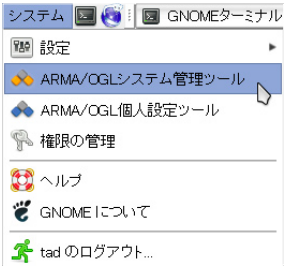
第3章 システムの設定

3.1 ARMA / OGL 管理ツール

ARMA / OGL 管理ツール（以下管理ツール）は、インストーラで行った各種設定や各種ハードウェア・ソフトウェア設定などを統合して行えるツールです。可能な限り広い場面で設定が可能なように管理ツールには複数の起動法があります。

3.1.1 GNOME デスクトップから開始する場合

GNOME デスクトップから開始する場合は下記のようにパネルから開始してください。



ルートパスワードを尋ねられますので入力してください。

3.1.2 KDE デスクトップから開始する場合

KDE デスクトップから開始する場合は、「K」-「アプリケーション」-「設定」から下記の項目を選択してください。

ルートパスワードを尋ねられますので入力してください。

3.1.3 ランレベル4で開始する場合

ランレベル4で起動する場合は、GRUBで起動設定を書き換えます。

```
   GRUB  version 0.97  (638K lower / 2086400K upper memory)

+-------------------------------------------------------------------+
| ARMA 3.0 (2.6.31.6-smp)                                           |
| Windows                                                           |
|                                                                   |
|                                                                   |
|                                                                   |
+-------------------------------------------------------------------+
   Use the ↑ and ↓ keys to select which entry is highlighted.
   Press enter to boot the selected OS, 'e' to edit the
   commands before booting, or 'c' for a command-line.

 The highlighted entry will be booted automatically in 10 seconds.
```

起動時に、このようなGRUBメニューが表れたら［↑］［↓］でARMAの行を選択し、［e］を押して修正画面に入ります。

```
| root (hd0,1)                                                      |
| kernel /boot/vmlinuz-2.6.31.6-smp root=/dev/sda2 vga=0x301        |
```

ここで、kernel行を選択して、また［e］を押して行修正モードに入り、行末に4を追加し、［Enter］を押します。

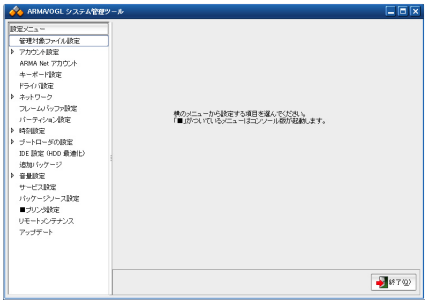
```
| root (hd0,1)                                                      |
| kernel /boot/vmlinuz-2.6.31.6-smp root=/dev/sda2 vga=0x301 4      |
```

ここで［b］を押せば、ランレベル4でARMAが起動し、管理ツールが起動します。

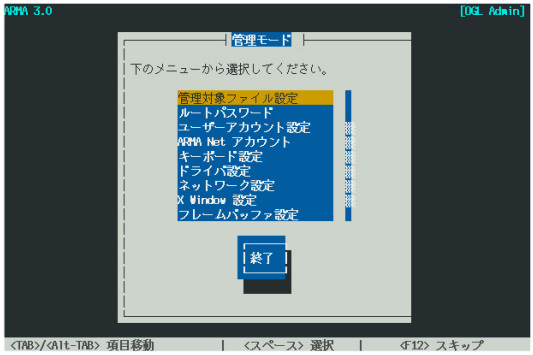
カーネルの引数に「4」を追加する、というのが作業の意味合いになります。

3.1.4 管理ツール

いずれかの方法で起動すれば、次のように管理ツールのスタート画面が表示されます。



コンソール等から起動した場合等は下記のようなスタート画面が表示されます。



X版でも非X版でも設定のロジックは共通です。またごく一部のメニュー(プリンタ設定とブートエントリ設定)のみは非X版だけに設定項目があります。

管理ツールを起動した後はインストーラと同様の操作で各種ツールを使うことができます。なおこの後の章では、各分野の設定についてシステム管理の背景となる知識について詳説します。

管理ツールは elk というインタープリタを使用して実行されています。このインタープリタには X 版と非 X 版があり、X Window System 下でもパッケージの整合性が不十分な場合などでは自動的に非 X 版を起動します。例えばアップデート中などがこれに該当します。

3.2 パッケージ管理

3.2.1 パッケージ…の説明の前に

突然ですが、「パッケージ」とは何かを考えるためにソフトウェアをごはんに例えてみましょう。使える状態のソフトウェアをほかほかごはんだとすると、「パッケージ」はちょうど冷凍されてパックに入ったごはんに相当します。

しかし、パックごはんが出てくるまでは、ごはんはお米を炊いて食べるものでした。もちろん、炊いただけで炊飯器から直接食べるわけにはいきませんので、しゃもじで茶碗によそわなければいけません。ソフトウェアの世界でもこのモデルは同じで、お米は「ソースファイル」、炊くことは「コンパイル」、よそうことは「インストール」に相当します。

お米から作れば、とぎ方や水加減や炊き方次第でいくらでも自分好みのごはんができます。しかし、毎度毎度ごはんを炊くのは少々面倒ですし、水加減を失敗して食べられないごはんができてしまうかもしれません。その点、パックごはんは出来合いですがチンすれば誰でもすぐに食べられ、忙しい人にはとても重宝するものです。

それでは、たとえ話はここまでにして、実際のソフトウェアのパッケージがどのようなものであるかを説明していきましょう。

3.2.2 パッケージ

Linux に限らず、多くの OS は静的な情報をファイルという単位で管理しています。しかし、実際に使う１本のソフトウェアは１つのファイルではなく、あちこちのディレクトリに分散した多くのファイルの集合体です。例えばごく小さなソフトウェアにも、以下のようなファイルがあります。

/usr/bin/hoge	実行ファイル
/usr/lib/libhoge.so	共有ライブラリファイル
/usr/man/man1/hoge.1.gz	オンラインマニュアル
/usr/share/doc/hoge/README	添付文書（ReadMe）

このため、ファイル単位でソフトウェアを管理するには、ソフトウェアをインストールしたり削除したりするたびにあちこちのディレクトリでファイル操作をしなくてはなりません。大規模なソフトウェアでは数千ものファイルが数十のディレクトリに散らばっているので、このファイルを過不足なく削除することはかなり面倒な作業になります。もちろん、ファイルを削除し忘れれば、ディスクを浪費するばかりでなくシステムの見通しも悪くなってしまいますし、逆に他のソフトウェアのファイルまで誤って削除してしまえば、システムに重大な悪影響を及ぼしかねません。

ところで、UNIX の世界では伝統的にソフトウェアはソースで配布されてきました。ソースを自分の環境に合わせて調整してコンパイルすれば、UNIX 系 OS で共通のソフトウェアが使えるというのが大きな利点なのですが、誰もが経験と勘を要するコンパイルという作業をこなせるわけではありませんし、できる人でもコンパイルが面倒になることが少なくありません。

Linux ディストリビューションのパッケージシステムは、主にこのような煩雑さを解消するために生まれました。このシステムの元では、ソフトウェアのソースはコンパイル済みの構成ファイルを１つのアーカイブファイルに固めた「パッケージ」の形で配布されます。このパッケージを、パッケージマネージャにかけてインストール・削除などの指示を与えれば、ファイルの配置・回収などの作業は全てマネージャが行ってくれます。

つまり、パッケージシステムとは、ソフトウェアをファイル単位ではなく「ソフトウェア１本」を単位として扱えるようにし、ソフトウェアを使えるようにするまでの手間を最低限に抑えるための機構と言えるでしょう。

❗ ソースコード形態で配布されているソフトウェアをパッケージ単位に編集する作業をする人を「メインテナ」と呼びます。ここでは詳しく述べませんが、パッケージの作成規則、リリースに関する規則、メインテナになるための規則などが、Debian プロジェクトによっていろいろと決められています。

3.2.3 Debian パッケージシステムとは？

Linux ディストリビューションの多くは Debian プロジェクトが開発した「deb 形式」か RedHat 社が開発した「RPM 形式」のどちらかのパッケージシステムを採用しています。ARMA は deb 形式を採用しています。

この Debian パッケージシステムは以下のような機能をもっています。

パッケージの依存関係の解決

大半のソフトウェアは単体では動かず、他のソフトウェアや共有ライブラリがあってはじめて動くようになっています。このほか、必須ではなくてもセットで使うともっと能力を発揮できるソフトウェアの組み合わせや、逆に互いに共存できない組み合わせもあります。これらのソフトウェア同士の相性＝「依存」関係は、以下のように分類されてパッケージに記録され、パッケージマネージャはこれを元に依存問題を解決しています。

依存	A depends B	A の動作には B が必要である。
推奨	A recommends B	A の動作には B がないと不便である。
提案	A suggests B	A の動作には B があればより便利である。
衝突	A conflicts B	A と B は共存できない。
置換	A replaces B	A は B に代わって動くことができる。
提供	A provides B	A は B の機能も提供する。

ネットワークから最新版をダウンロード

パッケージの入手元として DVD-ROM, ハードディスクなどのローカルディスクだけでなく、FTP, HTTP, NFSなどのサイトも選択できます。これにより、簡単にパッケージの配布元（オモイカネ , Debian Project 等）から最新版をダウンロードしてアップグレードができます。

設定ファイルの保護

大半の設定ファイルは / etc 以下に保存されています。

パッケージをアップグレードするときに、ユーザが書き換えた設定ファイルは保護され、無断では書き換えられません。また、パッケージを削除するときに設定ファイルのみを残しておくこともできます。

3.2.4 deb ファイルの命名規則

debパッケージはその名の通り拡張子がdebのファイルで、以下の規則によってファイル名が付けられています。

```
<パッケージ名>_<バージョン>_<機種>.deb
<パッケージ名>_<バージョン>-<リビジョン>_<機種>.deb
```

ソフトウェアの deb パッケージ化の責任者をメインテナ（Maintainer）と言います。

パッケージ名は bash, gcc などのソフトウェアの名前です。バージョンはソフトウェア作者（以下「原作者」）が付けるソフトウェアの「版」を表す番号です。これに対して、リビジョンはソフトウェアを deb パッケージ化する際に付ける、「deb パッケージとしての版数」で、同じバージョンのソフトウェアでもパッケージ化の工程に修正を入れるたびに数字を上げていきます。機種（アーキテクチャ）はパッケージの対応機種を表し、例えば x86 用は i386・機種依存しないものは all です。以上のような規則により hoge_1.2-3_i386.deb なら、ファイル名から hoge のバージョン 1.2・リビジョン 3 の PC 互換機用パッケージであると分かります。

> バージョン番号をどう付けるは原作者の意向次第ですが、UNIX 系では慣習的に "." で区切った 0 以上の整数を使う例が多いようです。例えば、バージョン 1.0（いち・てん・れい）を最初の正式版として、以下 1.1 , 1.2…となります。1.9 の次も 1.10（いち・てん・じゅう）となり、1.11 , 1.12…と続けます。そして、全面改良や設計変更などのメジャーバージョンアップがあると、晴れて 2.0 となります。このほかに、一通りの完成（1.0）に到達していないことを示すために 0.x というバージョンを使ったり、"." を 2 つ以上使って、1.2.34 のようにする流儀などもよく見かけられます。
>
> リビジョン番号はメインテナが付ける番号で、通常は 1 から始まる整数や 20011028 などの日付を入れます。一般的には、1.1-1，1.1-2…，

1.1-10… と続き、原作者がバージョンアップすると1.2-1とリビジョンを1に戻すようになっています。また、メインテナ以外の人が作ったパッケージをDebianで公開されるNMU（Non Maintainer Update）の場合は、リビジョンに‘‘.’’を付けます。例えば、1.1-1にNMUがあると1.1-1.1，1.1-1.2…となり、メインテナが復帰すると1.1-2になります。

3.2.5 パッケージマネージャ

Debianには2種類のパッケージマネージャがあります。一つは低水準パッケージマネージャdpkg、もう一つは高水準パッケージマネージャaptです。dpkgはそれ自体コマンド名でもありますがaptの方はapt-get、apt-cacheといった複数のコマンドから構成されています。〜 .debというファイル名を直接指定してパッケージをインストールする処理などはdpkgが、ダウンロードサイトの情報を調べたり、依存関係の解析をしたりする処理はaptが担当しています。

dpkgはDebian PacKaGeから、aptはA Package Toolからきています。

個々の操作については、たいていdpkgとaptで同じことができますが、インストール元や依存関係を考慮してくれる分、普通はaptの方が便利でしょう。ここではaptを中心に、必要に応じてdpkgも使うという立場でdebパッケージの管理について解説していくことにします。

実際のところaptは内部でdpkgを呼び出しています。このため「aptはdpkgのフロントエンドである」というふうにも呼ばれます。

3.2.6 aptを使うための準備

aptの設定ファイルは/etc/aptにまとめられています。通常このファイルは管理ツール（ogl-admin）のパッケージソース設定によって管理されてお下記のような形式になっています。

```
deb https://WS30XXXXXXX:ZZZZZZZ@www.omoikane.co.jp/
arma_3.0_updates/deb/ogl ./
deb-src https://WS30XXXXXXX:ZZZZZZZ@www.omoikane.co.jp/
arma_3.0_updates/deb/ogl ./
```

（実際には折り返しはなく一行です）

ARMA以外に、例えばDebianサイトからアプリケーションを追加したい場合などは下記のような行を追加します。

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

（実際には折り返しはなく一行です）

上記の例では、Debianのパッケージを FTP サイトからダウンロードするよう指定しているのですが、ここで各行が下のA），B）いずれかの書式になっていることに注目してください。

A）deb <ルート> <ディストリビューション> <コンポーネント（複数可）>
B）deb <ルート> <ディレクトリ>

Packages.gz 以外に Release というファイルもダウンロードしますが、ここでは特に触れません。

各行頭の deb はパッケージの入手元を指定していることを示す識別子です。apt は deb から始まる行を見付けると、あるディレクトリ（具体的な位置は後述します）から Packages.gz というファイルをダウンロードします。これは収録パッケージの目録のようなもので、apt のパッケージデータベースの元になります。

ここから後は、A）, B）で書き方が違います。A）は多数の deb パッケージを整理して配置するのに適した構造で、Debian などの大規模なサイトが多数の deb パッケージを配布する場合に使うことが多いようです。これに対して B）は比較的単純な構造で、ソフトウェアの開発者やユーザが自作の deb パッケージをいくつか配布するのに適しています。

まずは、A）について説明します。

<ルート> には、Debian や ARMA があるサーバの「ルート」となる URL を指定します。例えば ftp.debian.org には /debian 以下に Debian が置かれているので、ftp://ftp.debian.org/debian/ となります。ローカルマシンのディスクからパッケージを入手する場合は file: に続けて、例えば file:/cdrom/debian/ のように指定します。

正確に言うと、ディストリビューションには、<ルート>/dists にあるディレクトリを指定できます。

<ディストリビューション> には、Debian のバージョン、具体的に言えば stable, unstable, testing, frozen のいずれかを指定します。woody, sarge などのコードネームを指定することもできますが、これは Debian の開発が進むに従って［開発版→安定版→廃止］とその性格が変わってしまうのでお勧めできません。

正確に言うと、コンポーネントには、<ルート>/dists/<ディストリビューション> にあるディレクトリを指定できます。

<コンポーネント> には、通常 main, contrib, non-free の３つを指定します。以上のような A）方式では、さらに apt が自動的にマシンの機種を判断して Packages.gz を読み込むディレクトリを決定します。例えば、先の追加の sources.list 例では以下の URL になります。

```
ftp://ftp.debian.org/debian/dists/unstable/main/binary-i386/
ftp://ftp.debian.org/debian/dists/unstable/contrib/binary-i386/
ftp://ftp.debian.org/debian/dists/unstable/non-free/binary-i386/
```

