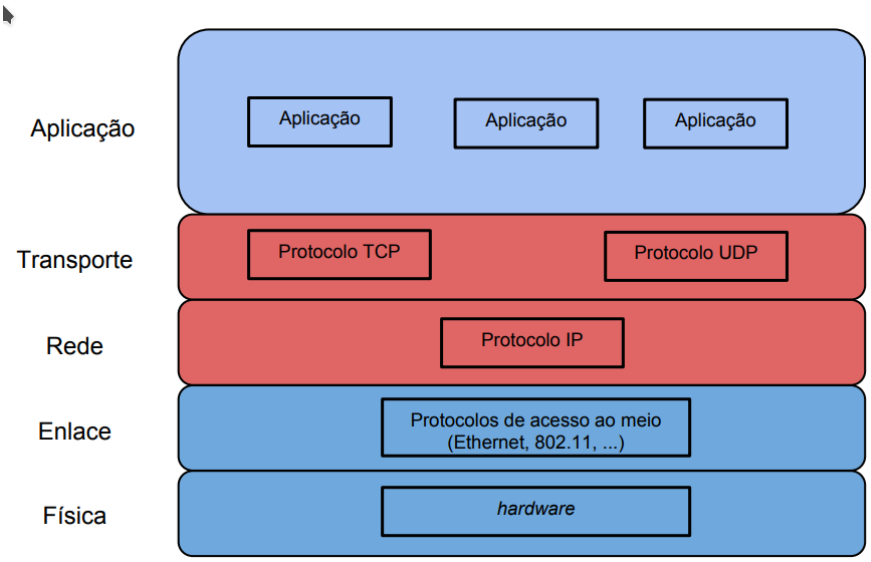


Figura 7 – Modelo OSI simplificado para comunicação cliente-servidor.

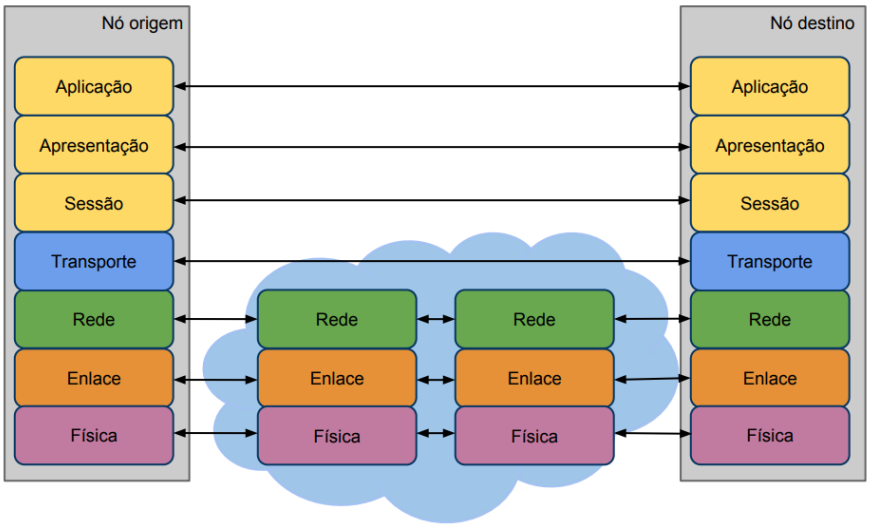


Figura 8 – Modelo OSI de 7 camadas.

prática da administração Linux, também é comum usar o modelo TCP/IP, mais próximo da implementação real da Internet.

A figura 8 apresenta o modelo ISO/OSI. ISO corresponde a *International Organization for Standardization*, ou Organização Internacional para Padronização. OSI corresponde a *Open Systems Interconnection*, ou Interconexão de Sistemas Abertos.

Para o administrador iniciante, o mais importante é entender a relação prática entre as camadas:

- a camada de rede trata principalmente de endereços IP e roteamento;
- a camada de transporte trata de TCP, UDP, portas e entrega entre processos;
- a camada de aplicação trata dos protocolos usados pelos serviços, como DNS, SSH e HTTP.

Ao diagnosticar um problema, essa separação evita conclusões apressadas. Se o IP não responde, talvez o problema esteja em rota, interface ou firewall. Se o IP responde, mas a porta TCP não abre, talvez o serviço esteja parado, filtrado ou escutando em outro endereço. Se a porta abre, mas a aplicação falha, o problema pode estar no protocolo, na autenticação, na permissão ou na configuração do serviço.

15.7 TCP e UDP no contexto cliente-servidor

O TCP é orientado à conexão. Antes de trocar dados da aplicação, cliente e servidor estabelecem uma sessão. Isso permite controle de sequência, confirmação de entrega e retransmissão quando necessário. Serviços como SSH, HTTP e HTTPS usam TCP porque precisam de uma comunicação ordenada e confiável.

O UDP não estabelece uma conexão antes do envio. Ele é mais simples e tem menor sobrecarga, mas não garante entrega, ordem ou retransmissão por conta própria. O DNS usa UDP na maior parte das consultas comuns, mas também pode usar TCP em situações específicas, como respostas grandes ou transferência de zona.

Para o diagnóstico, essa diferença é importante. Um teste com *telnet* ou *nc* abrindo uma conexão TCP não prova que um serviço UDP está funcionando. Para DNS, por exemplo, é mais adequado usar ferramentas como *dig*, *resolvectl* ou capturas com *tcpdump* na porta 53.

15.8 Observando serviços em uma máquina Linux

Em uma máquina Linux, o comando *ss* mostra conexões e portas em uso. Para listar serviços TCP escutando:

```
ss -ltn
```

Para incluir o processo responsável, normalmente é necessário privilégio de administrador:

```
sudo ss -ltnp
```

Exemplo de interpretação:

```
State   Local Address:Port  Peer Address:Port Process
LISTEN        0.0.0.0:22         0.0.0.0:*     sshd
```

Neste exemplo, há um processo *sshd* escutando na porta TCP 22 em todos os endereços IPv4 locais. Isso não garante que o acesso remoto funcionará de fora da máquina, mas confirma que o serviço local está ativo e escutando.

15.9 Laboratório guiado: comunicação local com netcat

Este laboratório usa apenas a própria máquina. Ele demonstra a ideia de um processo servidor escutando em uma porta e um processo cliente enviando dados.

15.9.1 Preparação

Abra dois terminais no mesmo computador.

No primeiro terminal, inicie um servidor TCP simples:

```
nc -l 127.0.0.1 9000
```

Em algumas distribuições ou variantes do *netcat*, a sintaxe pode ser:

```
nc -l -p 9000 127.0.0.1
```

Se uma forma não funcionar, consulte a ajuda local:

```
nc -h
```

15.9.2 Teste do cliente

No segundo terminal, conecte-se ao servidor local:

```
nc 127.0.0.1 9000
```

Digite uma mensagem no segundo terminal:

```
teste de comunicacao cliente-servidor
```

A mensagem deverá aparecer no primeiro terminal. Nesse momento:

- o primeiro terminal representa o servidor;
- o segundo terminal representa o cliente;
- *127.0.0.1* é o endereço de loopback;
- *9000* é a porta TCP usada no teste;
- o protocolo de aplicação é apenas texto simples digitado pelo usuário.

15.9.3 Observação com ss

Enquanto o servidor estiver escutando, abra um terceiro terminal e execute:

```
ss -ltn | grep 9000
```

Saída esperada:

```
LISTEN 0 1 127.0.0.1:9000 0.0.0.0:*
```

A saída confirma que há um socket TCP em escuta no endereço local *127.0.0.1*, porta *9000*.

15.9.4 Observação com tcpdump

Para observar a comunicação no loopback:

```
sudo tcpdump -i lo -nn tcp port 9000
```

Repita a conexão com *nc* e observe os pacotes. O objetivo não é decorar todos os campos da captura, mas identificar que existe uma tentativa de conexão TCP e tráfego associado à porta testada.

15.10 Diagnóstico básico

Ao investigar um serviço cliente-servidor, siga uma ordem simples:

1. o endereço IP do servidor está correto?
2. existe rota até o servidor?
3. o serviço está escutando na porta esperada?
4. o firewall permite a comunicação?
5. o protocolo usado pelo cliente é o protocolo esperado pelo servidor?
6. há logs indicando erro de autenticação, permissão ou configuração?

Essa sequência evita confundir problemas diferentes. Uma falha de DNS, por exemplo, pode impedir que o nome do servidor seja encontrado, mas não significa que o serviço SSH esteja parado. Uma porta fechada pode indicar serviço parado, firewall bloqueando ou serviço escutando apenas em outro endereço.

15.11 Relação com DNS e SSH

O próximo passo natural é aplicar esse modelo a serviços reais.

No DNS, o cliente normalmente pergunta: "qual é o endereço IP deste nome?". O servidor DNS responde com registros de nomes, endereços ou mensagens de erro. O serviço usa principalmente a porta 53.

No SSH, o cliente solicita uma sessão remota segura. O servidor SSH escuta uma porta TCP, negocia criptografia, valida a identidade do servidor, autentica o usuário e, se tudo estiver correto, permite a administração remota.

Portanto, DNS e SSH não devem ser vistos como comandos isolados. Eles são serviços de rede que seguem o mesmo raciocínio básico: cliente, servidor, endereço, porta, protocolo, requisição, resposta e diagnóstico.

15.12 Resumo

O modelo cliente-servidor descreve a comunicação entre um processo que solicita um serviço e outro processo que oferece esse serviço. O endereço IP identifica a máquina, a porta identifica o serviço e o protocolo define as regras da conversa. Comandos como *ss*, *nc* e *tcpdump* ajudam a observar esse comportamento na prática.

Esse entendimento será usado nos capítulos seguintes para estudar DNS, SSH e outros serviços de rede com mais clareza operacional.

15.13 Perguntas de revisão

1. Qual é a diferença entre cliente e servidor?
2. Por que uma máquina precisa de portas além do endereço IP?
3. O que significa dizer que um serviço está "escutando"?
4. Por que um teste TCP não valida automaticamente um serviço UDP?
5. Em que camada prática aparecem portas TCP e UDP?
6. Como o comando *ss -ltn* ajuda no diagnóstico?

15.14 Desafio opcional

Repita o laboratório com duas máquinas na mesma rede. Em uma delas, execute o *netcat* em modo escuta usando o endereço IP da interface de rede. Na outra, conecte-se a esse endereço e porta. Se a conexão falhar, investigue rota, firewall local, serviço em escuta e endereço usado pelo servidor.