B）方式は、Packages.gz を読み込むディレクトリを <ルート> に続けて直接指定し、/ を付けてディレクトリであることを明示しておくだけです。apt はこの / の有無で A）と B）を判別するので、/ の有無は明確にする必要があります。

なお、この Debian サイトの追加の例で挙げたサーバは全て１次配布元ですので、サーバに負荷を集中させないためにも、実際の sources.list には、別のサーバ（ミラーサーバ）を指定するようにしてください。国内の Debian のミラーサーバの情報も Debian プロジェクトの WWW サイトで公開されていますので参考にしてください。

ご注意

ARMA Net プライベートリポジトリ以外からのソフトウェアを導入される場合、外部導入部分については基本的にサポートの対象外となりますので、自己責任で行っていただきますようお願い致します。

また Debian のパッケージをインストールされる場合は直接バイナリを入れるよりも、以下のように一旦ビルドをした方がシステムの整合性が保たれやすくなります。

```
# apt-get source -b < ソースパッケージ名 >
# dpkg -i 〜 .deb
```

3.2.7 apt と dpkg の使い方

apt とはいくつかのコマンド群の総称ですが、このうちよく使うコマンドは apt-get, apt-cache の２つです。apt-get はパッケージのインストールや削除、パッケージデータベースの更新など、実際にファイルの書き換えを伴う作業を担当します。これに対し、apt-cache はパッケージデータベースを利用した情報をユーザに提供するコマンドで、ファイルは一切書き換えません。dpkg もいくつかのコマンド群の総称ですが、パッケージマネージャとして主に使うコマンドは dpkg だけです。

apt は自力で依存関係の問題を解決できますが、そのために現在インストールされているパッケージの削除や置換を提案することがあります。通常は提案に対して［y］と答えて構いませんが、念のため apt の提案内容を把握しておき、自分に必要なソフトウェアが使えなくなってしまう場合は［n］と答えてください。逆に、dpkg は依存関係の問題を自力で解決することはできず、エラーメッセージを出力するだけです。dpkg を使う場合は、このエラーメッセージを参考に手動で問題を解決しなければなりません。

それでは、具体的な apt と dpkg の使い方について説明していきましょう。

パッケージデータベースの更新（apt-get update）

Packages.gz などのパッケージデータベースをダウンロードし、/var/lib/apt/lists に保存します。Packages.gz は収録パッケージが更新される度に変わります。DVD-ROM からパッケージを入手する場合は、同じ DVD-ROM を使う限りデータベースも変化しないので update の必要はありません。通常はパッケージソースを ARMA Net にしておきオモイカネ社のアラート情報が発行されたタイミングで実行します。

```
# apt-get update
```

パッケージのインストール（apt-get install / dpkg -i）

パッケージをインストールします。apt-get install と dpkg -i は似た機能ですが、apt-get は「パッケージ名」を指定すれば apt が最新版を探して自動的にダウンロードしインストールまでするのに対し、dpkg では直接インストールする「deb ファイル名」を指定しなければなりません。dpkg の方が気が利かないようですが、敢えて最新版でないパッケージをインストールしたい場合などには、dpkg -i でしか対応できないなど重宝することもあります。なお、apt-get は最新版が既にインストールされていると処理を行いませんが、--reinstall をつけると強制的に再インストールできます。

```
# apt-get [--reinstall] install <パッケージ（複数可）>
# dpkg -i <*.deb（複数可）>
```

パッケージの削除（apt-get [--purge] remove / dpkg -r / dpkg -P）

インストールされているパッケージを削除します。--purge をつけると、remove だけでは削除されない設定ファイル類も含めて「パージ」（完全削除）します。ただし「削除」した段階で、apt のデータベースからもパッケージの情報が削除されるため、既に「削除」されているパッケージをさらに「パージ」する場合は dpkg -P しか使えません。

```
# apt-get [--purge] remove <パッケージ（複数可）>
# dpkg {-r|-P} <パッケージ（複数可）>
```

パッケージのアップグレード（apt-get upgrade / apt-get dist-upgrade）

インストールされているパッケージを最新版へアップグレードします。

```
# apt-get upgrade
# apt-get dist-upgrade
```

なお、アップグレードのために自分以外のパッケージの追加や置換が必要になる場合は、安全のため upgrade でのアップグレード対象から外されます。dist-upgrade ではこのような場合も構わずアップグレードをします。dist-upgrade でなく部分的に最新版にアップグレードする場合は apt-get install <パッケージ> として、明示的にパッケージを指定します。

```
# apt-get upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages have been kept back
  hoge
0 packages upgraded, 0 newly installed, 0 to remove and 1 not upgraded.

# apt-get install hoge
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  libhoge1
The following NEW packages will be installed:
  libhoge1
1 packages upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 1234kB of archives. After unpacking 5.6MB will be used.
Do you want to continue? [Y/n]
```

パッケージの一覧（dpkg -l）

条件に当てはまるパッケージの一覧を表示します。条件を指定しないとインストールされているパッケージの一覧が表示されます。

```
$ dpkg -l [< 条件 >]
```

パッケージの検索（apt-cache search）

パッケージデータベースからキーワードを含むパッケージを探して表示します。通常はパッケージの紹介文なども検索の対象になりますが、--names-only を付けると「パッケージ名にキーワードを含む」もののみが検索の対象になります。

```
$ apt-cache [--names-only] search < キーワード >
```

条件にはワイルドカードも使えますが、このときシェルにワイルドカードを展開されてしまわないよう「'」（シングルクウォート）や「"」（ダブルクウォート）で条件を囲んでおきましょう。

apt-get install / dpkg -i の場合と同じ状況です。

パッケージの情報を表示（apt-cache show / dpkg -I）

パッケージの依存情報・バージョン・サイズ・管理者・簡単な紹介などの付属情報を表示します。apt-cache ではパッケージデータベースから検索しますが、dpkg -I では deb ファイルから情報を直接読み込みます。そのため、apt-cache show ではパッケージ名、dpkg -I では deb ファイルを指定するという違いがあります。

apt-cache show では、複数のバージョンのパッケージが見つかるとそれぞれの情報を表示します。例えばネットワーク上にある最新版とインストールされているパッケージのバージョンが違う場合などがこれに当てはまります。

```
$ apt-cache show <パッケージ（複数可）>
$ dpkg -I <*.deb（複数可）>
```

パッケージとファイルの相互検索（dpkg -L / dpkg -S / dpkg -c）

パッケージに含まれるファイル名を調べたり、逆にファイル名を指定してそのファイルがどのパッケージに含まれていたかを検索したりすることができます。パッケージ名を指定して中に含まれるファイルを一覧表示するには dpkg -L を、逆にファイル名を指定してどのパッケージからインストールされたものかを調べるには dpkg -S を使います。また、特定の deb ファイルの中身を見るには dpkg -c を使います。

```
$ dpkg -L <パッケージ（複数可）>
$ dpkg -S <ファイル（複数可）>
$ dpkg -c <*.deb（複数可）>
```

キャッシュファイルの整理（apt-get autoclean / apt-get clean）

apt-get はダウンロードしたパッケージを /var/cache/apt/archives に保存しています。このキャッシュにより同じファイルを何度もダウンロードしなくて済むのですが、キャッシュは自動的には削除されないため、このディレクトリには徐々に古いパッケージが溜まってきてしまいます。そこで、autoclean で最新版だけを残して古いキャッシュを削除したり、clean で全キャッシュを削除したりしして整理をします。

```
# apt-get autoclean
# apt-get clean
```

パッケージのホールド

ソフトウェアの仕様が変わった直後など、あえてパッケージをアップグレードしたくないときは、パッケージをホールド（Hold）して apt-get upgrade の対象から外すことができます。ここでは「カーネル2.6.31.6はもう固定しておきたいな」というケースを想定して説明します。

この場合ホールドを設定するには、dpkg の機能を使って、root で以下のようにコマンドを実行します。

```
# echo 'kernel-2.6.31.6 hold' | dpkg --set-selections
```

正しくホールドされたかどうかは dpkg -l で確かめることができます。行頭が h になっていればホールドがうまくいっています。

```
# dpkg -l kernel-2.6.31.6
要望 =(U) 不明 /(I) インストール /(R) 削除 /(P) 完全削除 /(H) 維持
| 状態 =(N) 無 /(I) インストール済 /(C) 設定 /(U) 展開 /(F) 設定失敗 /(H)
|/ エラー =( 空欄 ) 無 /(H) 維持 /(R) 要再インストール /X= 両方
||/ 名前                バージョン        説明
+++-=================-================-==============================
hi  kernel-2.6.31.6     1               Linux kernel for ARMA/OGL
```

この状態では kernel-2.6.31.6 は自動的に更新されることはなくなります。ホールド解除は、echo コマンドの引数の hold の部分を install とします。

```
# echo 'kernel-2.6.31.6 install' | dpkg --set-selections
```

この他に apt-get install などで明示的にパッケージを再インストールしてもホールドは解除されます。

正常にインストールできていないパッケージを探す

dpkg -l では正常にインストールされているパッケージは、先頭２文字を ii または hi（ホールドの場合）で表します。よって、以下のようなコマンドラインで異常な状態のパッケージの一覧を得ることができます。

```
$ dpkg -l | grep -v "^[hi]i"
要望 =(U) 不明 /(I) インストール /(R) 削除 /(P) 完全削除 /(H) 維持
| 状態 =(N) 無 /(I) インストール済 /(C) 設定 /(U) 展開 /(F) 設定失敗 /(H)
|/ エラー =( 空欄 ) 無 /(H) 維持 /(R) 要再インストール /X= 両方
||/ 名前        バージョン    説明
+++-============-=============-====================================
iU  hoge1       1.2-3         Hoge No.1
rc  hoge2       2.3-4         Hoge No.2
iFX hoge3       3.4-5         Hoge No.3
```

ここではコマンドラインベースでの作業を紹介しますが、この他に dselect というインターフェースを使用して設定することもできます。

dpkg --get-selections でも確かめることができます。

hoge1 はパッケージが展開されただけで、設定が終わっていないパッケージです。依存関係の解決を含め、パッケージを再インストールするのが有効でしょう。

hoge2 は削除されたが完全削除（パージ）はされていず、設定ファイルが残っているパッケージです。設定ファイルを残したければそのままで構いませんが、dpkg -P でパージすることもできます。

hoge3 のように先頭から３文字目があるパッケージはエラーを起こしています。エラーへの対処は一口では説明できませんが、パッケージを一旦削除してから再インストールするのが多くの場合有効です。

不要なパッケージの一覧

deborphan は使われていない共有ライブラリなど、不要なパッケージを検索してくれるコマンドです。--guess-all などのオプションを付けると、検索対象とするパッケージの範囲を広げられます。詳しくは man deborphan を参照してください。

```
$ deborphan [< オプション >]
```

プロキシサーバの利用

環境変数 ftp_proxy, http_proxy にプロキシサーバのプロトコル・ホスト名・ポート番号を http://cosmos:8080/ のように設定しておくと、apt-get はプロキシ経由でファイルをダウンロードします。

可能な範囲で apt-get の作業を続ける

ミラーサイトからパッケージをダウンロードする場合、ミラーリングのタイミングによっては Packages.gz と実際にサーバ上に存在するパッケージが一致しないために、パッケージをダウンロードできないことがあります。通常は１つでもダウンロードできないパッケージがあるとインストールやアップグレードは行いませんが、apt-get -m とすると、ダウンロードできたパッケージで可能な作業を続けさせることができます。

```
# apt-get -m [< パラメータ >]
```

apt-get のエミュレーション

apt-get, dpkg では、実際にパッケージをインストールしたり削除したりせず、作業をする「ふり」をして、どのような作業を行うかを表示するだけのオプションがあります。多くのパッケージを一気に操作する前には、この機能を使って事前に何が行われるのかを調べておくと安心です。

```
# apt-get -s [< 各パラメータ >]
# dpkg --no-act [< 各パラメータ >]
```

3.2.8 パッケージのディレクトリ構成

パッケージプール

ARMA や Debian ではパッケージはパッケージプール（package pool）と呼ばれるディレクトリ形式に従って配置され、バージョンや対応機種に関係なく同じソフトウェアのパッケージを１つのディレクトリにまとめて管理しています。例として、架空のパッケージ hoge のパッケージが１つのディレクトリに収められている様子を表します。

2000 年 12 月からこのような形態になっています。

```
$ ls pool/main/h/hoge
libhoge1_1.2-3_i386.deb    hoge_1.2-3_i386.deb
libhoge1_1.2-3_arm.deb     hoge_1.2-3_arm.deb
libhoge0_0.9-8_i386.deb    hoge_0.9-8.diff.gz
libhoge0_0.9-8_arm.deb     hoge_0.9-8.dsc
libhoge-dev_1.2-3_all.deb  hoge_0.9-8_i386.deb
libhoge-dev_0.9-8_all.deb  hoge_0.9-8_arm.deb
hoge_1.2-3.diff.gz         hoge_1.2.3.orig.tar.gz
hoge_1.2-3.dsc             hoge_0.9.8.orig.tar.gz
```

ここでは x86 用と ARM 用のパッケージのみを挙げていますが、本来は他アーキテクチャ用のパッケージも全て同じディレクトリに入っています。また、hoge のソースから作られたパッケージならバージョンも関係なく stable 用の 0.9-8、testing / unstable 用の 1.2-3 の両方が入っていますし、バイナリだけでなくソースパッケージも同じディレクトリに配置されます。パッケージ hoge に関係する配布物はすべて同じディレクトリに置かれているということになります。

サイトレベルのディレクトリ

プールの上位にはサイトレベルのディレクトリがあります。この構造は通常パッケージシステム（apt）に隠蔽されていますが、参考情報として説明します。

ARMAのCD / DVD-ROMやARMA Netリポジトリでは以下のようなディレクトリツリーで配布されています。

```
/ -+- arma_3.0_updates
   +- arma_3.0_updates --- deb -+- Packages.gz
                               +- hogehoge_2.3-4o5_i386.deb
                               +- hogehoge_2.3-4.orig.tar.gz
                               +- hogehoge_2.3-4o5.dsc
                               +- hogehoge_2.3-4o5.diff.gz
                               |
```

また Debian のサイトでは以下のような構造化されたディレクトリツリーを持っています。下記で / は配布物の「ルート」を表します。例えば、Debian の公式サイトでは ftp://ftp.debian.or.jp/debian/ が以下の / に相当します。

```
/-+- dists -+- sid -+- main -----+- binary-all
  |         |       +- contrib   +- binary-armel
  |         |       `- non-free  +- binary-i386 ---- Packages.gz
  |         +- squeeze
  |         +- stable -> lenny
  |         +- testing -> squeeze
  |         `- unstable -> sid
  |
  `-pool--+--main -----+- a
          +--contrib   +- b
          `--non-free  +- c
                       |
                       +- h -+- hoge --+- libhoge1_1.2-3_i386.deb
                       |     |         +- libhoge1_1.2-3_arm.deb
                       |               +- libhoge0_0.9-8_i386.deb
                       |               |
                       |               `- hoge_0.9.8.orig.tar.gz
                       `- z
```

OGL1.2までは ogl ディレクトリと並んで、debian, debian-jp, debian-non-US ディレクトリがありました。ARMA2 以降で ogl ディレクトリに一本化されました。