16 Servidores de Serviços

16.1 DNS

16.1.1 Objetivos

Ao final desta seção, o aluno deverá ser capaz de:

- compreender o papel do DNS na resolução de nomes;
- diferenciar domínio, zona, delegação e autoridade;
- identificar servidores recursivos, autoritativos e de cache;
- reconhecer registros DNS básicos, como A, AAAA, CNAME, MX, NS, SOA, PTR e TXT;
- utilizar ferramentas de consulta e diagnóstico, como `dig`, `host` e `resolvectl`.

16.1.2 Pré-requisitos

Para acompanhar este capítulo, é importante que o aluno já tenha estudado:

- fundamentos de redes TCP/IP;
- modelo cliente-servidor;
- portas de rede, especialmente UDP e TCP;
- uso básico do shell Linux;
- leitura de arquivos de configuração em `/etc`.

O DNS (*Domain Name System*) é um "sistema de nomes" cujo objetivo primário é mapear, em escala global, nomes de domínios de rede e nomes de máquinas em endereços IP, processo conhecido por "resolução de nomes". O DNS possui também a funcionalidade reversa, traduzindo endereços IP em nomes.

Na prática, quando um usuário acessa um endereço como `www.exemplo.com.br`, alguma parte do sistema precisa descobrir qual endereço IP corresponde àquele nome. O DNS resolve esse problema de forma distribuída, evitando que cada computador precise manter uma lista completa de todos os nomes existentes na Internet. As consultas DNS usam, normalmente, a porta 53/UDP; a porta 53/TCP também é usada em situações como respostas maiores, transferências de zona e outros cenários específicos.

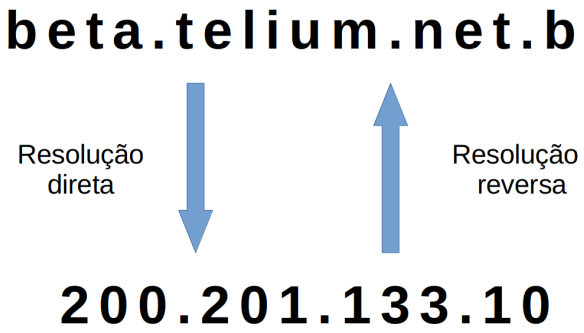


Figura 9 – Resolução de nomes de domínio - DNS.

O DNS é estruturado na forma de um banco de dados hierárquico distribuído como mostrado na figura 10, onde cada servidor autoritativo é responsável por responder pelos dados de uma ou mais zonas.

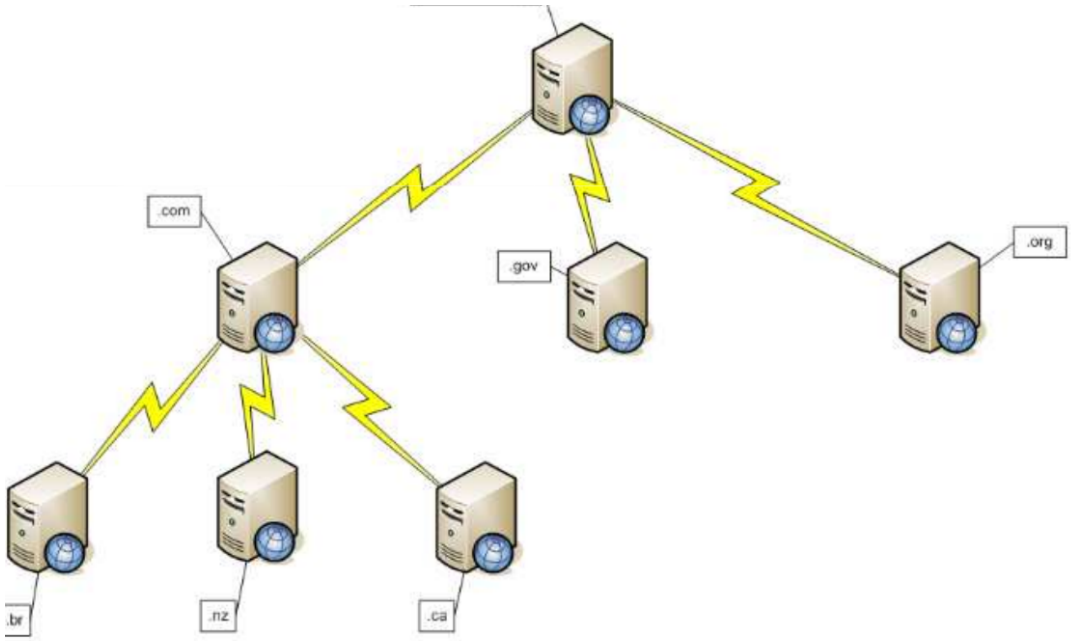


Figura 10 – Distribuição hierárquica do DNS.

16.1.3 Histórico

Antigamente, o mapeamento entre nomes de hosts e endereços IP era mantido em uma tabela estática, implementada como um arquivo de texto único (Unix: arquivo */etc/hosts*). Esta tabela era gerenciada de forma centralizada e era distribuída para todos os computadores da antiga Arpanet. Os nomes de hosts não seguiam o esquema hierárquico atual. O procedimento para nomear um computador incluía verificar se já existia um outro computador com aquele nome. Como se pode deduzir, o arquivo estava constantemente desatualizado.

O DNS resolveu o problema de se ter uma tabela estática administrada centralmente introduzindo dois novos conceitos:

- Nomes de hosts hierárquicos;
- Distribuição da responsabilidade pela resolução de nomes.

O DNS foi especificado formalmente por Paul Mockapetris nas RFCs 882 e 883 (1983), alterado pelas RFCs 1034 e 1035 (1987) e estendido nas RFCs 1101 e 1183 (1990). Em 1985, Kevin Dunlap, em Berkeley, produziu o BIND – Berkeley Internet Name System, uma implementação de sucesso do DNS. O BIND é hoje parte da maioria das implementações Unix.

16.1.4 Espaço de nomes do DNS

O espaço de nomes do DNS é representado em uma estrutura hierárquica, em forma de árvore, onde cada nó possui um *label* de até 63 caracteres. Não existe diferenciação entre letras maiúsculas e minúsculas, e a raiz da árvore é um nó especial, de *label* nulo. O espaço de nomes do DNS é, na verdade, uma árvore de nomes de domínios (*domain names*); a raiz é chamada de "." (ponto). Antigamente a raiz estava no Departamento de Defesa dos EUA (DoD – *Department of Defense*); atualmente a coordenação global envolve a ICANN/IANA e operadores independentes dos *root servers*.

Existem 13 servidores especificados como *root servers*, com nomes no formato *letra.root-servers.net* , onde a letra varia de **a** a **m**. A escolha de treze servidores de nomes foi feita devido às limitações na especificação DNS original, o qual especificava um tamanho máximo de pacote de 512 bytes ao usar o protocolo UDP. Tecnicamente, no entanto, 14 servidores de nomes cabem em um pacote IPv4. Com a adição de endereços IPv6 nos *root servers*, houve a necessidade de mais 512 bytes, o que foi facilitado pela extensão EDNS0 do padrão DNS (**??**).

A distribuição dos *root servers* pode ser encontrada em https://root-servers.org/. Na figura 11, temos a distribuição de *root servers* em SP - Brasil.

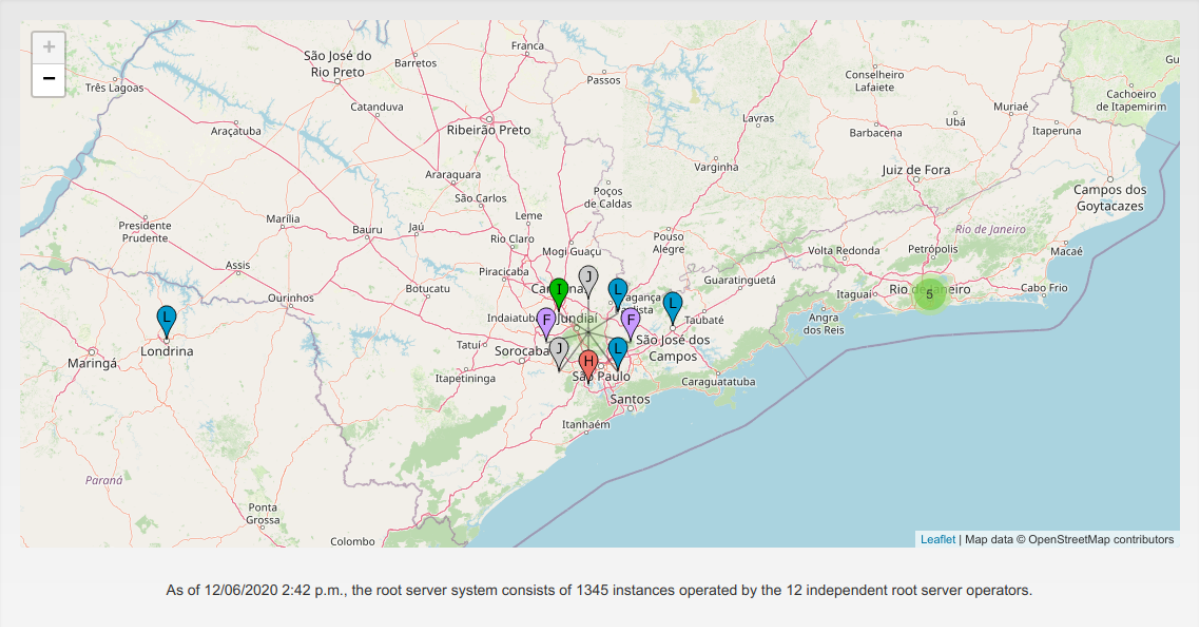


Figura 11 – Distribuição *root servers* no estado de SP - Brasil - Fonte (ROOT-SERVERS. . . , 2020).

O domínio de qualquer nó da árvore é a lista dos *labels*, começando por aquele nó, até a raiz, usando um ponto (".") como separador de *labels*. Cada domínio representa uma parte distinta do espaço de nomes e é mantido por uma (única) entidade administrativa.

Exemplo: telium.net.br amazon.com petrobras.com.br columbia.edu

Debaixo da raiz estão os domínios denominados de "*top-level domains*" (TLDs). Esses domínios são relativamente estáveis. Por razões históricas, existem dois tipos principais de *top-level domain names*, como mostrado na figura 12.

- Domínios genéricos, como `.edu`, `.net`, `.com` e `.org`.
- Domínios de países, como `.br`, `.uk`, `.fr` e `.jp`.

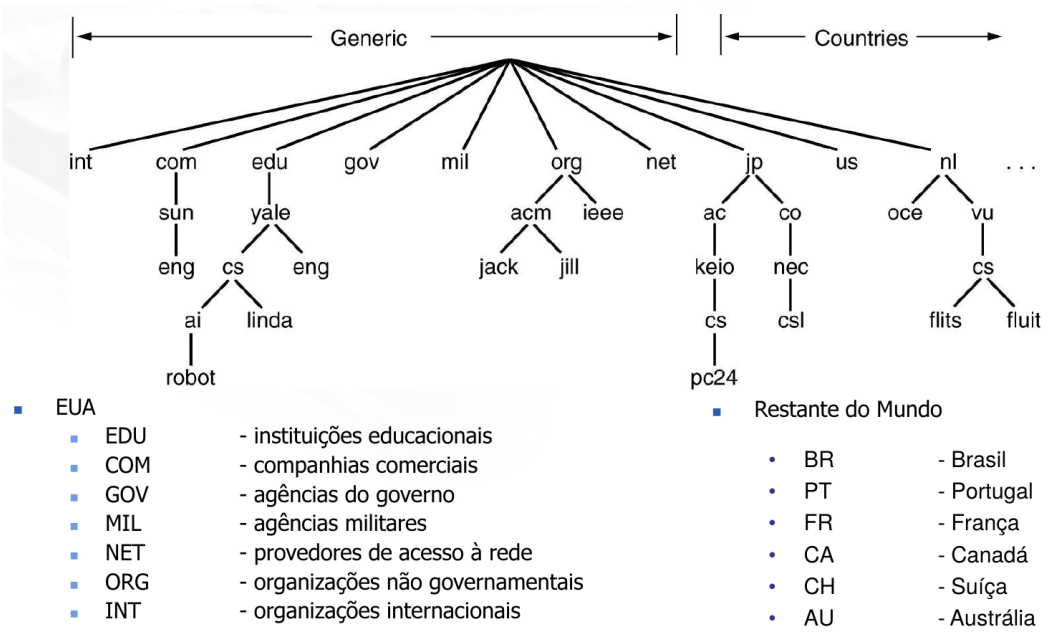


Figura 12 – Exemplo da estrutura de árvore dos *top-level domain names*.

Um nome de domínio que termina com um ponto é dito totalmente qualificado.

FQDN – fully qualified domain name

Se o nome não termina com ponto, ele necessita ser completado pelo software que implementa o DNS.

- beta
- beta.telium.net.br.

Cada parte da árvore de domínios pode ser gerenciada por uma organização diferente, de forma hierárquica. A responsabilidade pelos domínios brasileiros (nomes terminando em "**.br**") está a cargo do serviço Registro.BR, mantido pelo Comitê Gestor da Internet no Brasil (**CGI.br**).

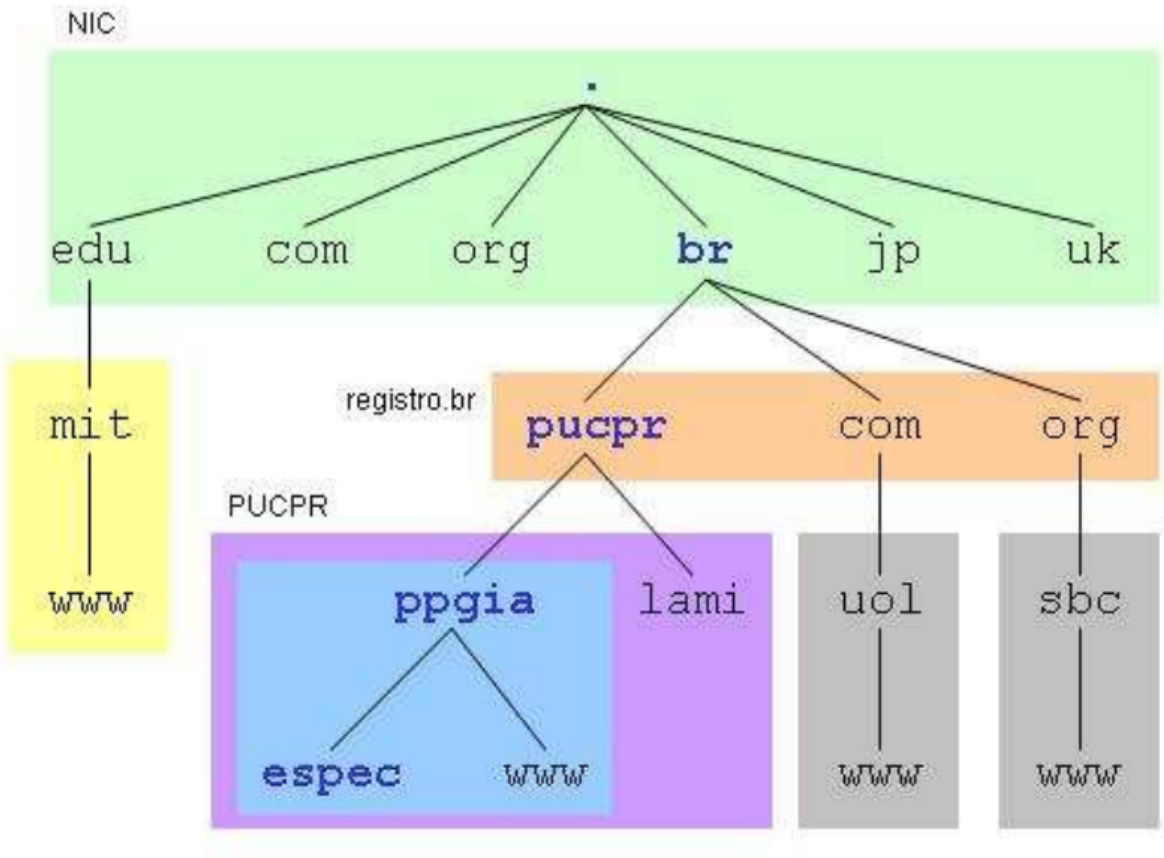


Figura 13 – Exemplo da estrutura de árvore de responsabilidade dos domínios.

16.1.5 DNS nas Organizações

As recomendações para servidores de DNS variam conforme o tamanho e a criticidade do ambiente:

- **Pequena organização**: pode usar DNS do provedor, DNS público confiável ou manter um servidor interno quando houver nomes locais a resolver.
- **Organização média**: com várias sub-redes, pode manter servidores recursivos internos para reduzir latência, controlar políticas locais e melhorar o diagnóstico.
- **Organização grande**: tende a separar DNS recursivo interno, DNS autoritativo interno e DNS autoritativo público, além de delegar zonas conforme a estrutura administrativa.

Observações

Essas recomendações devem considerar o número de domínios, o volume de consultas, a criticidade do serviço e a separação entre nomes internos e nomes publicados na Internet.

16.1.6 Zonas e Domínios

Domínio e zona são conceitos relacionados, mas não são sinônimos. Um domínio é uma parte da árvore de nomes, como `telium.net.br`. Uma zona é a parte desse domínio administrada por um conjunto de servidores autoritativos. Em muitos ambientes simples, domínio e zona parecem ser a mesma coisa; em ambientes maiores, um domínio pode delegar subdomínios para outras zonas.

As zonas devem obedecer à estrutura hierárquica. Cada zona se relaciona com sua zona superior, registra nomes de máquinas e pode delegar uma ou mais zonas inferiores. Cada zona inferior deve ser registrada na zona superior por meio de registros apropriados, normalmente registros NS e seus endereços associados. Na figura 14, temos uma representação dessa estrutura.

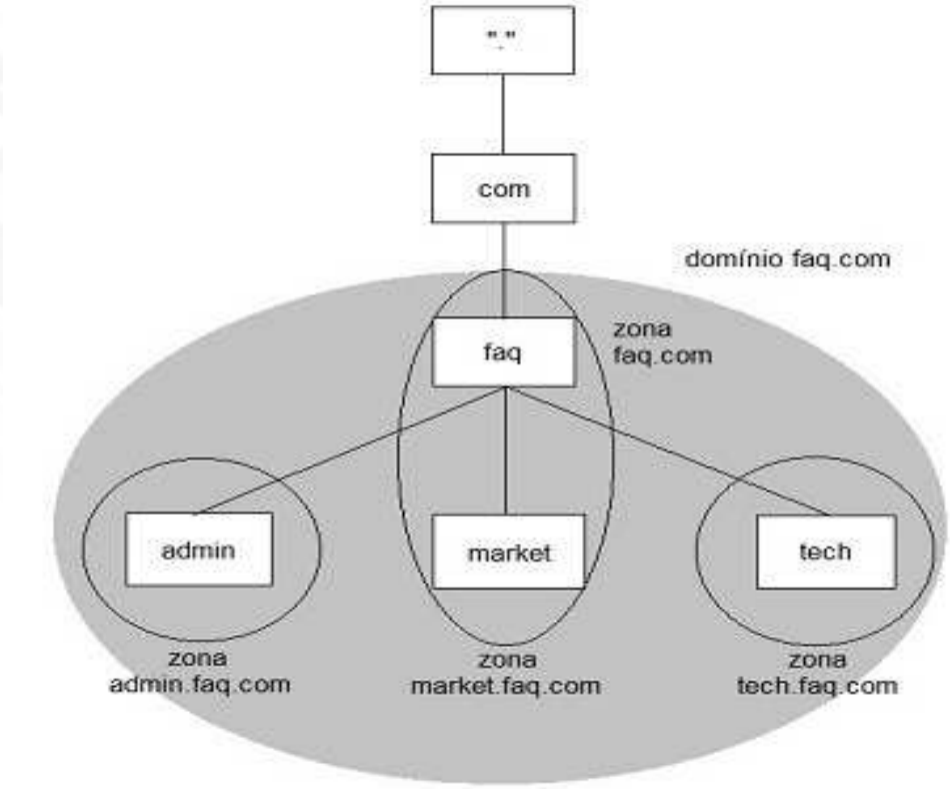


Figura 14 – Exemplo da estrutura de árvore de zonas.

Um servidor pode ser responsável por mais de uma zona, sempre obedecendo à estrutura de hierarquia, como na figura 15.