第3章 システムの設定（book 時消去）

3.3 マウントとアンマウント

3.3.1 ファイルシステム

OS は静的な情報を「ファイル」という単位で管理します。ファイルは単に平面的に並べられるだけではなく、ディレクトリ・リンクなどによって構造的に配置されます。また、ファイルの属性としてタイムスタンプや所有者・パーミッションなどを実装していることもあります。このようなシステムをファイルシステムと呼びます。

Linux は、Linux 専用のファイルシステムだけでなく Windows, MacOS, BSD, 各社 UNIX などで使われているファイルシステムも扱えます。また、Windows の共有ディスクや NFS も「ネットワークファイルシステム」として扱えます。ARMA に収録している Linux カーネルが対応する主なファイルシステムは以下の通りです。

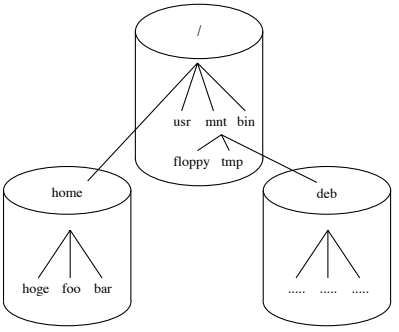
❗ 俗に言うディスクを「フォーマットする」とは「ファイルシステムを作る」ことである場合もよくあります。

正式名称	認識名	説明
XFS	`xfs`	ARMA 標準（SGI 開発）
Ext2	`ext2`	Linux 旧標準
Ext3	`ext3`	Ext2 + ジャーナリング
Ext4	`ext4`	Ext3 + より大きいファイルの扱い
ReiserFS	`reiserfs`	ジャーナリング FS の一つ
JFS	`jfs`	ジャーナリング FS の一つ
ISO-9660	`iso9660`	CD-ROM 用
UDF	`udf`	DVD-RAM, CD-RW のパケットライト用
FAT	`msdos,vfat`	Windows, MS-DOS 用
NTFS	`ntfs`	WindowsNT 用
NFS	`nfs`	UNIX 系 OS の共有ディスク
SMB	`smbfs,cifs`	Windows の共有ディスク

❗ 認識名はLinuxカーネルモジュールの名前で、後述の `/etc/fstab` など、様々な場面でファイルシステムの名前を示すときに使います。

3.3.2 マウント・アンマウント

UNIX 系 OS では、/（ルートディレクトリ）を頂点とする単一のディレクトリツリーでファイルを管理します。しかし、ファイルシステムはディスクやパーティションごとにありますから、単一のツリーを作るためにはそれらのファイルシステムを全て接続しなければなりません。この、ファイルシステムを接続する作業を「マウント」（mount）、逆に切り離す作業を「アンマウント」（un-mount）と言います。



3.3.3 fstab

マウントの設定ファイルは /etc/fstab です。Linux 起動時に自動的に行われるマウントや、後述の mount コマンドでの手動マウント時に参照されます。

```
proc            /proc          proc    defaults        0  0
none         /proc/bus/usb usbdevfs defaults        0  0
/dev/sda3       none           swap    sw              0  0
/dev/sda1       /              xfs     defaults        0  1
/dev/sda2       /home          xfs     defaults        0  1
/dev/sda5       /usr           xfs     defaults        0  1
/dev/sda6       /usr/local     xfs     defaults        0  0
/dev/cdrom      /mnt/cdrom     auto    ro,noauto,user  0  0
pc01:/dir01     /mnt/dir01     nfs     noauto          0  0
//pc02/dir02/ /mnt/dir02     smbfs   noauto          0  0
```

各行は次のような書式になっています。

< ディスク > < ポイント > < 種類 > < オプション > <num1> <num2>

第1パラメータにはマウントするディスクのデバイスファイルを指定します。ただし、ネットワークファイルシステムの場合はデバイスファイルではなく、9,10行目のようにNFSなら <サーバ>:<共有ディレクトリ>、SMBなら //<サーバ>/<共有ディレクトリ> という形式で指定します。

第2パラメータにはマウントポイントとするディレクトリを指定します。ここで注意すべきことは、Linuxの起動時に /etc/fstab が1行目から順に解釈されるため、上位のディレクトリをマウントポイントとするディスクほど前の行に書く必要があることです。例えば、3行目と4行目を入れ替えると、/dev/sda6 を /usr/local にマウントしようとしても、まだ /usr/local が存在しないのでマウントできません。

第3パラメータにはファイルシステムの種類を指定します。auto を指定するとファイルシステムを自動判別するので、様々な種類のメディアが挿入されるCD-RWドライブなどのリムーバブルディスクドライブでは便利です。ただし、特に「空ディスク」などで判別を間違える可能性もゼロではないので、ハードディスクなどファイルシステムの種類が確定している場合は xfs などと種類を明示した方が無難でしょう。

第4パラメータにはオプションを,(カンマ)で区切って指定します。オプションを指定しない場合は defaults とします。主なオプションには以下の通りです。

rw, ro	読み書きできる / 読み込みのみ
auto, noauto	起動時に自動でマウントする / しない
user, nouser	root 以外のマウントを許可 / 禁止
defaults	初期値(rw,auto,nouser)のままにする

user ではマウントしたユーザだけがアンマウントできますが、これを users とすると誰でもアンマウントできるようになります。

以上に挙げたのはファイルシステムに依存しない汎用のオプションですが、この他にファイルシステムごとに固有のオプションがあります。ファイルシステム固有のオプションについては man fstab を参照してください。

第5パラメータはバックアップコマンド dump を使用する場合に指定します。dump を使わない場合は0で構いません。

第6パラメータにはマウント時に fsck(FileSystem ChecKer)がファイルシステムを検証する順番を指定します。通常は / にマウントするものは1、その他の fsck が必要なものは全て2、fsck しなくてよいものは0を指定します。XFSのようにマウント時に自動的にファイルシステムの修復がおこなわれるタイプのファイルシステム(ジャーナリングファイルシステム)を使用している場合は0で問題ありません。

10,11行目では swap, proc という特殊なファイルシステムをマウントしています。swap は仮想メモリ(スワップ)領域として Linux カーネルが占有し何処にもマウントされないため、第2パラメータは none を指定します。proc は Linux カーネルが自分の状況を報告するための 仮想的なファイルシステムで、通常 /proc にマウントします。

ARMA ではパスの区切り文字に「/」を使います。Windows / DOS と異なり「¥」の方はエスケープ文字(特殊文字)としてしばしば使われます。

例えば / proc / interrupts は IRQ の状況を表します。

3.3.4 mount, umount

起動時以外に手動でマウントするには mount コマンドを使います。root であれば、以下の書式で任意のディスクを任意のマウントポイントにマウントできます。

```
# mount [-t <種類>] [-o <オプション>] <ディスク> <マウントポイント>
```

-t,-o, を省略すると、それぞれ -t auto と -o defaults と見なされます。その他の注意事項は /etc/fstab と同じです。

これに対し、一般ユーザーは fstab に user か users オプション付きで書かれた「特に許されたディスク」しかマウントできません。しかし、ディスクかマウントポイントのどちらか一方を指定すれば、fstab からマウントすべきディスクを特定できるので、root のときのように両方を指定する必要はありません。

```
$ mount {<ディスク> | <マウントポイント>}
```

この他に df コマンドを使ってもマウント状況をしることができます。

現在のマウント状況を調べるには、mount を引数なしで実行します。

```
$ mount
```

ディスクをアンマウントするには umount コマンドを使います。

```
$ umount {<ディスク> | <マウントポイント>}
```

一般ユーザが、特に許されたファイルシステムしかアンマウントできないことは mount と同じですが、その他に「内部のファイルがひとつでも開かれているとアンマウントできない」ことも要注意です。例えば、誰かがアンマウント対象となるファイルを開いていたり、カレントディレクトリにしていたりするとアンマウントできません。

このような場合具体的にどのプロセスが原因となっているかを調べるには lsof コマンドを使います。例えば /mnt/cdrom が開放できない、というような場合は以下のようにコマンドを実行します。

```
$ lsof | grep /mnt/cdrom
sh     6672  foo  cwd    DIR      3,2     6     517 /mnt/cdrom
```

さらに他のケースとしては、該当するファイルシステム内のどこかがマウントポイントとなっている場合や、ハードウェアレベルのエラーを起こしている場合等にアンマウントできない場合があります。

この場合、ユーザー foo の PID（プロセス ID）6672 のコマンド sh がアンマウントを妨害していることが分かりますので、kill を使って当該プロセスを排除すればアンマウントを行うことができるようになります。

3.3.5 マウントに関する諸注意

リムーバブルディスクのマウント

リムーバブルディスクを使う場合は、「メディア挿入→マウント」と「アンマウント→メディア取り出し」の順序を守るようにしてください。マウントされているメディアをアンマウントする前に取り出すと、メモリ上にあるキャッシュなどディスクへの最終的な書き込みが済んでいないデータが消失したり、ディスク上のファイルシステムが壊れたりすることがあります。DVD-ROM, CD-ROM 系や MO など、多くのリムーバブルディスクドライブでは、マウントされている間はイジェクトボタンがロックされてメディアを取り出せないようになるため、この種の事故は起こりにくくなっていますが、フラッシュメディアのアダプタや USB 接続のドライブのように特にロックされない場合もあります。

マウントポイントのディレクトリの下にあるコンテンツ

マウントポイントの下に既にディレクトリやファイルがあると、それらはマウント後には一時的に見えなくなってしまいますが、消去された訳ではありません。アンマウントするとまた何事もなく見えるようになります。しかし通常はマウントポイントにするディレクトリは空にしておいてください。

❗
カーネルがバッファしている内容を強制的に書き出させる場合は sync コマンドを使います。

3.4 デバイスファイル

3.4.1 デバイスファイル

UNIX 系 OS では静的な情報をファイル形式で取り扱いますが、そのような通常のファイルの他にデバイスファイルと呼ばれる特殊なファイルも存在します。これはデバイスドライバのインターフェースなのですが、「周辺機器を普通のファイルに見せかける」ための仮想的なオブジェクトです。言い換えると、デバイスファイルを普通のファイルと同様に読み書きすると、Linux が対応する周辺機器との入出力に変換してくれるという仕組みになっています。

なかなか説明では分かりづらいかもしれませんので、標準出力のリダイレクトを例にデバイスファイルの実例を見てみましょう。…まず、標準出力を普通のファイルにリダイレクトすると、出力内容がファイルに書き出されることは御存知の通りです。

```
# echo "hoge" > output
# cat output
hoge
```

ここで、普通のファイルの代わりにコンソールを指すデバイスファイルへリダイレクトすると、出力内容がそのコンソールへ「書き出され」ます。つまり、コンソールに文字が表示されます。

```
# echo "hoge" > /dev/tty6
```

/dev/tty6 は第 6 コンソールを指すデバイスファイルです。[Alt] + [F6] でコンソールを切り替えてみると、確かに hoge と出力されています。同様にして、プリンタにリダイレクトすれば文字が印刷され、シリアルポートにリダイレクトすればモデムを通して送信されます。

X から切り替える場合は [Shift] + [Alt] + [F6] です。

デバイスファイルは文字を扱う周辺機器に限ったものではありません。例えば音源も /dev/dsp というデバイスファイルで、読み込みは録音・書き込みは再生になります。

```
$ cat /dev/dsp > my_voice     ( 録音 )
$ cat my_voice > /dev/dsp     ( 再生 )
```

デバイスファイルは /dev ディレクトリ以下に、周辺機器ごとに慣習的に決められた名前で配置されています。一般的な周辺機器のデバイスファイル名は次の通りです。

仮想ターミナル= Pseudo tele TYpe

ALSA の音源デバイスファイルは /dev/snd 以下となります。

	慣用名
ターミナル（TeleTYpe）	tty[0-9]*
仮想ターミナル	pty*
ハードディスク	sd*
フロッピィディスク	fd*
シリアルポート	ttyS*
パラレルポート（Line Printer）	lp*
PS / 2マウス	psaux
USB マウス	input/mice
音源	audio, dsp* など
NULL デバイス	null

TTY とは、かつて主に大型コンピュータが活躍していた時代にコンピュータに接続していたテレタイプ端末（Teletype Terminal）に由来する名前です。現在ではディスプレイとキーボードの組がこれに相当し、TTY の読み取りはキーボードからの入力、書き込みはディスプレイへの出力になります。通常、パソコンに接続されているキーボードとディスプレイは１つだけですが、Linux では複数の TTY を［Alt］+［F1］～［F7］などで切り替えて使うことができるようになっています。これに対し、仮想ターミナルは TTY と同等の機能を提供する仮想的な周辺機器で、kterm、jfbterm、kon などのコンソール「エミュレータ」が利用します。なお、現在自分が使っているターミナルのデバイスファイル名は tty コマンドで確認できます。

```
$ tty
/dev/tty2
```

NULL デバイスはブラックホールの様な特殊なデバイスファイルで、読み込んでも内容はなく、書き込んでも何処にも出力されません。そのため、余計なメッセージの出力を /dev/null にリダイレクトして処分するために使うことがよくあります。例えば、次の例では標準出力を /dev/null にリダイレクトしていますので、標準エラー出力だけがターミナルに表示されます。

```
$ make > /dev/null
```

第７コンソールは通常 X が使います。また、X 上から他の TTY に切り替えるときは Ctrl + Alt + F1 ～ F7 を使います。

3.4.2 mknod

通常デバイスファイルはインストール時または udev によって動的に作成されますが、手動でデバイスファイルを作成することもできます。手動でデバイスファイルを作成する場合は mknod コマンドを使用します。

```
# mknod <ファイル> {b|c} <メジャー番号> <マイナー番号>
```

b はブロック型、c はキャラクタ型を表します。例えば、/dev/agpgart というキャラクタ型・メジャー番号 10・マイナー番号 175 のデバイスファイルを作るときは次のようにします。

```
# mknod /dev/agpgart c 10 175
```

ブロック型・キャラクタ型のタイプやメジャー・マイナーの番号は、周辺機器ごとに値が決まっています。また、たいていの場合はデバイスファイルの名前も慣用名として決まっています。これらの情報は Linux ソースの添付文書 Documentation/devices.txt や、デバイスファイルを使うソフトウェアの添付文書に書かれているので、そちらを参照してください。

3.5 ブートローダ

3.5.1 PC 互換機のハードディスクの構造

パーティション

Linux 環境を作るには、少なくとも１つのファイルシステムが必要です。実際には /home, /usr, /tmp, /var などを別のファイルシステムに分けて、マウントで繋いで運用することもよくあります。そこで、ハードディスクは中身を「パーティション」に分割して、パーティションごとにファイルシステムを作って使うようになっています。これによって、１台のハードディスクに複数のファイルシステムを使った Linux 環境を作ったり、複数の OS をインストールしたりできるようになっています。

そもそも現在の PC 互換機の元である IBM-PC / AT では、１台のハードディスクに４つまでパーティションを作れる仕様になっていました。しかし、当初は数十 MB だったハードディスクの容量が増大するに従ってこの規格も不十分になったため、元の４つの「基本」パーティションのうちの１つを「拡張」パーティションとして、その中を再分割して「論理」パーティションを作れるように拡張がなされました。論理パーティションには、一部の OS で起動パーティションとしにくいなどの制限もありますが、とにかく拡張パーティションの中にいくつでも作れるので、ハードディスクに作れるパーティションの数は無制限となりました。

Linux ではこのハードディスクやパーティションもデバイスファイルとして表します。ハードディスクはプライマリマスタから順に /dev/sda, /dev/sdb … となります。パーティションはハードディスクのデバイスファイルに、基本・拡張パーティションには１～４、論理パーティションには５から始まる番号を付け、/dev/sda1, /dev/sdb5 のように表します。