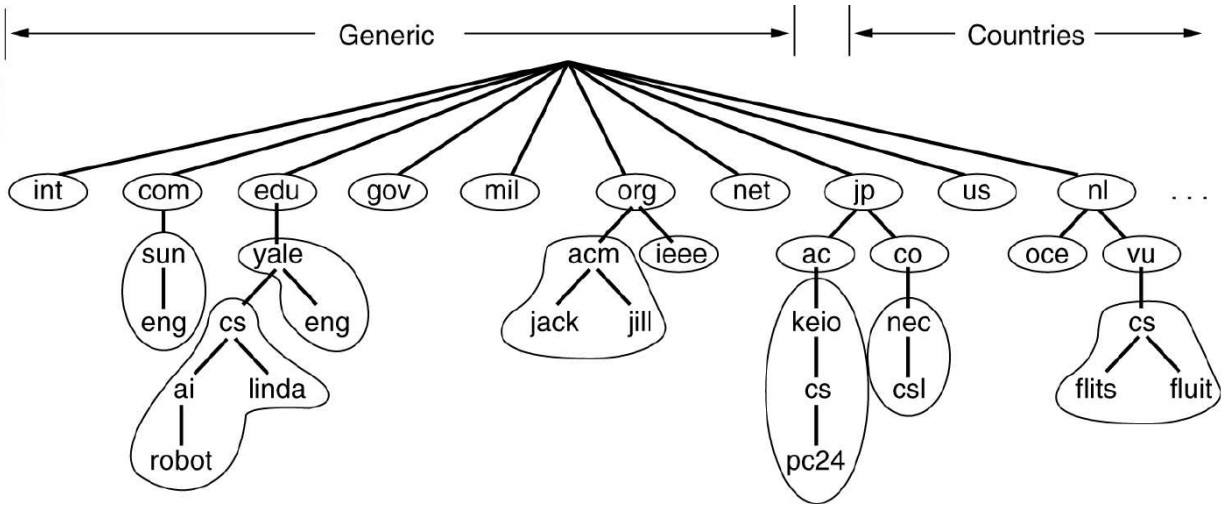


Figura 15 – Exemplo da estrutura de árvore com múltiplas zonas.

16.1.7 Configurações de Servidores de DNS

Podemos classificar servidores DNS de acordo com sua funcionalidade:

- Servidor autoritativo primário
- Servidor autoritativo secundário
- Servidor recursivo com cache

O servidor autoritativo primário mantém a cópia principal dos dados da zona. Existe apenas um servidor primário para cada zona. Deve estar localizado em uma máquina estável, segura, com boa conectividade e, preferencialmente, protegida por fonte ininterrupta de energia.

O servidor autoritativo secundário copia os dados do servidor primário através de uma operação chamada "transferência de zona". Pode haver vários servidores secundários para uma zona. É recomendado existir pelo menos um servidor secundário fora da mesma máquina e, em ambientes críticos, fora da mesma rede ou circuito de energia.

O servidor recursivo com cache recebe consultas dos clientes, busca a resposta na hierarquia DNS quando necessário e armazena temporariamente os resultados conforme o TTL dos registros. Em redes corporativas, ele costuma ser o servidor configurado nos clientes Linux, estações de trabalho e servidores internos.

16.1.8 Tipos de Servidores de DNS

16.1.8.1 Root Name Servers

Todo servidor DNS tem que saber, pelo menos, como contactar um dos "servidores raiz" (*root name servers*). Os *root servers*, por sua vez, sabem o nome e endereço IP de cada um dos servidores DNS autoritativos de cada domínio de segundo nível (.com, .edu, .br, .uk, etc.).

O *root server* encaminha ao servidor solicitante os endereços de servidores autoritativos que podem resolver a consulta desejada (p. ex., quem são os servidores de nomes autoritativos do domínio .br).

Existem atualmente 13 *root name servers*, com nomes na forma *letter.root-servers.net*, onde letter varia de A a M. Isto não significa que existam apenas 13 servidores físicos; cada operador usa equipamento redundante para prover um serviço confiável mesmo na ocorrência de falha de hardware ou software.

Parte desses servidores operam em múltiplos locais, o que provê maior performance e tolerância a falhas. Há agora servidores C, F, I, J, K, L e M localizados em diferentes continentes usando anúncios *anycast* para prover um serviço descentralizado.

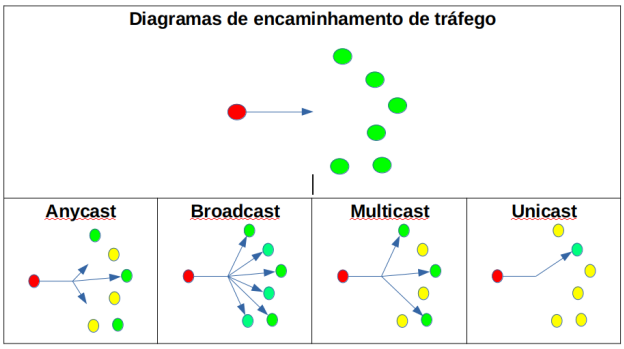


Figura 16 – Tipos de encaminhamento de tráfego.

Como resultado, a maioria dos *root servers* estão hoje fora dos EUA, permitindo uma maior performance nas consultas. Uma relação atualizada pode ser obtida no site: https://root-servers.org/

16.1.8.2 TLD – Top Level Domain Servers

Servidores *Top-level domain* (TLD) são os servidores DNS responsáveis por domínios como `.org`, `.net`, `.com`, `.edu` e pelos domínios de países, tais como `.br`, `.uk`, `.fr`, `.ca` e `.jp`.

- Operadores de registro mantêm a infraestrutura autoritativa para TLDs genéricos, como `.com`.
- O NIC.br/Registro.br mantém a infraestrutura relacionada ao domínio `.br`.

16.1.8.3 Servidor Interativo (não-Recursivo) e Recursivo

Servidor Interativo (não-Recursivo)

É um servidor considerado com comportamento "lazy". Se ele tiver a resposta em seu cache proveniente de uma consulta anterior ou se ele é autoritativo para o domínio ao qual o nome consultado pertence, então ele retorna uma resposta

apropriada; caso contrário, retorna uma referência a servidores autoritativos de um outro domínio que sejam mais prováveis de terem a resposta.

O cliente de um servidor DNS não-recursivo deve estar preparado para aceitar e saber agir ao receber as referências a outros servidores.

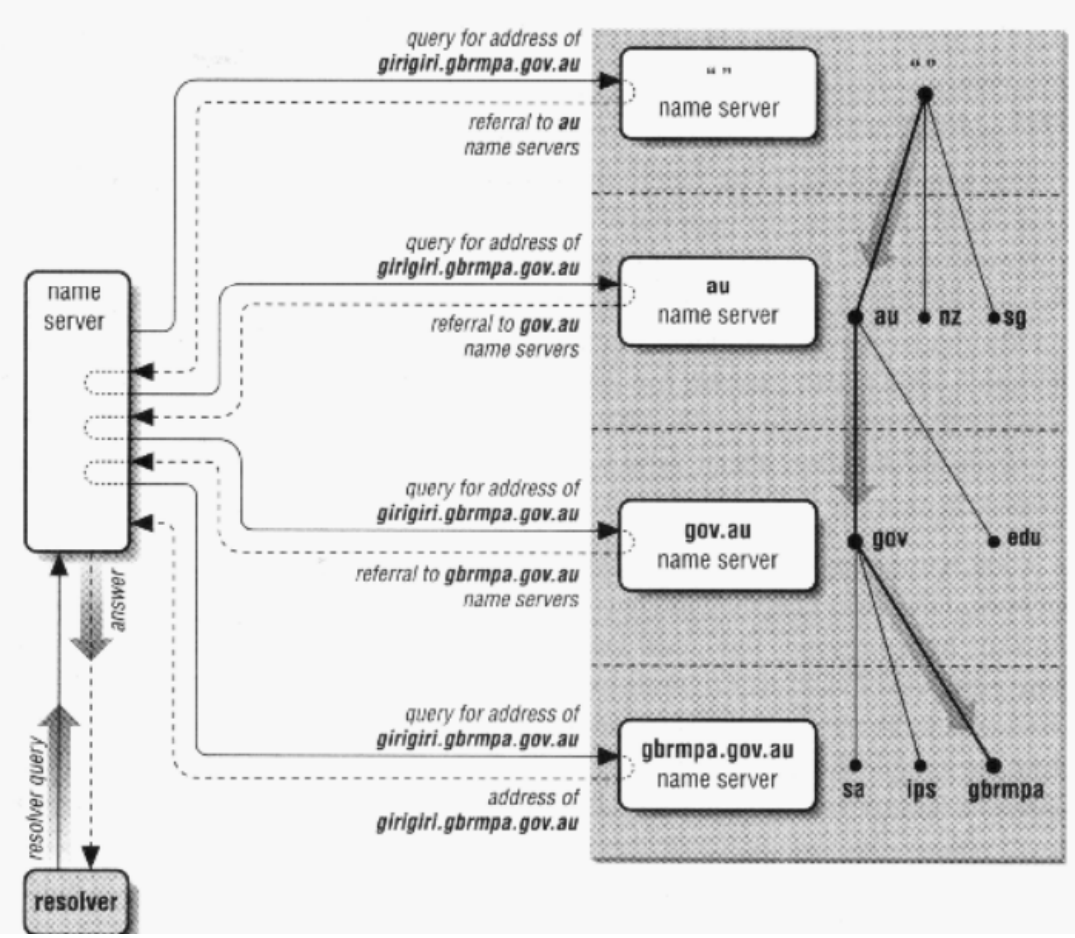


Figura 17 – Exemplo de consulta feita por um servidor interativo (não-recursivo).

Na figura 18, há um exemplo de fluxo de dados do DNS em um servidor interativo. O solicitante pergunta qual é o endereço de `www.gnu.org` ao seu servidor DNS local. O servidor local consulta um *root server*, que responde com a indicação dos servidores responsáveis por `.org`. Em seguida, o servidor local consulta o TLD `.org`, recebe a indicação dos servidores autoritativos de `gnu.org` e, por fim, consulta o servidor autoritativo da zona para obter o endereço IP procurado.