セクタ

ディスクは読み書きしたデータを、「セクタ」（Sector）という単位で入出力しています。PC 互換機のハードディスクでは、１セクタ は512Bytes です。このセクタのうち、特にハードディスクの先頭１セクタは MBR（Master Boot Record）と呼ばれ、基本・拡張パーティションのディスク上での位置を記録したパーティションテーブル（16Bytes ×４）と、ブートローダの一部（最大 446Bytes）、それに識別符号（2Bytes）が格納されています。

論理パーティションのパーティションテーブルは拡張パーティションの中にあります。

3.5.2 ブートローダ

ブートローダ（Boot Loader）は、一言で言えば「OS を起動するプログラム」です。定義には諸説ありますが、一般的には OS カーネルの起動前に稼働するプログラムのうち、BIOS が OS の手前で起動するソフトウェアをブートローダと呼んでいます。ここでは、PC 互換機の起動手順に沿ってブートローダの役目を説明します。

PC 互換機では、電源が入るとまず CPU が特定のメモリアドレスにある BIOS を実行します。BIOS はマシンの初期化などの処理を行い、最後に BIOS 設定で指定されている順にディスクの特定位置（多くの場合は先頭）を読み込み、そこにあるプログラムを実行します。ハードディスクの場合は MBR の中のブートローダが実行されるのですが、446 Bytes 以下というこの1段目のブートローダに複雑な処理はできないので、多くの場合1段目はディスク上の別の場所にある2段目を読み込んで実行するだけとなっています。実質的なブートローダの処理をするのはこの2段目で、OS カーネルを読み込んで実行するまでの処理をこなします。こうして無事OSが起動できるとブートローダの役目は終わりです。

実際のブートローダにはもっと多段式のものもありますが、とにかく「自分自身の処理→次の段を読み込んで実行」を繰り返して OS を起動する点は同じです。この流れをブートストラップ（Boot Strap）と言います。

単純なブートローダは特定のOSしか起動できませんが、複数のOSに対応し、起動する OS を選択できる「ブートセレクト」機能を持つブートローダもあります。また、そのようなブートローダの中にはブートストラップの仕組みを利用して、自分では OS を起動せず、選択した OS の専用ブートローダに継投するだけの「チェインローダ」というものもあります。Linuxで使われるブートローダは Windows などに配慮してブートセレクト機能を搭載するものが殆どですが、Linux 以外の OS はチェインロードの場合が多いようです。

3.5.3 GRUB

ARMA ではブートローダとして GRUB（GNU GRUB）を採用しています。GRUB には次のような特長があります。

（1） XFS, ReiserFS, Ext2, FATなどのファイルシステムを直接認識できる。

（2） 8.4GB（1024 シリンダ）以上の領域からも OS を起動できる。

（3） OS を矢印キーを使って選択できる「GRUB メニュー」。

（4） 柔軟に OS を起動できる「GRUB シェル」。

この他にも、BIOS トラップ、ネットワークブートなど、様々な局面で柔軟に OS を起動できる機能を意欲的に収録しています。

中でもファイルシステムを直接認識できるのが GRUB の最大特徴と言えます。詳しくは後述しますが、GRUB はファイルシステムから `/boot/grub/menu.lst` という設定「ファイル」を読み込み、それに従って、例えば `/boot/vmlinuz-2.6.31.6` というカーネルイメージ「ファイル」を読みこんで起動できるのです。よって、GRUB ではカーネルイメージのファイル名が変わっても `menu.lst` を書き換えるだけで OK です。

GRUB はファイルシステムを扱うことはできますが、ドライブは BIOS を経由して認識しますので、ドライブの表記は Linux の流儀とは異なります。「BIOSがディスクを認識する順」に番号を割り振った `(hd0)`, `(hd1)`,… や `(fd0)`, `(fd1)`,… を使います。これはあくまで BIOS が認識する順番ですから IDE と SCSI の区別はなく、BIOS の設定で IDE が先なら IDE から、SCSI が先なら SCSI から順に番号が振られます。

例えば、次のような IDE のハードディスクが２つ、SCSI のハードディスクか２つ、IDE の CD ドライブが一つの環境では、以下のように GRUB でのドライブ名が与えられます。

		デバイスファイル	GRUB での名前
IDE プライマリマスタ	HDD	/dev/sda	(hd0)
IDE プライマリスレーブ	CD	/dev/sr0	—
IDE セカンダリマスタ	HDD	/dev/sdb	(hd1)
SCSI ID=0	HDD	/dev/sdc	(hd2)
SCSI ID=2	HDD	/dev/sdd	(hd3)

さて、ここで BIOS の設定を変えて SCSI を先に認識するようにすると、GRUB でのディスク名と機器との対応が変わります。よって、IDE と SCSI を併用する場合は、事前に BIOS がディスクを認識する順番を確認しするようにしてください。

		デバイスファイル	GRUB での名前
SCSI ID=0	HDD	/dev/sda	(hd0)

CD-ROM など、BIOS がハードディスクと認識しない機器は GRUB では無視されます。また この他に USB メモリをドライブとしてカウントする場合など、BIOS によって変わる場合があります。

SCSI ID=2	HDD /dev/sdb	(hd1)
IDE プライマリマスタ	HDD /dev/sdc	(hd2)
IDE プライマリスレーブ	CD /dev/sr0	—
IDE セカンダリマスタ	HDD /dev/sdd	(hd3)

GRUB でのパーティション番号は、Linux のデバイスファイルと基本的に同じです。ただし、GRUB では 0 から、Linux では 1 から数え始めるので、GRUB でのパーティション番号は「Linux での番号 -1」になります。例えば、/dev/sda = (hd0) である環境では、/dev/sda1 = (hd0,0)、/dev/sda5 = (hd0,4) となります。

さらに、GRUB でファイルを指定するときは、GRUB 形式のパーティション名に続けて、その「パーティション内のファイルシステム」でのパスを付けます。例を示すと、/dev/sda = (hd0) の環境で、/dev/sda3 を /boot にマウントしているならば、Linux 上で /boot/grub/menu.lst というファイルは、GRUB では (hd0,2)/grub/menu.lst となります。

3.5.4 GRUB の動作

GRUB の動作は GRUB を MBR にインストールした場合とパーティションにインストールした場合で異なります。ここでは ARMA 標準の動作である MBR にインストールした場合について、PC 互換機の電源投入後の動きを追って説明します。

マシンの電源投入後、BIOS は MBR に書き込まれた GRUB stage1 を読み込みます。446Bytes 以下の stage1 ではファイルシステムを認識する余裕がないので、stage1 はセクタを直接指定して stage 1.5 を読み込みます。この stage 1.5 は MBR 直後にある「パーティションに割り当てられないセクタ」に書き込まれているので、そこを使う特殊なプログラム（多くの場合は他のブートローダ）を使わない限り書き換えられる心配はありません。

stage1.5 は、XFS, Ext2, ReiserFS, JFS, FAT などのファイルシステムのうち「1 つだけ」を認識でき、stage2 を読み込みます。stage2 は GRUB メニューや GRUB シェルを含む GRUB の本体です。stage2 は先程挙げたファイルシステムを「全て」認識でき、設定ファイルを読み込んで解釈して GRUB メニューを表示します。後は GRUB メニューや GRUB シェルを通じてユーザの指示した通りに OS を起動するわけです。

インストーラはあらかじめ、GRUB のインストール時に stage2 の所在とそこのファイルシステムの種類を調べ、それを認識できる stage1.5 を選んでインストールしています。

3.5.5 GRUB の設定

GRUB の設定ファイルは /boot/grub/menu.lst です。

```
timeout 15

title ARMA 3.0 (2.6.31.6)
root (hd0,1)
kernel /boot/vmlinuz-2.6.31.6 root=/dev/sda2 vga=0x301

title Windows
rootnoverify (hd0,0)
chainloader +1
```

timeout 行では GRUB メニューでユーザの選択を待つ時間を秒単位で設定します。時間切れになると、一番最初に書かれている OS を自動的に起動します。

title 行には GRUB メニューに表示させる OS の識別名を指定します。個々の OS の起動設定は、この title 行から次の title 行までの間に書きます。

root 行では GRUB の「ルートデバイス」、即ち GRUB が作業対象とするパーティションを指定します。通常は起動したい OS があるパーティションを GRUB 形式で指定すればよいでしょう。

kernel 行では起動させる OS のカーネルイメージファイルを GRUB 形式のパスで指定し、同時にカーネルオプションを与えます。このとき、イメージファイルがルートデバイスにあれば、パスのうちパーティションを表す部分、上の例で言えば (hd0,0)/boot/vmlinuz-2.6.31.6 と書くべきところの (hd0,0) を省略できます。さらに、Linux を起動する場合は、最低限のカーネルオプションとして root= に続けて / にマウントするパーティションを指定しておきます。

rootnoverify 行は root と似ていますが、GRUB にルートデバイスのファイルシステムを認識させない点が違います。

GRUB が認識できないファイルシステムを使う OS をチェインロードする場合などに使います。

chainloader 行にはチェインロードするブートローダのセクタを X+Y の形式で、X にブートローダの先頭セクタの番号（0 なら省略可）、Y には読み込むセクタ数を指定します。つまり、上の例の +1 では先頭 1 セクタを読み込むことを表しています。

逆に言えば、イメージファイルはルートデバイスになくても構いません。例えば、ハードディスクから起動するのにカーネルのイメージファイルは FD から読み込ませることも可能です。

3.5.6 GRUB メニューと GRUB シェル

実際の画面とは少々異なります。

設定ファイルを変更したらマシンを再起動します。設定が正しければ以下のような GRUB メニューが表示されます。GRUB メニューでは［↑］［↓］で起動する OS を選択し、［Enter］で起動できます。一時的に設定を変更して起動するには、OS を選択した後［e］で編集画面に入ると行単位での編集ができます。

```
   GRUB  version 0.97  (639K lower / 261056K upper memory)

+-------------------------------------------------------------------+
| ARMA 3.0 (2.6.31.6)                                               |
| Windows                                                           |
|                                                                   |
|                                                                   |
|                                                                   |
|                                                                   |
+-------------------------------------------------------------------+
  Use the ↑ and ↓ keys to select which entry is highlighed.
  Press enter to boot the selected OS, 'e' to edit the
  commands before booting, or 'c' for a command-line.

 The highlighted entry will be booted automatically in 15 seconds.
```

GRUB シェル上ではタブ補完や Emacs 風の行編集も使えるため、ブートローダ上のプログラムとは思えないほど操作感は良好です。

何らかのエラーが発生した場合や、GRUB メニューで［c］を押した場合は GRUB シェルに入ります。GRUB シェルは `/boot/grub/menu.lst` に書いたような GRUB の内部コマンドを実行できる「OS を手動で起動する」ための専用シェルです。ARMA3.0 では GRUB のキーボードは日本語に固定されています。英語キーボードをお使いの場合は以下の表を参考に（特に記号部分を）読み替えてください。

日本語 106 キーボード（部分）

英語 101 キーボード（部分）

この GRUB シェルのおかげで、ブートローダのトラブル時にとれる対策の幅が広がります。例えば、カーネルイメージの入った GRUB 起動フロッピーディスクを用意しておけば、殆どどんな場合でもとにかく OS を起動することだけはできます。

それでは、GRUB シェルの利用例として Linux と Windows の起動例を示しておきましょう。基本的には `/boot/grub/menu.lst` に書いたことをそのまま 1 行ずつ入力し、最後に `boot` で起動するだけです。

```
grub> find /boot/vmlinuz-2.6.31.6
 (hd0,0)

grub> root (hd0,1)
 Filesystem type is xfs, partition type 0x83

grub> kernel /boot/vmlinuz-2.6.31.6 root=/dev/hda2 vga=0x301
   [Linux-bzImage, setup=0x1400, size=0x10f456]
```

```
grub> boot
```

```
grub> rootnoverify (hd0,0)
 Filesystem type is fat, partition type 0xe

grub> chainloader +1

grub> boot
```

3.5.7 GRUB の修復（再インストール）

GRUB をアップグレードする場合、他のブートローダなどによって GRUB stage1 や stage1.5 が上書きされてしまい GRUB が起動できなくなった場合、stage2 の GRUB 形式でのパスが変わった場合、もしくは stage2 が入っているファイルシステムの種類が変わった場合には GRUB の再インストールが必要です。再インストールには様々な方法がありますが、ここでは最も確実な、GRUB をインストールしたフロッピーディスク（以下 GRUB FD）を使う方法を紹介します。

まず、GRUB FD を持っていない場合は次の手順で作ります。

```
# cd /boot/grub
# dd if=stage1 of=/dev/fd0 bs=512 count=1
# dd if=stage2 of=/dev/fd0 bs=512 seek=1
```

GRUB をアップグレードする場合は、grub パッケージは安全のため直接 /boot/grub のファイルを書き換えず /usr/lib/grub/i386-pc にファイルを置く仕様になっているため、/usr/lib/grub/i386-pc の GRUB 関連ファイルを全て /boot/grub にコピーしておきます。

さて、準備ができたら GRUB FD でマシンを再起動し、GRUB シェルに入ります。そして、Linux でいう /boot/grub があるパーティションをルートデバイスにしてから setup< ディスク > で GRUB をインストールします。

```
grub> root (hd0,1)
 Filesystem type is xfs, partition type 0x83

grub> setup (hd0)
 Checking if "/boot/grub/stage1" exists... yes
 Checking if "/boot/grub/stage2" exists... yes
 Checking if "/boot/grub/xfs_stage1_5" exists... yes
 Running "embed /boot/grub/xfs_stage1_5 (hd0)"... 20 sectors are
embedded.
succeeded
```

```
 Running "install /boot/grub/stage1 d (hd0) (hd0)1+20 p (hd0,1)/boot/
grub/
stage2 /boot/grub/menu.lst"... succeeded
Done.
```

以上でインストールは完了です。

3.6 TCP / IP ネットワーク

3.6.1 プロトコルの階層構造

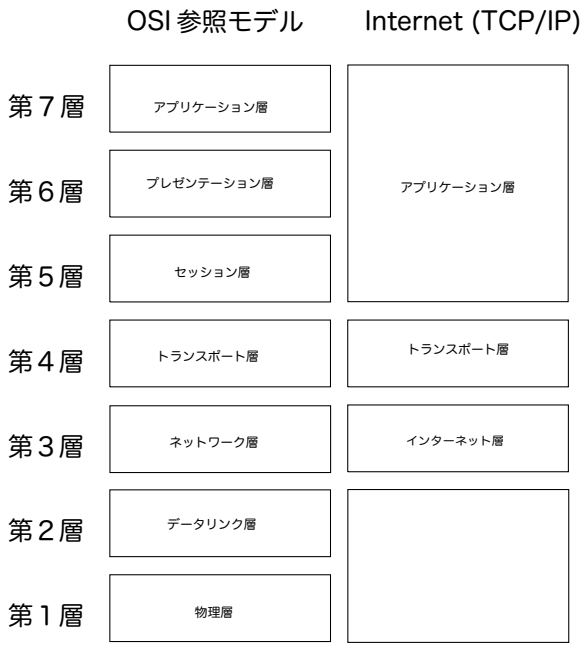
コンピュータネットワークはネットワークに参加するハードウェア・ソフトウェアが共通の仕様に則って通信することによって成立しています。この共通の通信仕様をネットワークプロトコル（Network Protocol）、あるいは単にプロトコルと呼びます。

プロトコルは、ネットワークアプリケーションの実装からハードウェアの電気的な設計に至るまで様々な分野に渡って定義されていますが、これでは余りにも広大で全体を一目で見渡すことができないので、プロトコルは「通信内容をどう捉えるか」という観点で分類して考えるようになっています。例えば、アプリケーションは通信内容を「データ」として扱いますが、同じものを OS は「パケット」として、さらにハードウェアは「電気信号」として扱っています。こうして分類されたプロトコルは、ユーザに近い方、即ちアプリケーション側を上位、逆にハードウェア側を下位とする階層構造に並べることができます。

このプロトコルの階層構造（プロトコルスタック）には、「OSI 参照モデル」に定義された7階層が標準ですが、インターネットではこれをアプリケーション・トランスポート・インターネットとそれより下位の層にまとめなおしています。具体的に言うと、アプリケーションが発する「データ」は、トランスポート層で「パケット」に分解され、ネットワーク層で配信ルートを決定され、ネットワークインタフェース層で「電気信号」として送信されると言った具合です。受信側でこれと逆の操作を行って電気信号をデータに復元して無事アプリケーションに届けられれば、めでたく通信が確立します。

ところで、ネットワークアプリケーションは、発信したデータが相手に届きさえすればよく、途中でどんなパケットに分割されどんな電気信号になろうとアプリケーション層には無関係です。このようにプロトコルは層ごとに独立しているので、各階層のプロトコルは入れ替えることもできます。

❗ 階層ごとに複数のプロトコルが存在することができます。



それでは、実際に各層の概要を紹介していきましょう。

アプリケーション層

ポートは正確に言えばアプリケーション層内部ではなく、トランスポート層との境界にあるデータ受け渡しの窓口です。

アプリケーション同士がデータをやりとりする層で、OSI 参照モデルでは第 5,6,7 階層に相当します。ここではネットワークアプリケーションごとに Web ブラウザは HTTP, ファイル転送は FTP, 電子メールは SMTP / POP…などとプロトコルが定義されています。この多数のプロトコルを整理するために 0 ～ 65535 番の「ポート」（Port）があり、各プロトコルのサーバは別々の決まった番号のポートで待ち受けをするようにしています。主なサーバとポート番号の割り当て表は `/etc/services` にありますが、ここでは特に有名なポート番号を例として挙げておきましょう。

ftp	20,21
ssh	22
telnet	23
smtp	25
http	80
pop3	110

逆に、クライアントは決まった番号ではなく OS に割り当てられた適当な空きポートを使い、サーバに接続するときにそのポート番号を知らせます。するとサーバはそのポートにたいして応答しますので、サーバとクライアントの間で間違いのない一対一の通信が可能になっています。

トランスポート層

通信を「制御する」層で、OSI 参照モデルでは第４階層に相当します。

アプリケーションから発信され、ポートからトランスポート層に入ってきたデータは、ここでパケットに分割されます。

トランスポート層のプロトコル一覧は `/etc/protocols` にありますが、この中で最も有名なものは TCP（Transmission Control Protocol） です。TCP はパケットの内容が通信中に化けていないか、パケットが順番通り漏れなく届いたかなど、正常に通信できたかどうかを確認できるプロトコルです。従って TCP で行われた通信は確実ですが、確認やエラーパケットの再送信などの手間のために通信速度が遅く、一定の速度を保ちづらいという欠点もあります。

そのため、動画配信など、若干データが狂うリスクを覚悟で高速・等速度通信を求める場合は TCP に代わって UDP（User Datagram Protocol）を使います。UDP はインターネット層とアプリケーション層をそのままつなぐプロトコルで、エラーチェックなどは行いません。

インターネット層（1）- IP アドレスとネットワーク部・ホスト部

インターネット上のホストからホストへパケットを届けるルートを決める層で、OSI 参照モデルでは第３階層に相当します。インターネット層のプロトコルは、その名の通りの IP（Internet Protocol）です。

IP では、データを送受信する機器のそれぞれに IP アドレスという相異なる番号をつけて機器を区別しています。現在の IP（IPv4）では IP アドレスは 32bit で、分かりやすく記述するために 8bit ずつ４つに "." で区切り、各 8bit を０～255 の 10 進数で表記します。例えば 11000000101010000000000100000001 は 192.168.1.1 です。

IP アドレスの 32bit はネットワーク部とホスト部の２つに分けられます。ネットワーク部は「ネットワークのネットワーク」たるインターネットを構成する個々の「ネットワーク」を指し、ホスト部でその中の個々のホストを表します。

例えば、192.168.1.1 の先頭 24bit がネットワーク部だとすると、ネットワーク自体を表す IP アドレス = ネットワークアドレスは 192.168.1.1 の先頭 24bit を切り出して残りを０で埋めた 192.168.1.0 となります。また、ネットワーク内の最大の IP アドレスである 192.168.1.255 は「ブロードキャスト」というネットワークの全ホストにデータを送るための IP アドレスとして使います。そして残りの 192.168.1.1 ～ 254 をホストに割り当てるのが慣習です。

これをまとめてネットワークを192.168.1.0 / 255.255.255.0、またはもっと簡単に192.168.1.0 / 24のように表すことができます。

この「ネットワーク部は何 bit か?」を表すのが「サブネットマスク」です。サブネットマスクは、IP アドレスとの and がネットワークアドレスとなるように決められます。例えばネットワーク部が 24bit ならサブネットマスクは先頭 24bit に 1 を立てた 255.255.255.0 となり、ネットワークアドレスは以下のように計算できます。

```
    192.168.  1.  1 = 11000000 10101000 00000001 00000000
and  255.255.255.  0 = 11111111 11111111 11111111 00000000
------------------------------------------------------------
    192.168.  1.  0 = 11000000 10101000 00000001 00000000
```

実は、サブネットマスクは IPv4 に後付けで考えられた仕組みです。初期の IPv4 では IP アドレスをクラス A ～ C に分類し、クラスごとにネットワーク部のビット数を事前に決めてありました。クラス A は 0.0.0.0 ～ 126.255.255.255 でネットワーク部 8bit、クラス B は 128.0.0.0 ～ 191.255.255.255 でネットワーク部 16bit、クラス C は 192.0.0.0 ～ 223.255.255.255 でネットワーク部 24bit です。(この他にクラス D,E (224.0.0.0 ～) もありますが、一般には使わないのでここでは説明しません。) しかし、この「クラス」の意味はサブネットマスク導入によって薄れ、現在ではどのクラスの IP も同様にホスト部を 3,4bit にまで細かく区切って配分されています。ところで、127.0.0.0/8 が空いていますが、これはループバックという「自分自身を表す」IP アドレスとして使われています。通常 127.0.0.1 をループバックアドレスとし、ここに送ったパケットは自分に届くようになっています。

インターネット層 (2) - ルーティング

それでは、実際にパケットを送る処理の説明に移りましょう。まず、トランスポート層から来た TCP / UDP のパケットに、インターネット層では送信・受信ホストの IP アドレスなどの情報を含んだ IP ヘッダを付けて IP パケットを作ります。

さて、このパケットを相手に届けようと思っても、インターネットを横断して世界中に直接パケットを送れるわけではありません。通常、パケットを直接送れるのは同じネットワーク内のホストだけです。しかし、インターネットには複数のネットワークが乗り入れる接点に「ルータ」というネットワークの乗り換えポイントが用意されています。このルータには「ルーティングテーブル」という、パケットの宛先別に次にパケットを送るルータを書いた表が用意されています。パケットはこの表に従ってルータからルータへとネットワークを横断していけば、最終的にいつか目的地に届くという訳です。また、ルータでない一般のホストにもルーティングテーブルはありますが、これには「直接送れないホスト宛のパケットはネットワーク内のルータに送る」とだけ

書くのが一般的です。

例えば、192.168.0.0 / 24 と 192.168.1.0 / 24 という 2 つのネットワークに、192.168.0.2・192.168.1.2 というホストと、両ネットワークをつなぐルータの 192.168.0.1 + 192.168.1.1 がある環境を考えます。このとき、192.168.0.2 → 192.168.1.2 は直接パケットを送れないので、192.168.0.2 は自分のルーティングテーブルを参照してルータにパケットを送ります。ここでルータのルーティングテーブルに、192.168.1.0 / 24 宛のパケットは自分の 192.168.1.1 から送信するよう書いてあれば、パケットが 192.168.1.2 に届くというわけです。

インターネットでは、このルータが多ければ数十個もつながってパケットをルーティングしています。先ほど挙げたものは極めて簡単な例ですが、実際の現場のネットワークの基幹部にあるルータも同じ原理で動いています。

インターネット層（3）- グローバルアドレスとローカルアドレス

IP アドレスには、これとは別に「グローバルアドレス」と「ローカルアドレス」という区分もあります。

グローバルアドレスはインターネットに接続するホストに割り当てられる IP アドレスで、世界中に二つと同じグローバルアドレスを持つホストがないよう、ICANN を総元締めに管理されています。このため、私たちが勝手なグローバルアドレスを名乗ることは許されず、プロバイダによって割り当てられたグローバルアドレスでしかインターネットに接続できません。

これに対し、ローカルアドレスは個々のネットワーク内でのみ有効な IP アドレスで、ネットワーク内の裁量で自由にホストに割り当てられる代わりに、ネットワークの外のホストとは交信できないと定められています。ローカルアドレスに割り当てられている IP アドレスは以下の範囲です。

```
 10.  0.  0.  0〜 10.255.255.255（クラス A が  1 個）
172. 16.  0.  0〜172. 31.255.255（クラス B が 16 個）
192.168.  0.  0〜192.168.255.255（クラス C が 256 個）
```

ただし、ローカルアドレスしか持っていないホストも、NAT（Network Address Translation）を経由してローカルアドレスとグローバルアドレスを変換すればネットワーク外と交信できます。NAT はローカルとグローバルの両方のアドレスを持つホストに装備し、自分を通る「ローカル発グローバル行」の IP パケットのヘッダを、自分の空きポートを使って「グローバル発」に書き換え、さらに相手からの返信パケットのヘッダを今度は逆に書き換えるという作業をします。

ローカルアドレスのことを「プライベートアドレス」とも呼びます。

ICANN = Internet Corporation for Assigned Names and Numbers

X,Y は WWW ブラウザや NAT に OS が割り当てた空きポート番号を表しています。

NAT の例を示すために、先ほどルーティングの説明で使ったネットワークの例をもう一度使いましょう。今回は cosmos がグローバルアドレスも持った NAT 装備機でインターネットにつながっているとします。すると 192.168.0.2 発の「192.168.0.2:X → www.omoikane.co.jp:80」というパケットが cosmos を通るとき、NAT は IP ヘッダを「cosmos:Y → www.omoikane.co.jp:80」に書き換えます。そして、この返信の「www.omoikane.co.jp:80 → cosmos:Y」というパケットも「www.omoikane.co.jp:80 → 192.168.0.2:X」に書き換えて 192.168.0.2 に送ります。こうすると、確かに 192.168.0.2 は www.omoikane.co.jp:80 と交信できたことになりますが、インターネットにはローカルアドレスを含むパケットは流れていないという訳です。

それより下位の層

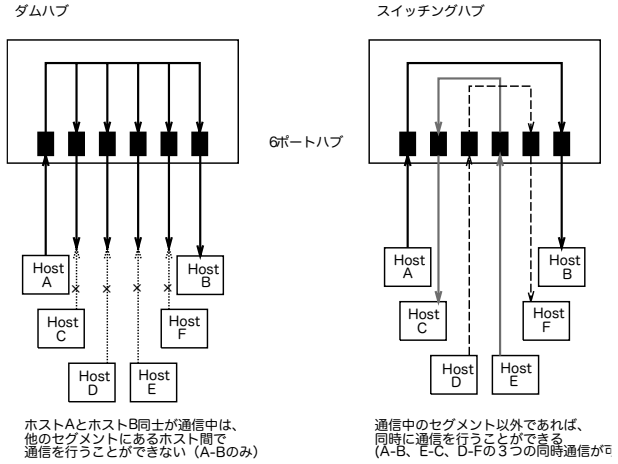
実際にデータを電気信号や光信号として送るハードウェアに密着した層で、OSI 参照モデルでは第 1,2 層に相当します。この層で主に使われるプロトコルは Ethernet と PPP（Point to Point Protocol）です。

前半 24bit はメーカに割り当てられ、後半 24bit を各メーカで製品ひとつひとつに割り当てています。

Ethernet は主に LAN（Local Area Network）に使われており、その 30 年以上の歴史の中で様々な規格がありましたが、現在普及しているのは 10-BASE-T, 100-BASE-TX, 1000BASE-T の 3 つです。これらの規格は、いずれも UTP ケーブルの両端に RJ-45 端子を付けハブに接続して配線するという点は同じですが、通信速度はその名の通り 10Mbps, 100Mbps, 1000Mbps と違います。そして、Ethernet ではひとつひとつの機器に MAC（Media Access Control）アドレスという 48bit の相異なる番号を付けて各機器を区別しています。

さて、Ethernet では IP パケットに、さらに宛先 MAC アドレスなどの情報を含んだ Ethernet ヘッダを付け、電気信号として Ethernet パケットを発信します。電気信号ですので電線で繋がった全機器に Ethernet パケットが届いてしまいますが、受け取った機器ではそれぞれ自分の MAC アドレス宛でない Ethernet パケットを捨てます。こうすると同じ MAC アドレスを持つ機器はひとつしかないので、最終的にパケットを受け入れる機器はひとつだけとなり、きちんと通信ができたことになります。

しかし、この原理上 Ethernet ではネットワークに繋がる機器が増えるほどパケットが衝突する（コリジョンが発生する）確率が高くなります。コリジョンが発生すると、その時点で Ethernet 上にあるパケットは全て無効となり後で再送信されるので、通信速度が遅くなってしまいます。ただし、ハブにスイッチングハブを使うと、Ethernet パケットを理解して不必要なパケットを流さないようにするため、コリジョンを減らすことができます。



PPP は、主に電話回線を通してインターネットに接続するときに使われるプロトコルです。PPP のパケットはモデムやターミナルアダプタ（TA）から電話回線を通ってプロバイダ側のモデムや TA に届き、そこから Ethernet など別のプロトコルに乗り換えてインターネットにつながっていきます。

3.6.2 IPv4アドレスの枯渇と IPv6

現在主流の IP = IPv4 では IP アドレスは 32bit です。従って、単純計算では全部で 43 億弱の IP アドレスが存在することになりますが、ルーティング効率の問題もあり、クラス D,E・ネットワークアドレス・ブロードキャストアドレスなど、43 億の中にはホストに割り当てられないアドレスが多く含まれています。さらに割り当ての都合上使われない IP アドレスというものが相当量あります。例えば 20 台のホストがあるネットワークではホスト部が最低でも 5bit 必要ですが、すると 10 個ものアドレスが無駄になってしまいます。こうして考えると、インターネットに存在できるホストは 43 億よりもかなり少なくなり、ごく近い将来インターネットの IP アドレスは枯渇する（割り当てがおこなわれない）と考えられています。

IPv4 に代わる方式として IPv6（IP version6）を使い、IP アドレスを増やすことで根本的に問題を解決する試みが始まっています。IPv6 では IP アドレスは 128bit（10 進法で 39 桁）という膨大な数になるので枯渇の心配は基本的になくなります。現在、IPv6 は Linux にも実装され、いくつかのプロバイダが参加して利用が広まりつつある段階です。近い将来には IPv4 からの移行が進むものと思われます。

3.6.3 ホスト名と名前解決

Domain : 和訳は色々考えられますが、この場合は「領域」がよいでしょうか？

インストーラでも尋ねられますが、ARMA（UNIX 系 OS）ではコンピュータにマシンに「ホスト名」という名前を付けます。また、ネットワークにもドメインという名前を付けます。