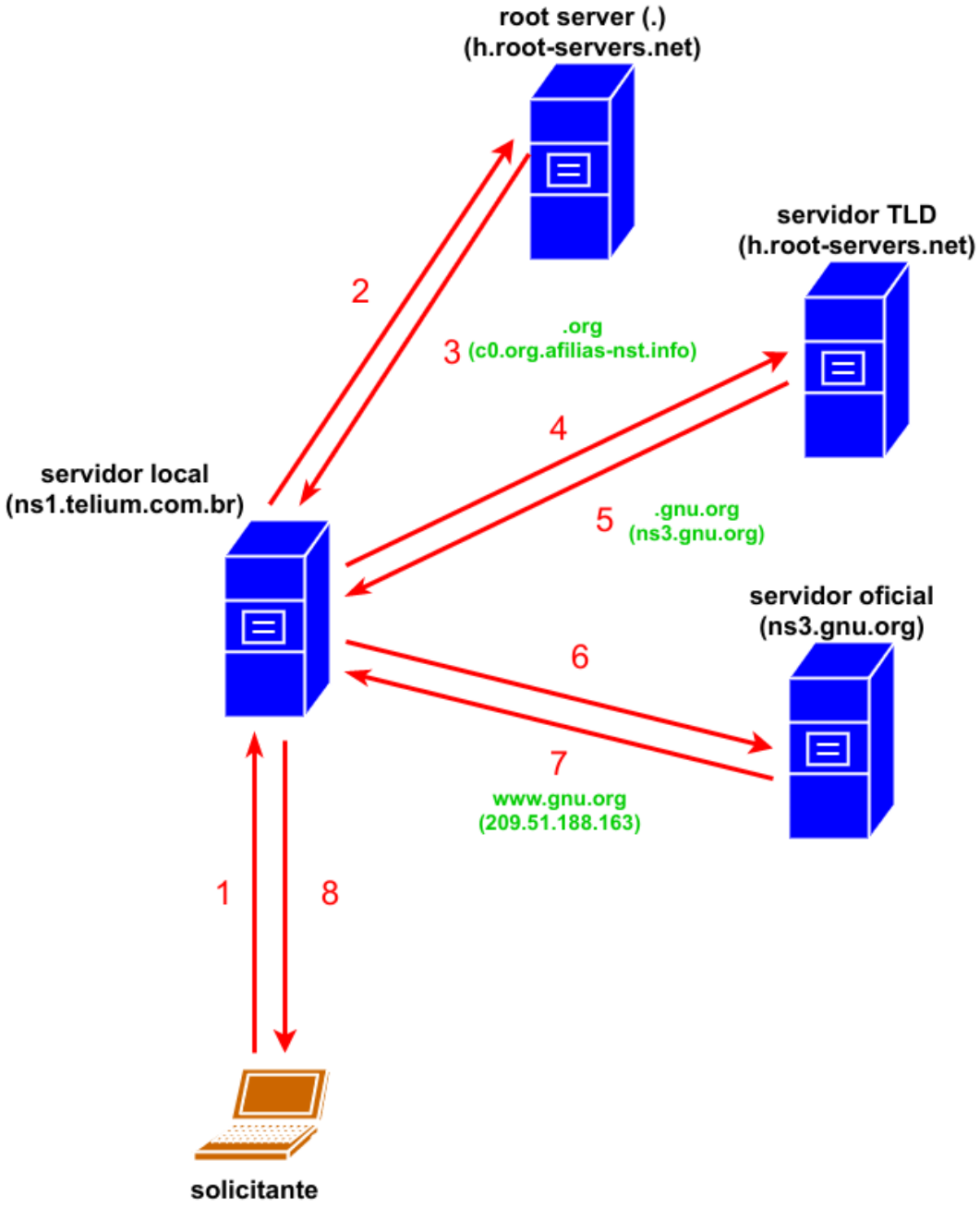


Figura 18 – Consulta a um servidor recursivo.

Servidor Recursivo

Retorna somente respostas reais ou mensagens de erro. Ele próprio segue as referências, liberando o cliente desta responsabilidade. O procedimento de resolver a consulta é o mesmo; o que muda é que o servidor recursivo contata os outros servidores, em vez de passar esta tarefa para o cliente. A figura 19 mostra o fluxo de dados do DNS em uma consulta recursiva.

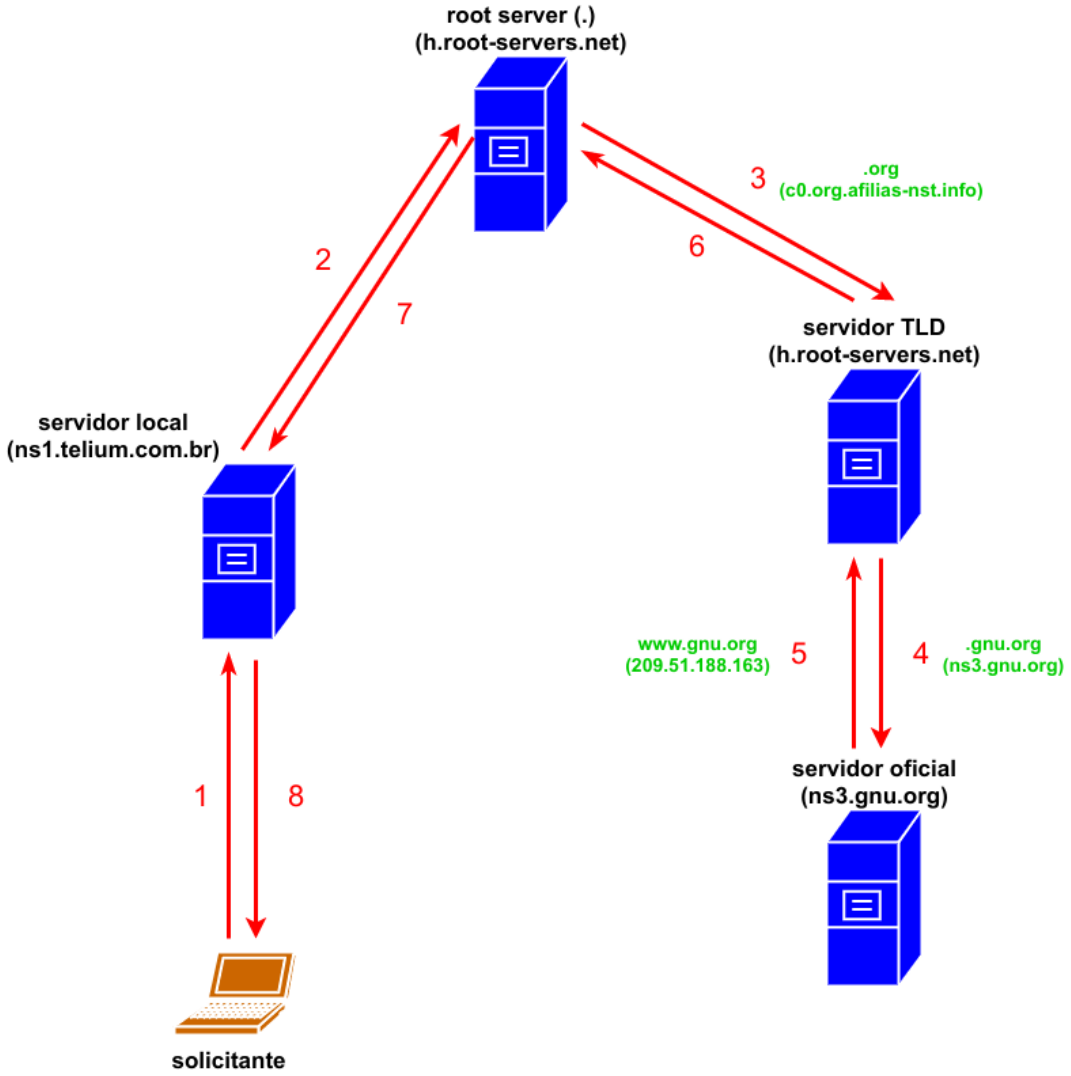


Figura 19 – Consulta a um servidor recursivo.

Servidor Interativo x Recursivo

Um efeito colateral de um servidor de nomes ter que seguir as referências é que seu cache adquire informação sobre domínios intermediários. Em uma LAN isso é interessante já que permite que consultas subsequentes de outros hosts se beneficiem do trabalho anterior do servidor.

Já um servidor de um domínio top-level, como .com ou .edu, não deveria salvar informação requisitada por hosts de vários domínios abaixo, porque isso poderia encher o seu cache rapidamente.

Por esta razão, servidores de nível mais baixo na árvore de nomes são usualmente recursivos enquanto que servidores dos níveis mais altos (top-level ou de segundo nível) não devem sê-los. O lado cliente do DNS (resolver) que vem com a maioria das implementações Unix espera que o servidor de nomes

local seja recursivo. Opções no BIND são providas para desligar a recursão (a partir da versão 4.9.3).

16.1.8.4 Servidor Caching

Uma vez que um servidor aprende um mapeamento, ele o coloca em um cache local. Entradas no cache são sujeitas a temporização, definida pelo TTL dos registros. Servidores TLD são tipicamente armazenados no cache dos servidores de nomes locais. Assim, servidores raiz acabam não sendo consultados com tanta frequência.

O DNS também possui extensões importantes. Atualizações dinâmicas são descritas na RFC 2136, mecanismos de autenticação de atualizações aparecem na RFC 3007, e DNSSEC é definido pelo conjunto de RFCs 4033, 4034 e 4035. Para o Módulo 1, o objetivo é reconhecer que esses recursos existem; a configuração detalhada fica para módulos posteriores.

16.1.8.5 DNS Autoritativo e não-Autoritativo

Uma "resposta autoritativa" vem de um servidor responsável pela zona consultada. Uma "resposta não-autoritativa" normalmente vem de cache; ela pode estar correta, mas não foi respondida diretamente pelo servidor responsável pela zona.

Servidores primários e secundários são autoritativos para as suas zonas, mas não o são sobre informações a respeito de outras zonas mantidas em seus caches.

Servidores apenas recursivos ou de cache não são autoritativos para zonas externas; entretanto, podem reduzir a quantidade de tráfego DNS na rede e melhorar o tempo de resposta das consultas.

É perfeitamente admissível uma máquina ser servidora primária para uma zona e servidora secundária para outras zonas.

16.1.9 O Sistema BIND

Escrito originalmente para o 4.3 BSD Unix, atualmente é mantido pelo Internet Software Consortium (https://www.isc.org/bind/) (CONSORTIUM,). O BIND apresenta três componentes principais:

- **named**: daemon que executa as consultas. A máquina na qual ele executa é chamada de "name server".
- **resolver**: rotinas de biblioteca que são usadas na resolução dos nomes. É o lado cliente do BIND.
- **nslookup, dig e host**: interfaces orientadas a linhas de comando.

Se o `named` não souber a resposta e estiver configurado para recursão, é ele quem consulta outros servidores e coloca as respostas em cache. É ele também o responsável pela operação de "transferência de zona".