例えば www.omoikane.co.jp ではマシン名が www で、ドメイン名が omoikane.co.jp です。見て分かるように、ドメイン名は .（ドット）区切りの階層構造になっており、後ろの区切りほど上位の階層で広い領域を表します。また、各区切りを、後ろからトップレベルドメイン（TLD = Top Level Domain）、セカンドレベルドメイン…と呼びます。この例では jp は日本を、以下 co は営利組織を、omoikane はオモイカネを表します。

この「ホスト名＋ドメイン名」で表すマシンのフルネーム（FQDN = Fully Qualified Domain Name）は IP アドレスに対応し、IP アドレスの代わりに使うことができます。例えば、www.omoikane.co.jp は 202.216.241.67 に対応しています。さらに、自分と同じドメインのマシンはドメイン名を省略してマシン名だけでアクセスできます。例えば、omoikane.co.jp 内のマシンなら www だけで www.omoikane.co.jp にアクセスできます。

従って、FQDN も IP アドレスのようにインターネットで重複がないように ICANN によって管理されています。そのため、ドメイン名も IP アドレスと同様に ICANN の承認が必要なのですが、IP アドレスとは違い、好きな名前を申請して既に登録された名前でなければ受理されるようになっています。ちなみに、ホスト名についてはネットワーク内の裁量で、ネットワーク内で重複がなければ勝手に付けて構いません。

しかし、こうして FQDN とホストの対応関係を作れても、IP は IP アドレスしか理解できないのでマシン名ではパケットを送信できません。そこで、何らかの方法で FQDN を IP アドレスに変換して、IP にはあくまで IP アドレスで指示を与えられるようにする仕組みが必要になります。この変換作業を「名前解決」、特に FQDN → IP アドレスを「正引き」、IP アドレス → FQDN を「逆引き」と言います。

最も簡単に名前解決をするには、/etc/hosts を使います。

```
127.0.0.1       localhost
192.168.0.1     cosmos
192.168.0.2     kakitsubata
192.168.0.3     syakuyaku
```

極めて単純ですが、このファイルを参照すれば正・逆引きどちらにも使えます。小規模な LAN では、この /etc/hosts を全てのホストに書いて名前解決をする方法でも実用に耐え、かつ一番面倒が少ないでしょう。しかし、インターネット全体のホストの名前解決の情報は膨大かつ刻々と変化するので、これでは対処しきれません。

そこで、各ドメインにFQDNとIPアドレスを相互変換するDNS（Domain Name System）サーバを立てる方法を使います。DNSサーバは、自分のドメイン内のホスト名とIPアドレスの対応や、自分の1階層下の子ドメインのDNSサーバのIPアドレスなどの情報を持っており、寄せられる名前解決の依頼に以下の3通りの反応を返します。

（1） 自分のドメイン内のホストに関する依頼ならば、自分で答える。

（2） 自分の子ドメインについての依頼ならば子DNSサーバに尋ねる。

（3） 上記のどれでもなければ、自分の親DNSサーバに尋ねる。

また、解決した案件について何回も他のDNSサーバに尋ねることをしないために、一定時間答えを記憶しておくキャッシュも持っています。このようにして、世界中のホストの名前解決が行えるようになっています。

3.7 基本的なネットワークの設定

本賞ではARMAでの基本的なネットワークの設定について説明します。

3.7.1 インターフェースの確認

ネットワークの設定がおこなわれるためにはハードウェアであるネットワークアダプタのデバイスドライバが設定されている必要があります。これはifconfigコマンドで確認することができます。以下がifconfigコマンドの出力例となります。

```
# ifconfig -a
eth0        Link encap:イーサネット ハードウェアアドレス
00:13:a9:8f:92:5e
          inet アドレス:192.168.0.209 ブロードキャスト:192.168.0.255
           マスク:255.255.255.0
          inet6 アドレス: fe80::213:a9ff:fe8f:925e/64 範囲:リンク
          UP BROADCAST RUNNING MULTICAST  MTU:1500 メトリック:1
          RX パケット:1869 エラー:0 損失:0 オーバラン:0 フレーム:0
          TX パケット:1033 エラー:0 損失:0 オーバラン:0 キャリア:0
          衝突(Collisions):0 TX キュー長:1000
          RX バイト:2321809 (2.2 MiB)  TX バイト:89645 (87.5 KiB)

lo        Link encap:ローカルループバック
          inet アドレス:127.0.0.1 マスク:255.0.0.0
          inet6 アドレス: ::1/128 範囲:ホスト
          UP LOOPBACK RUNNING  MTU:16436 メトリック:1
          RX パケット:73 エラー:0 損失:0 オーバラン:0 フレーム:0
          TX パケット:73 エラー:0 損失:0 オーバラン:0 キャリア:0
```

```
        衝突 (Collisions):0 TX キュー長 :0
         RX バイト :30953 (30.2 KiB)  TX バイト :30953 (30.2 KiB)
```

eth0、lo などが OS レベルのインターフェース名となりますが、機器の MAC アドレスに対応するインターフェースが出力されていな場合はドライバの設定が必要となりますので、以降に進む前に先にそちらを設定してください。

3.7.2 管理ツールによるネットワーク設定

ARMA では基本的なネットワークの設定は管理ツールを使っておこなうことができます。ネットワークの設定は、管理ツールの「ネットワーク」-「ネットワーク設定」から開始します。ツールでの設定はメニューに従って設定すれば問題ありませんが、設定のポイントになる考え方について説明します。

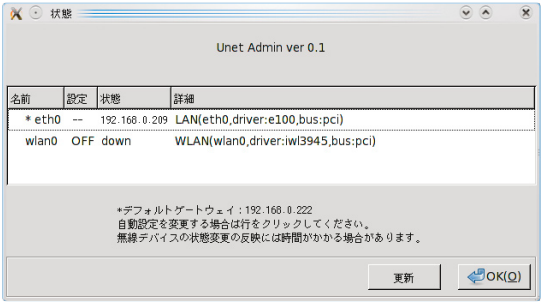
管理ツールはネットワークをルールベースで設定します。ルールはデバイスに対するルール、ドメインに対するルール、ESSID / SSID に対するルールがあり、有線順位に応じて検出したルールパターンが適用されます。ルールパターンは「固定 IP / DHCP / なし」から選択します。

モバイル端末では接続する可能性のあるシチュエーションに従って複数のルールを作成します。この場合は動的ネットワークデーモン（dynetd）が必要になり、dynetd が自動的に起動されます。そうでない通常のホストではデバイスルールを作成し「固定 IP / DHCP」から選択します。この場合は dynetd は必要ありません。

ネットワークの設定フローはインストーラと同じですので、より詳しいネットワーク設定の各メニューについては、マニュアルの「インストール」のネットワーク部を参照してください。

3.7.3 ogl-umin によるネットワークの確認とホットプラグデバイス

dynetd を使用している場合、デスクトップ上でネットワークの状態を確認することかできます。GNOME では右上のアプレット、KDE では左下のシステムトレイ内の該当アイコンにマウスカーソルを合わせるとネットワークの状態がポップアップします。そこで右クリックするとメニューが出ますので「ネットワークの状態」を選択すると下記のような画面が表示されます。



USB 等のホットプラグのデバイスは、接続しドライバが自動的にロードされただけでは dynetd の対象となりません。そのような「新しい」デバイスにドメインや ESSID / SSID ベースのネットワークルールを適用する場合、この確認画面の設定行をクリックして「ON」になるようにしてください。

3.7.4 手作業でのネットワーク設定

本節では管理ツールと同等の管理を直接おこなう場合の方法について説明します。

/ etc / network / interfaces

通常のネットワーク設定は主に /etc/network/interfaces に記述されます。このファイルは設定例としては下記のようになります。

```
auto lo eth0
iface lo inet loopback
iface eth0 inet static
  address 192.168.0.2
  netmask 255.255.255.0
  network 192.168.0.0
  broadcast 192.168.0.255
  gateway 192.168.0.1
iface eth1 inet dhcp
```

管理ツールで ESSID 等の動的なルールやホットプラグデバイスの設定は dynetd が受け持ちますが、デバイスルールを作成した場合はこのファイルに設定がされます。

auto 行には、ARMA 起動時に有効にするインターフェース名を指定します。ここに指定しないカードも 後述の ifup で有効にできます。

iface 行には、ネットワークデバイスの認識名・ネットワーク層のプロトコル・IP アドレスを取得する方法の 3 つのパラメータを指定します。認識名にはループバックなら lo、ネットワークデバイスなら eth0, eth1… 等を指定します。

IPv6 は inet6 とします。

プロトコルには、通常は IPv4 を表す inet を指定します。IP アドレスを取得する方法は loopback, static, dhcp から選びますが、どれを選択したかで続く行が異なります。

loopback は lo に対して設定するもので、続く設定はありません。

自動的にループバック扱いとして 127.0.0.1 が割り当てられます。

static はネットワークインターフェースに固定 IP アドレスを割り当てる場合に使い、続けて以下の項目を設定します。

address	ネットワークデバイスの IP アドレス
netmask	サブネットマスク
network	ネットワークアドレス（省略可）
broadcast	ブロードキャストアドレス（省略可）
gateway	デフォルトゲートウェイアドレス

そのホスト自身がゲートウェイは上流のゲートウェイを指定します。

デフォルトゲートウェイは、ルーティングテーブルで特に指定がない宛先のパケットを送信するホスト（ブロードバンドルータなど）のアドレスを指定します。ネットワークアドレスやブロードキャストアドレスを省略した場合は、それぞれネットワーク内の最小・最大の IP アドレスが使われます。

dhcp は DHCP で IP アドレスを自動的に取得する場合に使います。続く設定はありません。

ネットワークデバイスの有効化（up）・無効化（down）

/etc/network/interface に記述した ネットワークデバイスは ifup,ifdown コマンドで有効化 / 無効化できます。ネットワークデバイスの設定を変更した場合も ifdown で一旦無効にしてから ifup すれば設定が反映されます。

```
# ifup < ネットワークデバイスの認識名 >
# ifdown < ネットワークデバイスの認識名 >
```

3.7.5 名前解決

名前を解決するための設定ファイルとしては /etc/hosts と /etc/resolv.conf をあげることができます。ほとんどのプログラムは名前を解決するために共通のルーチンを使用しており、このルーチンをレゾルバと呼びます。レゾルバはまず /etc/hosts を参照し、つぎに /etc/hosts.conf を調べて DNS サーバを使って名前を解決しようとします。

/ etc / hosts の設定

/etc/hosts は、DNS が開発される前から存在する原始的な名前解決の方法です。

DNS が普及した現在では /etc/hosts は存在しなくても動きますが、DNS のトラブル時やシステム起動時に DNS にアクセスできない段階での名前解決、あるいは敢えて DNS を導入するまでもない小規模な LAN などのために残されています。無用なトラブルを避けるためにも、localhost と固定 IP を持っているホストなら、自ホストなどの情報を /etc/hosts に書いておいた方がよいでしょう。

```
127.0.0.1    localhost
192.168.0.1  cosmos.omoikane.co.jp       cosmos
192.168.0.2  kakitsubata.omoikane.co.jp  kakitsubata
192.168.0.3  syakuyaku.omoikane.co.jp    syakuyaku
```

/ etc / resolv.conf の設定

/etc/resolv.conf は DNS に関連する情報を設定します。DHCP や PPxP などを使っている場合は、システムが適切に /etc/resolv.conf を書き出してくれます。

```
search omoikane.co.jp, debian.org
nameserver 192.168.0.1
nameserver 192.168.0.2
```

search 行には、ドメイン名が省略した場合に検索するドメイン名を "," で区切って指定します。この例では、ホスト名を cosmos と指定すると cosmos.omoikane.co.jp,cosmos.debian.org の順に DNS サーバに問い合わせます。ただし、アクセスできないドメイン名を search に指定すると処理が止まってしまうので要注意です。

nameserver 行には、レゾルバが問い合わせをする DNS サーバの IP アドレスを指定します。複数の DNS サーバを使う場合は、上の例のように複数の nameserver 行を書けば、書いた順に利用されます。

さらに詳しく説明すると、この上位に /etc/nsswitch.conf 設定ファイルがあり、ここで hosts や resolv.conf をアクセスする順番が決められています。NIS のような他の名前解決システムを組み込む場合は /etc/nsswitch.conf を設定します。

通常は自ホストについてはインストーラが設定しておいてくれます。

/etc/resolv.conf がない場合は、search に自分のドメイン、nameserver にもループバックで自分 127.0.0.1 を指定したものと見なされます。

3.7.6 ルーティングテーブルの確認

ルーティングテーブルの設定は route で行います。

ただし、マシンをルータとして使わなければ、/etc/network/interfaces や /etc/pcmcia/network.opts に基づいて自動的に設定されるデフォルトゲートウェイだけでルーティングの設定は十分です。

よって、ここでは route でルーティングテーブルを確認する方法だけを簡単に説明します。

```
# route
Kernel IP routing table
Destination     Gateway         Genmask       …略…Iface
192.168.0.0     *               255.255.255.0…略…eth0
0.0.0.0         cosmos.omoikan  0.0.0.0       …略…eth0
```

この例では、宛先が自分のネットワーク 192.168.0.0/24 ならば eth0 から直接送信し、それ以外は cosmos に eth0 からパケットを送る設定になっています。つまり、cosmos がデフォルトゲートウェイです。

名前解決できないホストへのルーティングがあると、route は表示の途中で止まってしまいます。このような場合は、route -n と指定してください。

3.7.7 ネットワーク設定の確認

次に、IP レベルでの接続を確かめるため、「自分以外の」手近な稼働中のマシンに ping を打ちます。放っておくと止まりませんので、適当なところで［Ctrl］+［C］を押して終了させます。

```
# ping cosmos
PING cosmos.omoikane.co.jp (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=0.6 ms
【[Ctrl] + [C] を押す】
--- cosmos.omoikane.co.jp ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.6 ms
```

通常、同一 LAN 内のマシンに ping する場合は 10ms 以内に全てのパケットが帰ってきます。次のように、"Network is unreachable" の場合は、ルーティングが誤っている場合が多いでしょう。

```
# ping cosmos
PING cosmos.omoikane.co.jp (192.168.0.1): 56 data bytes
ping: sendto: Network is unreachable
ping: wrote 192.168.0.1 64 chars, ret=-1
ping: sendto: Network is unreachable
ping: wrote 192.168.0.1 64 chars, ret=-1
【[Ctrl] + [C] を押す】
--- 192.168.0.1 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

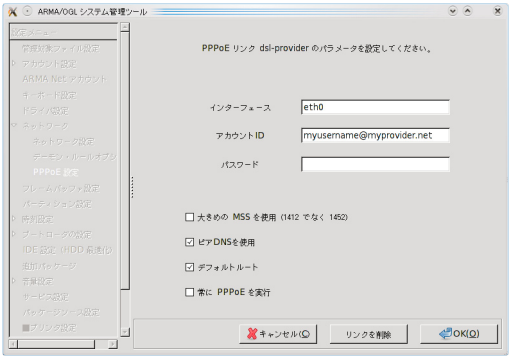
また、何の応答もない場合は IP の設定が間違っている可能性があります。

3.7.8 PPPoE の設定

管理ツールによる PPPoE 設定

管理ツールでは「ネットワーク」-「PPPoE 設定」で PPPoE の設定をおこなうことができます。

ここでリンク名を選択して各パラメータを設定してください。



管理ツールでは起動時の設定も可能ですが、ここで例えば dsl-provider という名前でリンクを作成した場合、