16.1.9.1 Biblioteca resolver

O serviço DNS é acessado pelos processos clientes através da biblioteca resolver. É por meio dela que os programas que dependem da resolução de nomes requisitam operações de tradução de nomes em endereços IP e vice-versa. Historicamente, funções como `gethostbyname()` e `gethostbyaddr()` eram comuns; em sistemas atuais, `getaddrinfo()` é preferível por lidar melhor com IPv4 e IPv6.

Na prática, a configuração do cliente se resume em definir o servidor de DNS que será usado para fazer a resolução de nomes e definir a ordem de consulta aos métodos de resolução de nomes.

```
# cat /etc/resolv.conf
search sistemasabertos.com.br
nameserver 10.1.0.101
nameserver 10.1.0.102
```

No UNIX, resolver é parte integrante do "release" de qualquer sistema. Utiliza (é preciso configurar) o arquivo */etc/resolv.conf*.

O resolver consulta a biblioteca NSS (Name Service Switch), que também faz parte da biblioteca Lib C, para determinar o mecanismo a ser utilizado para a resolução de um nome. A resolução de nomes pode fazer uso de diversos mecanismos, entre os quais os mais importantes são:

- ***/etc/hosts***: arquivo do sistema local que contém uma tabela de números IP e nomes associados.
- **DNS**: sistema de resolução de nomes da Internet.
- *NIS*: *Network Information System*, um sistema criado pela Sun para difundir informações administrativas pela rede, como nomes de hosts, informações de usuários, etc.

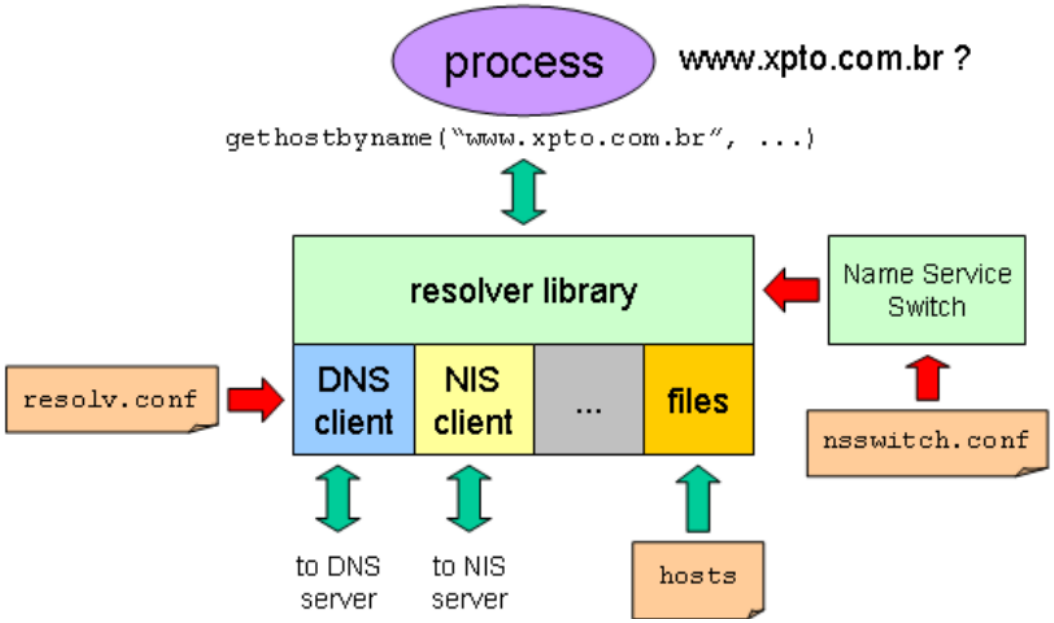


Figura 20 – Biblioteca Resolver.

16.1.9.2 NSS - Name Service Switch

O sistema NSS, criado inicialmente para o Solaris, permite definir a ordem de procura de serviços para a resolução de diversos tipos de nomes em um ambiente Unix. O arquivo Name Service Switch (*/etc/nsswitch.conf*) determina a ordem das buscas realizadas quando uma certa informação é requisitada. Exemplo:

```
passwd: files nis+ nis
shadow: files nis+ nis
group: files nis+ nis
hosts: files dns
ethers: files
services: files
```

A linha "hosts" do exemplo indica que a resolução de nomes de computadores deverá ser feita inicialmente via arquivos locais (*/etc/hosts*) e depois, caso o nome não tenha sido encontrado, através do serviço DNS.

16.1.9.3 resolv.conf

O arquivo */etc/resolv.conf* indica quais servidores de nomes serão usados nas consultas DNS. No arquivo, pode ainda ser utilizada a diretiva `search` ou `domain`, útil em consultas DNS nas quais é informado somente o nome do host em vez do endereço completo (FQDN). Vários domínios podem ser adicionados, mas isso deve ser feito com critério, pois pode causar consultas adicionais para cada sufixo configurado.

Em distribuições Linux atuais, esse arquivo pode ser gerado automaticamente por ferramentas como NetworkManager, systemd-resolved, Netplan ou clientes DHCP. Por isso, antes de editar */etc/resolv.conf* manualmente, o administrador deve verificar quem controla o arquivo.

Na inexistência do arquivo */etc/resolv.conf*, o comportamento normal é assumir que o servidor de nomes é o computador local e que o nome do domínio é obtido a partir do nome da máquina.

16.1.9.4 Consultas e diagnóstico básico

O diagnóstico de DNS deve começar separando três perguntas:

- o cliente sabe qual servidor DNS deve consultar?
- o servidor DNS responde na porta 53?
- a resposta recebida é autoritativa, recursiva, em cache ou um erro?

Exemplos de comandos úteis:

```
resolvectl status
dig www.gnu.org A
dig www.gnu.org AAAA
dig -x 209.51.188.116
dig @8.8.8.8 www.gnu.org
host www.gnu.org
```

O comando `resolvectl status` ajuda a verificar a configuração do resolvedor local em sistemas com systemd-resolved. O comando `dig` mostra detalhes da consulta, como servidor usado, tipo de registro, flags da resposta, TTL e tempo de resposta. O comando `host` é mais simples e útil para testes rápidos.

16.1.10 Configuração do Servidor

A configuração do BIND consiste em vários arquivos, que estão incluídos no arquivo de configuração principal, o `named.conf`. Estes nomes de arquivos começam com *named* porque este é o nome do processo que o BIND executa. Devido à complexidade de configuração do BIND 9, veremos apenas os parâmetros básicos de seu funcionamento. Para configurações mais complexas, consultar o manual em https://bind9.readthedocs.io/en/latest/introduction.html.

named.conf

Contém as configurações primárias para o servidor DNS. No exemplo abaixo, são definidos quais arquivos devem ser carregados.

```
include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
```

named.conf.options

Contém opções globais de configuração do DNS.

- **directory** - Define o diretório de trabalho do servidor. Quaisquer nomes de caminho não absolutos no arquivo de configuração são considerados

relativos a este diretório. O diretório especificado deve ser um caminho absoluto e deve ser gravável pelo usuário efetivo do processo `named`.

- **listen-on port** - Especifica a porta e as interfaces em que o servidor irá escutar.
- **zone** - Especifica a zona pela qual o servidor será responsável. Nesse parâmetro, deve-se indicar se a zona será primária ou secundária por meio do parâmetro **type**, além do arquivo onde os registros da zona estão armazenados por meio do parâmetro **file**.

Exemplo:

```
options {
    directory "/var/cache/bind";
    listen-on port 53 {
        localhost;
    };
};
zone "lab.local" {
    type primary;
    file "/etc/bind/lab.local.db";
};
zone "200.100.10.in-addr.arpa" {
    type primary;
    file "/etc/bind/rev.200.100.10.in-addr.arpa";
};
```

named.conf.default-zones

Contém as definições de zonas locais padrão e dos servidores raiz.

```
// prime the server with knowledge of the root servers
zone "." {
 type hint;
 file "/etc/bind/db.root";
};

// be authoritative for the localhost forward and reverse
// zones, and for broadcast zones as per RFC 1912

zone "localhost" {
 type primary;
 file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
 type primary;
 file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
 type primary;
 file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
 type primary;
 file "/etc/bind/db.255";
};
```

16.1.10.1 Base de Dados do DNS

É um conjunto de arquivos texto mantido pelo administrador do sistema no servidor primário da zona. Itens armazenados na base de dados são denominados resource records (RR). Os tipos e formato dos RR são definidos nas RFCs 1034, 1035 e em extensões posteriores.

O formato dos RR tem a seguinte definição:

<nome> <ttl> IN <tipo> <dados>

- **nome**: nome do objeto
- **ttl**: tempo em segundos que a informação deve permanecer no cache
- **IN**: Internet DNS resource record
- **tipo**: tipo do registro
- **dados**: informação específica ao tipo do registro

Os **tipos**[1] mais comuns são:

- **SOA** (Start of Authority) - Marca o início de uma zona.
- **NS** (Name Server) - Define o(s) servidor(es) de nomes da zona. Permite delegar autoridade para subdomínios de níveis inferiores.
- **A** (Address) - Usado para converter um nome de host em um endereço IPv4.
- **AAAA** (Address) - Usado para converter um nome de host em um endereço IPv6.
- **PTR** (Pointer) - Usado na resolução reversa do endereço, ou seja, converte um IP em um nome de host.
- **MX** (Mail Exchange) - Usado para o direcionamento de correio eletrônico.
- **CNAME** (Canonical Name) - Permite definir um apelido (alias) para um nome de host.

[1] Uma lista mais completa pode ser obtida em <https://en.wikipedia.org/wiki/List_of_DNS_record_types> com suas descrições

- **TXT** (Registro de Texto) - Usado para associar texto a um nome DNS. É comum em validações de domínio, SPF, DKIM, DMARC e outras integrações.

Dentre os **tipos**, o **SOA** possui uma estrutura especial, pois marca o início de uma zona. Existe apenas um registro SOA para cada zona.

```
<zone> [ttl] IN SOA <origin> <contact> {
                                <serial>
                                <refresh>
                                <retry>
                                <expire>
                                <minimum>
}
```

- **zone**: o nome da zona. O "@" referencia a origem da zona definida no arquivo de zona.
- **origin**: nome (FQDN) do servidor primário para a zona.
- **contact**: e-mail do gerente do domínio.
- **serial**: número sequencial usado para indicar a necessidade de atualização dos dados nos servidores secundários.
- **refresh**: tempo em segundos que o servidor secundário vai esperar para testar se precisa atualizar os dados (1 a 6 horas).
- **retry**: tempo em segundos que determina o intervalo de nova tentativa de um pedido de refresh não respondido por um servidor primário (20-60 minutos).
- **expire**: tempo em segundos que o servidor secundário poderá ficar com os dados sem um refresh.
- **minimum**: tempo em segundos que os registros podem ficar no cache de outro servidor.

16.1.11 Laboratorio com Debian

Para o laboratório será usado o bind 9 no Debian.

Observações sobre o servidor DNS

As configurações de DNS quando visualizado são simples, contudo são extremamente rigoroso em sua descrição, tomar cuidados se abriu "{"fecha-la corretamente com "}"; no final de cada comando usar o ";".

Ao iniciar o serviço é altamente recomendável verificar no log */var/log/syslog* se o serviço foi carregado corretamente, é no log que terá informação para solução de possiveis problema.

Instalando o bind9

```
apt update
apt-get -y install bind9 vim
```

Configurando Named Local

Será configurado os parâmetros da base de dados do domínio a ser resolvido

```
vim /etc/bind/named.conf.local
mkdir -p /etc/bind/zones
```

```
zone "lab.local" {
        type master;
        file "/etc/bind/zones/lab.local.db";
        };

zone "200.100.10.in-addr.arpa" {
     type master;
     file "/etc/bind/zones/rev.200.100.10.in-addr.arpa";
};
```

Configurando Named Options

Será configurado como o bind ira operar, como interface, porta ...

```
vim /etc/bind/named.conf.options
```

```
options {
 directory "/var/cache/bind";
        listen-on port 53 { localhost; 0.0.0.0/0; };
 allow-query { localhost; 10.100.200.0/24; };
 forwarders { 127.0.0.1; 10.100.200.2; };
 dnssec-validation auto;
 auth-nxdomain no;    # conform to RFC1035
 recursion yes;
};
```

Configurando base de dados de nomes

Será configurado a base de dados da resolução de nome

```
vim /etc/bind/zones/lab.local.db
```

```
$TTL 86400
@ IN SOA labdns.lab.local. root.localhost. (
         1  ; Serial
    604800  ; Refresh
     86400  ; Retry
   2419200  ; Expire
     86400 ) ; Negative Cache TTL
;
@ IN NS labdns.lab.local.

; Primary DNS Controller for LAB
labdns IN  A 10.100.200.1

; Clients for Chef
chef IN A 10.100.200.2
node1 IN A 10.100.200.3
node2 IN A 10.100.200.4
node3 IN A 10.100.200.5
```

Configurando base de dados de reverso

Será configurado a resolução do reverso.

```
vim /etc/bind/zones/rev.200.100.10.in-addr.arpa
```

```
$TTL 86400
@  IN  SOA   lab.local. root.lab.local. (
                        21;
                        28800;
                        604800;
                        604800;
                        86400 )

;Name server information in LAB
@          IN    NS     labdns.lab.local.
labdns     IN A 10.100.200.1

; Client machines for LAB
2      IN    PTR  chef.lab.local
3          IN    PTR node1.lab.local
4          IN    PTR node2.lab.local
5          IN    PTR node3.lab.local
```

16.2 SSH

SSH, abreviação de *Secure Shell*, é o protocolo mais usado para administração remota de servidores Linux. Ele permite abrir uma sessão de terminal em outra máquina, executar comandos, copiar arquivos e encaminhar conexões de rede usando um canal criptografado.

Em um ambiente real, o administrador raramente trabalha sentado em frente ao servidor. O acesso normalmente é feito pela rede, a partir de uma estação de trabalho, de uma VPN administrativa, de um bastion host ou de outro servidor. Por isso, antes de tratar o SSH apenas como um comando, é importante entender o modelo envolvido.

16.2.1 Objetivos

Ao final desta seção, o aluno deverá ser capaz de:

- diferenciar cliente SSH, servidor SSH e serviço *sshd*;
- acessar um servidor Linux usando usuário, endereço e porta;
- compreender o papel de chaves públicas, chaves privadas e *known_hosts*;
- instalar uma chave pública em *authorized_keys*;
- copiar arquivos com *scp* e *sftp*;
- diagnosticar erros comuns de conexão, autenticação e chave de host;
- aplicar cuidados básicos de segurança em um servidor SSH.

16.2.2 Pré-requisitos

Para acompanhar os exemplos, é recomendado que o aluno já compreenda:

- usuários e permissões básicas em Linux;
- endereçamento IP, porta TCP e modelo cliente-servidor;
- uso básico do shell;
- leitura inicial de logs com *journalctl* ou arquivos em */var/log*.

16.2.3 Cliente, servidor e porta 22

Em uma conexão SSH existem pelo menos dois componentes:

- **cliente SSH**: programa usado para iniciar a conexão. O comando mais comum é *ssh*;
- **servidor SSH**: serviço que escuta conexões remotas. Em sistemas Linux, normalmente é o serviço *sshd*;
- **porta TCP**: por padrão, o SSH usa a porta TCP 22.

O fluxo básico é:

1. o cliente tenta conectar no endereço e porta do servidor;
2. o servidor apresenta sua chave de host;
3. o cliente verifica se aquela chave já é conhecida;
4. cliente e servidor negociam algoritmos criptográficos;
5. o usuário se autentica por senha, chave pública ou outro método configurado;
6. após a autenticação, uma sessão remota é aberta.

A forma mais simples de conexão é:

```
ssh usuario@servidor
```

Exemplo usando endereço IP:

```
ssh lab@192.168.56.20
```

Se o servidor SSH estiver em uma porta diferente da porta 22:

```
ssh -p 2222 lab@192.168.56.20
```

Atenção

Alterar a porta padrão pode reduzir ruído de varreduras automatizadas, mas não substitui autenticação forte, atualização do sistema, firewall, logs e boas permissões de chaves.

16.2.4 Instalação do servidor SSH

Em muitas instalações de servidor, o cliente SSH já vem instalado. O servidor SSH pode ou não estar presente, dependendo do perfil escolhido na instalação do sistema operacional.

Em Debian e Ubuntu:

```
sudo apt update
sudo apt install openssh-server
sudo systemctl enable --now ssh
sudo systemctl status ssh
```

Em Rocky Linux, RHEL e derivados:

```
sudo dnf install openssh-server
sudo systemctl enable --now sshd
sudo systemctl status sshd
```

Para verificar se a porta está em escuta:

```
ss -ltnp | grep ':22'
```

Em distribuições com firewall ativo, pode ser necessário liberar o serviço. Em Rocky Linux, por exemplo:

```
sudo firewall-cmd --add-service=ssh --permanent
sudo firewall-cmd --reload
```

16.2.5 Primeiro acesso e *known_hosts*

No primeiro acesso a um servidor, o cliente SSH normalmente exibe uma mensagem informando que a autenticidade do host ainda não é conhecida. Isso ocorre porque a chave pública do servidor ainda não está registrada no arquivo *$HOME/.ssh/known_hosts* do cliente.

Exemplo:

```
The authenticity of host '192.168.56.20' can't be established.
ED25519 key fingerprint is SHA256:....
Are you sure you want to continue connecting (yes/no)?
```

Ao responder *yes*, o cliente grava a chave de host. Em acessos posteriores, se a chave do servidor mudar, o cliente emitirá um alerta. Essa proteção ajuda a identificar troca real do servidor, reinstalação do sistema ou tentativa de interceptação.

Cuidado operacional

Não remova uma entrada de *known_hosts* automaticamente sem entender o motivo do alerta. Em laboratório isso pode ocorrer após reinstalar uma VM. Em produção, pode indicar troca indevida de servidor ou ataque de *man-in-the-middle*.

16.2.6 Autenticação por senha

Quando a autenticação por senha está habilitada no servidor, o acesso pode ser feito informando a senha do usuário remoto:

```
ssh lab@192.168.56.20
```

Esse método é simples e útil em laboratório, mas exige cuidado em servidores expostos. Senhas fracas, usuários previsíveis e serviços abertos para a Internet são alvos comuns de tentativas automatizadas de login.

16.2.7 Chaves SSH

A autenticação por chave usa um par de arquivos:

- **chave privada**: fica somente com o usuário, no cliente;
- **chave pública**: pode ser copiada para servidores nos quais o usuário terá acesso.

A chave privada não deve ser enviada por e-mail, compartilhada em chat, copiada para servidores de terceiros ou armazenada em locais públicos. A chave pública, por outro lado, é a parte que será instalada no servidor.

Atualmente, para uso geral, uma chave *ed25519* costuma ser uma boa opção:

```
ssh-keygen -t ed25519 -C "lab@curso-linux"
```

Em ambientes que exigem RSA por compatibilidade:

```
ssh-keygen -t rsa -b 4096 -C "lab@curso-linux"
```

Durante a criação, o comando perguntará onde salvar a chave e se deseja proteger a chave privada com uma frase de senha. Para máquinas pessoais ou estações administrativas, é recomendável usar frase de senha.

Exemplo de arquivos criados:

```
/home/lab/.ssh/id_ed25519
/home/lab/.ssh/id_ed25519.pub
```

O arquivo sem *.pub* é a chave privada. O arquivo terminado em *.pub* é a chave pública.

16.2.8 Permissões do diretório *.ssh*

O SSH é rigoroso com permissões. Se o diretório ou os arquivos estiverem abertos demais, a autenticação por chave pode falhar.

No cliente:

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/id_ed25519
chmod 644 ~/.ssh/id_ed25519.pub
```

No servidor, para o usuário que receberá o acesso:

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

O arquivo *authorized_keys* contém as chaves públicas autorizadas a acessar aquela conta. Ele não deve ficar gravável por outros usuários.

16.2.9 Instalando a chave pública no servidor

A forma mais simples, quando o acesso por senha ainda funciona, é usar *ssh-copy-id*:

```
ssh-copy-id lab@192.168.56.20
```

Se a chave não estiver no nome padrão:

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub lab@192.168.56.20
```

Também é possível fazer manualmente. No servidor:

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
cat id_ed25519.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

Depois disso, o acesso pode ser testado:

```
ssh lab@192.168.56.20
```

Se houver mais de uma chave no cliente, pode-se indicar uma chave específica:

```
ssh -i ~/.ssh/id_ed25519 lab@192.168.56.20
```

16.2.10 Agente SSH

Quando a chave privada possui frase de senha, o usuário precisa digitá-la para desbloquear a chave. O *ssh-agent* permite manter a chave desbloqueada na sessão do usuário, evitando digitar a frase de senha a cada conexão.