```
# /etc/init.d/pppoe-dsl-provider start
# /etc/init.d/pppoe-dsl-provider stop
```

といった形で手作業で PPPoE リンクを張ったり停止したりすることができます。

コマンドレベルの PPPoE 設定

PPPoE（PPP over Ethernet）は、PPP パケットをシリアルポートでなく、イーサネットによる PPP リンクです。この PPPoE の設定は事業者によって若干異なりますが、以下では NTT 東 / 西日本の FLET'S ADSL を例にして説明します。

まず、PPPoE 接続自体の設定は /etc/ppp/peers/dsl-provider に書きます。pppoe パッケージをインストールした時点で既に雛形はできているので、ここでは最低限変更を加えるべき 3 行について説明します。

```
user junko@omoikane.co.jp
pty "pppoe -I eth1 -T 80 -m 1414"
mtu 1454
```

user 行には、プロバイダから指定された < ユーザ >@< ホスト > 形式のアカウント名を指定します。メールアドレスと似ていますが、無関係なので注意してください。

pty 行には、PPPoE クライアントの pppoe を起動するコマンドラインを書きます。変更が必要なのは、-I に続けて書く ADSL モデムに繋がるネットワークインタフェース名だけです。これ以外のパラメータは上の例のままの値にしておいてください。調整が必要な場合は man pppoe を参照して

ください。

mtu 行には、MTU サイズをバイト単位で指定します。FLET'S ADSL では 1454 が最適とされていますが、他の DSL 事業者では状況が異なることもあります。DSL 事業者が提供する情報などを参考にして、適切な値を設定してください。

続いて、/etc/ppp/pap-secrets の最後に、プロバイダに接続するために必要なパスワードの情報を追加します。

```
junko@omoikane.co.jp   omoikane.co.jp   eplihtayz6aArmdy
```

左から、/etc/ppp/peers/dsl-provider で指定したアカウント名、プロバイダのホスト名、パスワードを指定します。なお、この /etc/ppp/pap-secrets は root 以外には読めないようにパーミッションを付けてください。

さて、続いて /etc/ppp/ppp_on_boot.dsl を編集します。書き換える行は次の 1 行だけで、/etc/ppp/peers/dsl-provider の pty 行と同じように、ADSL モデムの繋がったネットワークインタフェース名を指定します。

```
INTERFACE=eth1
```

さらに、/etc/ppp/ppp_on_boot.dsl を指すように /etc/ppp/ppp_on_boot というシンボリックリンクを作ります。

```
# ln -s /etc/ppp/ppp_on_boot.dsl /etc/ppp/ppp_on_boot
```

お疲れさまでした。下のようにして PPPoE を起動させれば ADSL / PPPoE 接続が始まります。接続できているかどうかは ifconfig で ppp0 というネットワークインタフェースを確認します。

```
# /etc/init.d/ppp start
# ifconfig ppp0
ppp0      Link encap:Point-to-Point Protocol
          inet  addr:XX.XX.XX.XX   P-t-P:XX.XX.XX.XX
Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1454  Metric:1
          RX packets:11024 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10601 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:3231161 (3.0 Mb)  TX bytes:1835426 (1.7 Mb)
```

3.8 ssh による暗号化通信

3.8.1 安全な通信のための暗号化

コンピュータネットワークにおけるセキュリティの重要性は、近年ようやく広く認識されるようになりました。しかし、よく言われる「セキュリティ」は「侵入者から」の防衛が主で、もうひとつ重要な「盗聴者から」の防衛即ち「暗号化通信」に対する意識はそれほどでもないようです。

インターネットがルーティングで成り立っている限り、その途中でパケットが盗聴される可能性はゼロにはなりません。そこで、盗聴されないのではなく、盗聴されても内容が分からないようにするというのが暗号化通信の発想です。

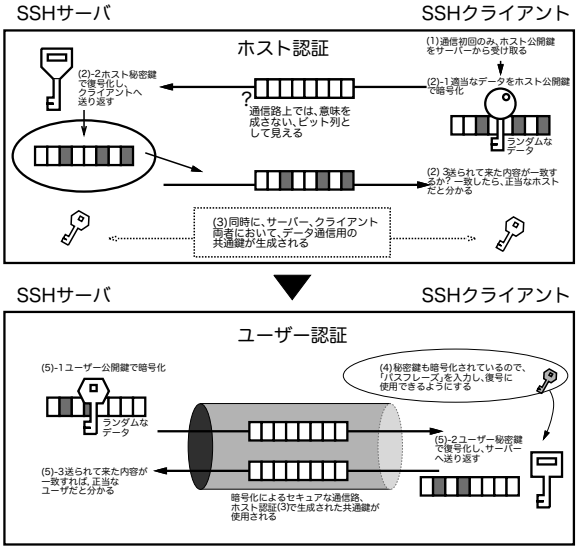
ARMA では、暗号化通信システムとして SSH を標準採用しています。SSH は公開鍵暗号方式で暗号化通信を実現するシステムで、telnet,rsh,rcp,ftp によるリモートシェルやリモートファイルコピーの代わりに ssh, scp, sftp を提供します。特に ARMA のデフォルトでは telnetd, rsh, rcp を提供していないので、SSH は必須になります。

あえて rsh を使用する場合は、ARMA Net リポジトリから rsh-client パッケージを導入してください。

3.8.2 ssh の仕組み

公開鍵暗号システムでは、「公開鍵」と「秘密鍵」の２つの電子的なデータの「鍵」からなる「鍵セット」を使います。この２つの鍵は、その名前の通り公開するか秘密にするかの違いはありますが、機能的な差はなく、どちらも「片方の鍵で暗号化したら、もう片方でしか復号できない」という特徴を持っています。もちろん、片方の鍵からもう一方を類推することは不可能になっています。

この性質を生かして、SSH では以下の要領で通信を行います。



（1） 初めて接続するサーバなら、クライアントはサーバからそのホスト公開鍵を受けとり、鍵リストに加える。

（2） クライアントは、適当なデータを鍵リストにあるホスト公開鍵で暗号化してサーバに送り、サーバは自分のホスト秘密鍵で復号化して返す。クライアントは、送信したデータと返信が一致すればサーバを正しいと認める。（ホスト認証成功）

（3） ホスト認証終了と同時に、サーバ・クライアント間の通信を暗号化する共通鍵が生成され、以降の通信は全てこの共通鍵で暗号化する。

（4） ユーザは「パスフレーズ」を入力する。パスフレーズは、暗号化されて記録されているユーザ秘密鍵を復号化し、使えるようにする。

（5） サーバは、適当なデータをユーザ公開鍵で暗号化してクライアントに送り、クライアントはユーザ秘密鍵で復号化してサーバに返す。サーバは送信したデータと返信が一致すればユーザを正しいと認める。（ユーザ認証成功）

> 公開鍵で暗号化されたデータを正しく復号化できるのはペアの秘密鍵のみです。ですから、ホスト認証のときにホスト公開鍵で暗号化したデータを正しく復号化できるのは、ホスト秘密鍵を知っている正しいサーバだけということになります。このホスト認証によってはじめて、接続するサーバに誰かが「なりすまし」ていないことが証明できるわけです。もちろん、ホスト秘密鍵が漏れてはホスト認証の意味がありませんので、秘密鍵はきちんと管理しなければなりません。ユーザ認証についても、これと同じ仕組みで認証が行われています。

「パスフレーズ」がユーザ認証時の交信内容に含まれないことにも注目してください。また、仮にパスフレーズが漏洩してもユーザ秘密鍵にアクセスできなければユーザ認証を突破できませんが、ユーザ秘密鍵にアクセスするために必要なユーザのパスワードは、この一連の流れの中で一度も出てきません。このようにパスフレーズは、ログイン権限とパスワードの奪取を防ぐための砦になるものなので、「パスワードと違う」文字列でなければなりません。

ユーザ認証時にパスフレーズが使えない場合、次善の策としてパスワードを使うこともありますが、これも（3）で作られた共通鍵で暗号化された通信路の中をパスワードが通っているので、単に盗聴してもパスワードは分かりません。これに対し `telnet` や `rsh` による非暗号化通信では、パスワードがそのままネットワーク上を流れるため、通信を盗聴されると簡単にパスワードが漏洩してしまい大変危険です。

3.8.3 ssh の設定

SSH の実装には、大きく分けて SSH 社による SSH Tectia と、OpenBSD プロジェクトによる OpenSSH があります。前者は商用ソフトですが、後者は BSD ライセンスのフリーソフトです。ARMA では Debian と同じく OpenSSH を標準として採用しており、ssh パッケージに含まれる SSH の実装も OpenSSH です。

また、SSH には SSH1 と SSH2 のプロトコルがありますが、両者の間には暗号化の方式などあまり互換性がありません。OpenSSH は両方のプロトコルを扱えますが、設定は SSH1,2 を別々に行う必要があります。

❗ SSH1 プロトコルは脆弱性が発見されているため利用は推奨されません。

ホスト認証用鍵セットの作成

SSH サーバのホスト認証用の鍵セットや設定ファイルは /etc/ssh に格納されています。ssh パッケージをインストールした時点で、既に一通りの設定とホスト認証用の鍵セットはできていますが、万一の場合に備え ssh-keygen で鍵セットを再生成する方法を説明します。

```
# ssh-keygen -t <暗号化タイプ> -f <秘密鍵ファイル名> -N
```

暗号化のタイプには SSH1 の RSA と、SSH2 の RSA, DSA があり、それぞれ rsa1, rsa, dsa と指定します。またホスト秘密鍵のファイル名は、以下のようにプロトコル毎に決まっています。一緒に作られる公開鍵は、秘密鍵に .pub を付けたファイル名になります。-N はパスフレーズで秘密鍵を暗号化しないようにするオプションで、ホスト認証用の鍵セットを作るときには指定します。

SSH2 では RSA が一般的ですので、DSA の鍵はここでは作りませんが、作る場合は /etc/ssh/ssh_host_dsa_key という名前にします。

```
# ssh-keygen -t rsa  -f /etc/ssh/ssh_host_rsa_key
```

ここで、生成された秘密鍵の属性は所有者・所有グループが root/root で、パーミッションが 600 = -rw------- であること、公開鍵の属性は所有者・所有グループが root/root で、パーミッションが 644 = -rw-r--r-- であることを確認してください。秘密鍵が root 以外に読めるようでは SSH が台無しです。

安全のため、間違ったパーミッションでは sshd が起動しないようになっています。

```
$ ls -l ssh_host_key_*
-rw-------  1 root  root     668 Aug  1 12:28 ssh_host_rsa_key
-rw-r--r--  1 root  root     600 Aug  1 12:28 ssh_host_rsa_key.pub
```

ユーザ認証用鍵セットの作成と配置

続いて、SSH クライアント機にログインして、ユーザ認証用の鍵セットを作成します。ユーザ認証用の鍵のファイルには、暗号化タイプに対応した決まった名前が使われますので、ファイル名を指定する -f オプションは使わず、-t オプションのみを指定して下さい。

```
$ ssh-keygen -t rsa
```

これにより、~/.ssh に SSH2 / RSA 用の id_rsa, id_rsa.pub の合計２つの鍵ファイルができます。ホスト認証鍵の時と同じように、所有者・所有グループとパーミッションが正しいかどうかを確認しておきましょう。

```
$ ls -l id_rsa* identity*
-rw-------    1 junko    users         736 Nov  4 17:05 id_rsa
-rw-r--r--    1 junko    users         330 Nov  4 17:05 id_rsa.pub
```

続いて、各プロトコルの公開鍵を SSH サーバ機にコピーします。このとき、SSH2 / RSA の公開鍵 ~/.ssh/id_rsa.pub をサーバ機の ~/.ssh/authorized_keys に、名前を変えてコピーします。もちろん、これは公開鍵ですからどんな方法でコピーしてもかまいません。また、2つ以上の公開鍵をサーバ機に保存する場合は、それぞれの公開鍵を cat などで連結します。

3.8.4 SSH

それでは、ssh で SSH サーバに接続します。

```
$ ssh [-C] [<ユーザ>@]<ホスト> [<コマンド>]
```

-C では通信を gzip 圧縮します。圧縮通信は ADSL 級以下の低速回線では有効ですが、LAN などの高速通信では大量のデータの圧縮の方に手間取って逆効果になる場合があるので、CPU への負荷も考慮に入れつつ適宜指定してください。さて、初めて接続するサーバの場合は、サーバの公開鍵を ~/.ssh/known_hosts に登録する際に以下のような質問をされますが、問題なければ yes と答えます。更に続いて SSH サーバが以下のようにパスフレーズを聞いてくれば登録成功です。

例えばすでに圧縮されているファイルを転送する場合などは -C の効果はほとんどなくなります。

```
$ ssh  cosmos
The authenticity of host 'cosmos (X.X.X.X)' can't be established.
RSA key fingerprint is xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx.
Are you sure you want to continue connecting (yes/no)?  yes
Warning: Permanently added 'cosmos,X.X.X.X' (RSA) to the list of known
hosts.
Enter passphrase for key '/home/junko/.ssh/id_rsa':
```

パスフレーズによる認証ができない場合は、SSH サーバの設定により、次善の策としてパスワードによる認証をするか、そのまま接続失敗します。

```
$ ssh cosmos
junko@cosmos's password:
```

```
$ ssh cosmos
Permission denied (publickey,keyboard-interactive).
```

ホスト認証に失敗した場合は、次のような目立つ警告が出ます。サーバの鍵セットが変わったことを知っている場合は、~/.ssh/known_host, ~/.ssh/known_host の中の、そのサーバの公開鍵だけを削除してから再度接続し、新しい公開鍵を登録させましょう。サーバの鍵セットが変わった覚えがない場合は、何物かがそのサーバに「なりすまし」ている可能性もあるので、う

かつに接続せずサーバの管理者に問い合わせてください。

```
$ ssh cosmos
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle
attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx:xx.
Please contact your system administrator.
Add correct host key in /home/junko/.ssh/known_hosts to get rid of this
message.
Offending key in /home/junko/.ssh/known_hosts:30
RSA host key for cosmos has changed and you have requested strict
checking.
```

さて、ログインした後はサーバのシェルとの対話になります。ログアウトする場合は exit でログインシェルを抜けます。

ユーザ名を指定すれば、サーバに違うユーザ名でログインできます。(ユーザ名を指定しなければ、現在のユーザ名でログインを試みます) ただし、特にサーバが許している場合以外は root でのログインはできません。

```
$ ssh user@hostname
```

また、コマンドを直接指定して、そのコマンドだけをサーバ機で実行させることもできます。次の例では、サーバのディスク容量をチェックしています。

```
client:$ ssh hostname df ↓ このコマンドは SSH サーバ機で実行
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/sda1              4016088   1767132   2248956  45% /
/dev/sda2             32125200   6077056  26048144  19% /home

client:$          ← このプロンプトは SSH クライアント機が出力
```

SSH では、X プロトコルの転送もできます (設定により禁止することもできます)。X プロトコルの転送を使うと、SSH サーバの X アプリのウィンドウを手元の SSH クライアントのデスクトップに表示することができます。クライアントでは -X オプションの指定が必要です。

```
client:$ ssh -X hostname xosview
```

scp, sftp, rsync でファイル転送

scp は SSH の暗号化通信を使ってファイルコピーを行う、rsh のセキュリティ強化版コマンドです。

```
$ scp [-p] [-r] [-C] <ユーザ名>コピー元(複数可)> <コピー先>
```

-p　コピー元のタイムスタンプ・所有者・パーミッションを保持
-r　ディレクトリの中身をまとめてコピー
-C　通信内容を gzip 圧縮（ssh と同じ）

コピー元やコピー先は [<ユーザ>@][<ホスト>:]<パス> の形式で指定します。例えば tobiume にユーザ名 sayuri でログインし、/tmp/report.txt を自分のマシンのホームディレクトリに scp する場合は次のようにします。