Para adicionar a chave ao agente:

```
ssh-add ~/.ssh/id_ed25519
```

Para listar chaves carregadas:

```
ssh-add -l
```

Para listar as chaves públicas carregadas:

```
ssh-add -L
```

Para remover uma chave:

```
ssh-add -d ~/.ssh/id_ed25519
```

Para remover todas as chaves da sessão:

```
ssh-add -D
```

Se aparecer a mensagem:

```
Could not open a connection to your authentication agent.
```

inicie o agente na sessão atual:

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
```

Atenção

O encaminhamento de agente com *ssh -A* deve ser usado com cuidado. Ele é útil em saltos administrativos, mas expõe o agente local ao host remoto durante a sessão. Em servidores não confiáveis, evite essa opção.

16.2.11 Copiando arquivos com *scp*

O *scp* copia arquivos usando SSH como transporte. Para enviar um arquivo local para o servidor:

```
scp relatorio.txt lab@192.168.56.20:/home/lab/
```

Para copiar um arquivo do servidor para a máquina local:

```
scp lab@192.168.56.20:/var/log/syslog ./syslog-servidor.txt
```

Para copiar um diretório recursivamente:

```
scp -r scripts/ lab@192.168.56.20:/home/lab/
```

Em ambientes modernos, algumas distribuições também recomendam *sftp* ou *rsync* para cópias mais controladas, especialmente quando há muitos arquivos.

16.2.12 Transferência interativa com *sftp*

O *sftp* fornece uma sessão interativa de transferência de arquivos por SSH. Ele não usa o serviço FTP tradicional.

```
sftp lab@192.168.56.20
```

Comandos comuns dentro do *sftp*:

```
pwd
lpwd
ls
lls
put arquivo.txt
get arquivo-remoto.txt
exit
```

O prefixo *l* indica operação local. Assim, *pwd* mostra o diretório remoto e *lpwd* mostra o diretório local.

16.2.13 Encaminhamento de portas

O SSH também pode encaminhar conexões TCP. Esse recurso é útil para acessar serviços internos por meio de um servidor intermediário, desde que a política da organização permita esse uso.

Encaminhamento local:

```
ssh -L 8080:127.0.0.1:80 lab@192.168.56.20
```

Nesse exemplo, conexões feitas no cliente para *localhost:8080* serão encaminhadas pelo túnel SSH até o servidor, que tentará acessar *127.0.0.1:80* do ponto de vista dele.

Encaminhamento remoto:

```
ssh -R 9000:127.0.0.1:22 lab@192.168.56.20
```

O encaminhamento remoto depende da configuração do servidor e deve ser usado com autorização. Em produção, túneis mal controlados podem contornar políticas de firewall e auditoria.

16.2.14 Acesso gráfico com X11

O SSH pode encaminhar aplicações gráficas usando X11:

```
ssh -X lab@192.168.56.20
```

Esse recurso depende de servidor gráfico no cliente, pacotes adequados e configuração do servidor SSH. Para administração básica de servidores, o uso mais comum continua sendo o terminal. Em ambientes modernos, muitas tarefas gráficas são substituídas por interfaces web, consoles remotos ou automação.

16.2.15 Boas práticas básicas

Algumas práticas ajudam a reduzir riscos:

- usar chaves SSH com frase de senha para contas administrativas;
- desabilitar login remoto direto do usuário *root*, quando houver contas administrativas com *sudo*;
- manter o OpenSSH atualizado pelos repositórios da distribuição;
- liberar a porta SSH apenas para redes ou origens necessárias;
- auditar tentativas de login;
- remover chaves públicas de usuários que não precisam mais de acesso;
- considerar ferramentas como *fail2ban* em servidores expostos, quando fizer sentido operacional.

O arquivo principal de configuração do servidor costuma ser:

```
/etc/ssh/sshd_config
```

Após alterar a configuração, valide antes de reiniciar:

```
sudo sshd -t
```

Em Debian e Ubuntu:

```
sudo systemctl reload ssh
```

Em Rocky Linux, RHEL e derivados:

```
sudo systemctl reload sshd
```

Cuidado operacional

Ao administrar um servidor remoto, mantenha uma sessão SSH aberta antes de reiniciar ou recarregar o serviço. Se a nova configuração estiver incorreta, essa sessão pode ser a forma mais rápida de corrigir o erro sem perder acesso.

16.2.16 Diagnóstico de problemas comuns

16.2.16.1 Conexão recusada

Mensagem típica:

```
ssh: connect to host 192.168.56.20 port 22: Connection refused
```

Possíveis causas:

- serviço SSH parado;
- porta diferente da informada;
- firewall local rejeitando a conexão;
- serviço não instalado.

Comandos úteis no servidor:

```
systemctl status ssh
systemctl status sshd
ss -ltnp | grep ':22'
```

16.2.16.2 Timeout

Mensagem típica:

```
ssh: connect to host 192.168.56.20 port 22: Connection timed out
```

Possíveis causas:

- endereço IP incorreto;
- rota ausente;
- firewall descartando pacotes;
- servidor desligado;
- problema de conectividade entre redes.

Comandos úteis no cliente:

```
ip route
ping 192.168.56.20
nc -vz 192.168.56.20 22
```

16.2.16.3 Permissão negada

Mensagem típica:

```
Permission denied (publickey,password).
```

Possíveis causas:

- usuário incorreto;
- senha incorreta;
- chave pública ausente no servidor;
- permissões incorretas em *.ssh* ou *authorized_keys*;
- autenticação por senha desabilitada no servidor.

Para depurar pelo cliente:

```
ssh -v lab@192.168.56.20
```

Para consultar logs no servidor:

```
sudo journalctl -u ssh --since "10 minutes ago"
sudo journalctl -u sshd --since "10 minutes ago"
```

16.2.16.4 Chave de host alterada

Mensagem típica:

```
WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
```

Em laboratório, isso costuma ocorrer após reinstalar a VM. Em produção, deve ser investigado.

Após confirmar que a mudança é legítima, remova a entrada antiga:

```
ssh-keygen -R 192.168.56.20
```

Em seguida, conecte novamente e aceite a nova chave somente se ela for esperada.

16.2.17 Laboratório guiado

Objetivo: configurar acesso SSH entre duas máquinas Linux ou, na falta de duas VMs, testar cliente e servidor na mesma máquina usando *localhost*.

16.2.17.1 Cenário

- Cliente: estação do aluno;
- Servidor: VM Debian 12, Ubuntu 24.04 LTS ou Rocky Linux 9;
- Usuário de teste: *lab*;
- Rede: comunicação IP entre cliente e servidor.

16.2.17.2 Etapa 1: preparar o servidor

No servidor, instale e ative o serviço SSH conforme a distribuição:

```
# Debian/Ubuntu
sudo apt update
sudo apt install openssh-server
sudo systemctl enable --now ssh

# Rocky/RHEL
sudo dnf install openssh-server
sudo systemctl enable --now sshd
```

Verifique o endereço IP:

```
ip addr
```

16.2.17.3 Etapa 2: testar conexão por senha

No cliente:

```
ssh lab@IP_DO_SERVIDOR
```

Saída esperada: após aceitar a chave de host, o sistema solicita a senha do usuário *lab* e abre um shell remoto.

16.2.17.4 Etapa 3: gerar chave no cliente

No cliente:

```
ssh-keygen -t ed25519 -C "lab@curso-linux"
```

Confirme se os arquivos foram criados:

```
ls -l ~/.ssh
```

16.2.17.5 Etapa 4: instalar a chave pública

No cliente:

```
ssh-copy-id lab@IP_DO_SERVIDOR
```

Teste novamente:

```
ssh lab@IP_DO_SERVIDOR
```

Saída esperada: o acesso deve ocorrer por chave. Se a chave tiver frase de senha, ela será solicitada no primeiro uso ou pelo agente SSH.

16.2.17.6 Etapa 5: copiar arquivo

No cliente:

```
echo "teste de copia via ssh" > teste-ssh.txt
scp teste-ssh.txt lab@IP_DO_SERVIDOR:/home/lab/
```

No servidor:

```
cat /home/lab/teste-ssh.txt
```

16.2.17.7 Etapa 6: observar logs

No servidor:

```
sudo journalctl -u ssh --since "15 minutes ago"
sudo journalctl -u sshd --since "15 minutes ago"
```

Use o comando correspondente ao nome do serviço existente na distribuição.

16.2.18 Exercícios de revisão

1. Qual é a diferença entre o comando *ssh* e o serviço *sshd*?
2. Por que a chave privada não deve ser copiada para o servidor?
3. Qual é a função do arquivo *known_hosts*?
4. Quais permissões são recomendadas para */.ssh* e *authorized_keys*?

5. Como investigar uma falha do tipo *Connection refused*?

6. Em que situação o alerta *REMOTE HOST IDENTIFICATION HAS CHANGED* pode ser legítimo em laboratório?

16.2.19 Resumo

O SSH é uma ferramenta central para administração Linux. Ele combina acesso remoto, autenticação, criptografia, cópia de arquivos e recursos de túnel em um mesmo conjunto de ferramentas. Para um administrador júnior, o ponto mais importante é dominar o fluxo básico: serviço ativo no servidor, porta acessível, usuário correto, chave de host conhecida, autenticação válida, permissões corretas e leitura de logs quando algo falha.

Referências

CONSORTIUM, I. S. *Bind 9*. Disponível em: <https://www.isc.org/bind/>. Acesso em: 2020-12-05. 198

DALHEIMER, M. K. et al. *Running linux*. 4th. ed. [S.l.]: O'Reilly & Associates, Inc, 2002. 692 p. ISSN 08981221. ISBN 0596002726. 53

Free Software Foundation. *GNU Awk*. 5.1. ed. Boston, USA: [s.n.], 2020. ISBN 1882114280. Disponível em: <https://www.gnu.org/software/gawk/manual/gawk.pdf>. 92

JONES, T. M. *Anatomy of linux journaling file systems*. 2008. 1–9 p. Disponível em: <http://www.ibm.com/developerworks/linux/library/l-journaling-filesystems/index.html>. 23

MUSăLOIU-E., R. A Visual Expedition Inside the Linux File Systems. 2009. Disponível em: <http://cs.jhu.edu/{~}razvanm/fs-expedition/p.100dpi.p>. 23

ROOT-SERVERS.ORG. 2020. Disponível em: <https://root-servers.org/>. Acesso em: 2020-12-05. 189

STALLMAN, R. The GNU operating system and the free software movement. 1999. Disponível em: <https://www.gnu.org>. Acesso em: 2020-01-20.

The Linux Foundation. *Filesystem Hierarchy Standard Specifications Archive*. 2015. Disponível em: <https://refspecs.linuxfoundation.org/fhs.shtml>. 40