```
$ scp sayuri@tobiume:/tmp/report.txt ~
```

sftp は、ftp と同様の操作感で暗号化ファイル転送を行うコマンドです。ただし、あまり高機能ではなく、ssh と scp の連携でファイルの確認と転送はできることもあるので、出番は多くないかもしれません。

```
$ sftp [-C] [<user>@]<hostname>
```

-C　通信内容を gzip 圧縮（ssh と同じ）

rsync は、内部的に ssh, scp を利用してディレクトリ以下の内容を同一にします。rsync を使ってミラーリングやバックアップをすることができます。ここまで rsh, rcp の SSH 版として ssh, scp を紹介してきましたが、ssync(?) というコマンドはないので注意してください。

```
$ rsync [<オプション>] <コピー元(複数可)> <コピー先>
```

-r	ディレクトリの中身をまとめてコピー
-u	新しいファイルのみコピー（アップデート）
--delete	コピー元にないファイルはコピー先から削除
-l, -H	シンボリックリンク・ハードリンクを保持
-L	シンボリックリンクを参照するファイルに変換
-D	デバイスファイル属性を保持（要 root 権限）
-o, -g	所有者・所有グループを保つ（-o は要 root 権限）
-p	パーミッションを保つ
-t	タイムスタンプを保つ

-a	-rlpogtD と等しい（要 root 権限）
-x	コピー元でファイルシステムをまたがった処理をしない
-n	処理内容を表示するだけで実際には処理しない

「～を保ってコピー」とは、コピー元のファイルと同じ属性をコピー先でも設定することを意味します。例えば、同じファイルを単純に rsync, rsync -og, rsync -t の３通りでコピーすると次のようになります。

```
# ssh tobiume ls -l /tmp/report.txt
-rw-r--r--    1 junko    users       12345 Sep  8 07:06 report.txt
# rsync tobiume:/tmp/report.txt .
# ls -l file1
-rw-r--r--    1 root     root        12345 Nov 11 11:11 report.txt
# rsync -og tobiume:/tmp/report.txt .
# ls -l file1
-rw-r--r--    1 junko    users       12345 Nov 11 11:11 report.txt
# rsync -t tobiume:/tmp/report.txt .
# ls -l file1
-rw-r--r--    1 root     root        12345 Sep  8 07:06 report.txt
```

シンボリック属性の保持も原理は同じです。-l ではシンボリックリンクのままコピーされますが、-L ではリンクが指していたファイルとしてコピーされています。

```
# ssh tobiume ls -l /tmp/r*
lrwxrwxrwx    1 junko    users          10 Sep  8 07:06 readme
     -> report.txt
-rw-r--r--    1 junko    users       68030 Sep  8 07:06 report.txt
# rsync -l tobiume:/tmp/r* .
lrwxrwxrwx    1 junko    users          10 Nov 11 11:11 readme
     -> report.txt
-rw-r--r--    1 junko    users       68030 Nov 11 11:11 report.txt
# rsync -L tobiume:/tmp/r* .
-rw-r--r--    1 junko    users       68030 Nov 11 11:11 readme
-rw-r--r--    1 junko    users       68030 Nov 11 11:11 report.txt
```

-H も仕組みは同様で、同じファイルの実体を２つの名前がハードリンクしている場合、コピー先でもこの状態を保ちます。

ただし、処理が遅くなるので必要な場合だけ指定するのがよいでしょう。

-D では、コピー元がデバイスファイルならば、コピー先でも同じデバイスファイルを作ります。

-x では、コピー元のあるファイルシステム以外の部分はコピー対象外になります。例えば rsync -xr / cosmos:/backup とすると、/ と違うファイル

システムである /proc 以下はコピーされません。

パスフレーズの転送

SSH を使いこなすようになってくると、1 日に何十回もの ssh, scp の度のパスフレーズの入力が面倒になります。これを解消するために、前もって ssh-agent, ssh-add というコマンドでユーザ秘密鍵を記憶させておいて、毎回の入力を省略できるようにする方法があります。

まず、ssh-agent を通してシェルを起動します。

グラフィカルログイン（kdm）を使用している場合は ssh-agent の実行は省略できます。kdm が ssh-agent を実行するためです。

```
$ ssh-agent $SHELL
```

次に ssh-add でパスフレーズを入力すると、パスフレーズによって復号化された秘密鍵が内部的に記憶されます。これ以降 ssh-agent が起動したシェルを終了するまでは ssh, scp などでのパスフレーズの入力を省略できます。

```
$ ssh-add <秘密鍵ファイル（複数可）>
```

秘密鍵のファイル名は SSH1 なら ~/.ssh/identity 、SSH2 / RSA なら ~/.ssh/id_rsa です。通常は SSH2 を使いますから、次のようになります。

```
$ ssh-add ~/.ssh/identity ~/.ssh/id_rsa
Need passphrase for identity
Enter passphrase for junko@tobiume
Identity added: id_rsa (id_rsa)
```

秘密鍵の記憶は ssh-agent で起動したシェルを抜けない限り、マシンが変わっても引き継がれます。

```
tobiume:~$ ssh-agent $SHELL
tobiume:~$ ssh-add ~/.ssh/identity ~/.ssh/id_rsa
Need passphrase for identity
Enter passphrase for junko@tobiume
Identity added: identity (junko@tobiume)
Identity added: id_rsa (id_rsa)
tobiume:~$ ssh cosmos
Last login: Thu Oct 11 15:09:33 2009 from tobiume.omoikane.co.jp on
pts/0
Linux cosmos 2.6.31.6 #1 SMP Wed Oct 10 16:01:09 JST 2009 i686 unknown
cosmos:~$ ssh syakuyaku
Last login: Thu Oct 11 15:09:33 2009 from cosmos.omoikane.co.jp on
pts/0
Linux syakuyaku 2.6.31.6 #1 SMP Thu Oct 11 19:20:34 JST 2009 i686
unknown
syakuyaku:~$
```

ssh-add には、GNOME 環境用のパスフレーズ入力ウィンドウを備えたものもあります。これを使うには、ssh-agent startx として X を起動し、~/.xinitrc 内で gnome-session の前に以下の 1 行を加えます。すると X を終了するまでは全て ssh-agent のおかげでパスフレーズの入力が不要になるので X 上での作業が大変楽になります。

```
ssh-add ~/.ssh/identity ~/.ssh/id_rsa < /dev/null &
```

この設定はデスクトップを選択するインターフェースでも行うことができます。(単独で起動する場合は ogl-umin コマンドを実行してください。)

3.9 X Window System

3.9.1 X Window System

X Window System (以下 "X") は、UNIX 系 OS を中心に広く標準的に使われている GUI システムで、サーバ・クライアント方式を採用し、ネットワークにも対応しているという特徴があります。

もう少し詳しく説明すると、X サーバはキーボード・マウス・ビデオカードなどの GUI コンソールの入出力機器を制御し、クライアントの要求に従って入力された内容を伝えたり画面を実際に描画したりするのが役目です。これに対し X クライアントは、Mozilla や Sylpheed などのいわゆる「GUI プログラム」と呼ばれるものであり、サーバから受け取った入力機器の情報に基づいて処理をし、結果を X サーバに描画してもらっています。また、X サーバ・クライアント間の通信は X プロトコルによって行われるので、他のマシンにログインして起動した X クライアントを手元のマシンの X サーバで操作することもできます。

ARMA も含め Linux システムでは XFree86 Project を引き継いだ Xorg Foundation による Xorg という X サーバが一般的です。ARMA では長らく使われた XFree86 4.3 から、後継の Xorg 7.4 を採用しています。

3.9.2 xorg.conf

X サーバの設定ファイルは /etc/X11/xorg.conf です。通常は X サーバの設定は管理ツールで設定しますが、設定がうまくいかない場合はこの賞を参考にしてください。

xorg.conf では、"#" 以降の文字はコメントと見なされます。また、内部はいくつかのセクションに分かれていますので、下記では各セクションについて説明します。

3.9.3 Files セクション

Xorg で使うフォントやカラーデータベースのパスを指定します。

```
Section "Files"
  FontPath "/usr/share/fonts/X11/misc:unscaled"
  FontPath "/usr/share/fonts/X11/75dpi:unscaled"
  FontPath "/usr/share/fonts/X11/100dpi:unscaled"
  FontPath "/usr/share/fonts/X11/misc"
  FontPath "/usr/share/fonts/X11/Speedo"
  FontPath "/usr/share/fonts/X11/Type1"
  FontPath "/usr/share/fonts/X11/CID"
  FontPath "/usr/share/fonts/X11/75dpi"
  FontPath "/usr/share/fonts/X11/100dpi"
  FontPath "/usr/share/fonts/X11/TrueType"
  FontPath "/var/lib/defoma/x-ttcidfont-conf.d/dirs/CID"
  FontPath "/var/lib/defoma/x-ttcidfont-conf.d/dirs/TrueType"
EndSection
```

FontPath 行には、フォントがあるディレクトリを指定します。通常は、例の通り /usr/share/fonts/X11 以下の misc, 75dpi, 100dpi, TrueType の各ディレクトリ等が指定されています。また、:unscaled は、そのディレクトリ内の unscaled なビットマップフォントを使うことを表しています。通常は unscaled を優先するため、例のような順番で FontPath を指定することになります。

Modules セクション

Xorg の拡張機能モジュールを読み込みます。

```
Section "Module"
  Load "dbe"
  Load "dri"
  Load "extmod"
  Load "glx"
  Load "record"
EndSection
```

Load 行のパラメータに、読み込むモジュールを１つずつ指定します。

モジュールの実体は /usr/lib/xorg/modules 以下にある lib*.so というファイルになります。例えば extmod の実体は /usr/lib/xorg/modules/extensions/libextmod.so です。

モジュールに内部オプションを指定するときは Load の代わりにサブセクションを使います。例えば extmod に omit XFre86-DGA というオプションを渡す

場合は下のようにします。

```
Section "Module"
(…中略…)
  SubSection "extmod"
    Option "omit Xorg-DGA"
  EndSubSection
(…中略…)
```

ServerFlags セクション

Xorg の挙動を決めるオプションを指定できます。ただしほとんどは、通常指定する機会がなかったり、GNOME や KDE のコントロールセンターで同じ設定ができるオプションなので、全て初期値のままでよいと思われます。詳細は man xorg.conf を参照してください。

InputDevice セクション

マウス、キーボードなどの入力機器を設定します。

```
Section "InputDevice"
    Identifier  "Keyboard0"
    Driver      "kbd"
    Option      "XkbRules" "xfree86"
    Option      "XkbModel" "jp"
    Option      "XkbLayout" "jp"
    Option      "XkbOptions" "ctrl:nocaps"
EndSection

Section "InputDevice"
    Identifier  "Mouse0"
    Driver      "mouse"
    Option      "Protocol" "ImPS/2"
    Option      "Device" "/dev/input/mice"
    Option      "Buttons" "5"
    Option      "ZAxisMapping" "4 5 6 7"
    Option      "Emulate3Buttons"
EndSection
```

Identifier 行には、後述の ServerLayout セクションで入力機器を識別するための ID（認識名）を指定します。

Driver 行には入力機器のドライバを、キーボードなら keyboard 、マウスなどのポインティングデバイスなら mouse と指定します。

Option 行では入力機器の仕様を設定しますが、ここでのキーボードの配置等は標準では使われません。実際には Xorg サーバは hal と呼ばれる仕組みによってキー配列を取得しています。実際のキーボードの配置は /etc/hal/fdi/policy/10-keymap.fdi で決定されます。もし hal を使わず InputDevice セクションのキーボード配列を有効にする場合は下記の設定を追加します。

```
Section "ServerFlags"
    Option "AutoAddDevices" "False"
EndSection
```

Protocol オプションではマウスのプロトコルを、ホイール無しマウスなら PS/2、マイクロソフト IntelliMouse 互換のホイールマウスなら IMPS/2 と指定します。通常ホイールマウスの多くは IntelliMouse 互換です。また USB マウスでもプロトコルは同じく PS/2, IMPS/2 です。

Device オプションではマウスのデバイスファイルを、PS / 2 なら /dev/psaux、USB なら /dev/input/mice と指定します。

! X には 3 ボタン以上のマウスが必要です

Emulate3Buttons と Emulate3Timeout オプションでは、2 ボタンマウスを「左右同時クリック＝中クリック」とすることで 3 ボタンマウスとして使えるようにする機能を設定します。Emulate3Buttons でこの機能を有効にし、Emulate3Timeout で中クリックと見なせる左右クリックの最大時間差をミリ秒単位で指定します。ちなみに、2 ボタン + 1 ホイールのマウスではホイールクリックが中クリックになるため、この機能は必要ありません。

Button と ZAxisMapping オプションでは、ホイールを有効にする設定をします。現状の X では、内部的にはホイールの回転をマウスの第 4,5 ボタンが押されているとして処理するため、2 ボタン + 1 ホイールの一般的なマウスでは先に挙げた例のような設定となります。

Monitor セクション

ディスプレイ（モニタ）の設定をします。

```
Section "Monitor"
  Identifier    "Monitor0"
  VendorName    "NERGAL"
  ModelName     "NG-CRT21LP"
  HorizSync     27.0 - 96.0
  VertRefresh   50.0 - 160.0
EndSection
```

Identifier 行では、後述の Screen セクションでディスプレイを識別するための ID（認識名）を設定します。

VendorName 行と ModelName 行では、それぞれディスプレイのメーカと型番

を指定できますが、特に設定する必要はありません。

HorizSync 行と VertRefresh 行ではディスプレイの水平・垂直周波数の範囲を設定します。ディスプレイの仕様書にはこの数値が書いてあるはずですので、水平は kHz 単位、垂直は Hz 単位で正確に指定してください。

後述の Screen セクションで解像度ごとにディスプレイが対応できるリフレッシュレート（1秒間のコマ数）を計算します。

Device セクション

ビデオカードの設定をします。

```
Section "Device"
    Identifier  "Card0"
    Driver      "psb"
    BusID       "PCI:0:2:0"
    Screen      0
    VendorName  "SomeVendor"
    BoardName   "unknown"
EndSection
```

Identifier 行では、後述の Screen セクションでビデオカードを識別するための ID（認識名）を設定します。

BusID 行は、ビデオカードを２枚以上使った場合に、どのビデオカードについての設定であるかを明示するために使います。従ってビデオカードが１枚しかない場合は指定する必要はありません。PCI バス ID は lspci で確認できます。

PCI Express 等にも PCI バス ID はあります。

```
# lspci | grep VGA
00:02.0 VGA compatible controller: Intel Corporation ( 略 )
```

Screen 行には、複数のディスプレイを接続できるビデオカードを使っている場合、何番目のコネクタについての設定であるかを 0,1,2… で指定します。従ってディスプレイを１つしか接続できないビデオカードでは設定する必要はありません。

デュアルヘッド・マルチヘッドなどと呼ばれる機能です。

Driver 行には、ビデオカード（チップ）のドライバ名を指定します。ビデオチップとドライバ名の対応はある程度明確ですが man < ドライバ名 > が参考になる場合もあります。

VendorName 行と BoardName 行では、ビデオカードのメーカ・型番を指定できますが、特に設定は必要ではありません。

Xorg 標準添付のドライバとは別に、nVIDIA や Matrox などのビデオチップメーカが独自に開発した Xorg 用ドライバを公開しているものもあります。これらは標準添付のものより完成度が高いこともあります。nVIDIA については ARMA では nvidia-display パッケージによって nvidia ドライバを追加することができるようになっています。

Screen セクション

Monitor, Device セクションの設定に基づき、X の出力機器としての「画面」の設定をします。

```
Section "Screen"
  Identifier "Screen0"
  Device     "Card0"
  Monitor    "Monitor0"
  DefaultDepth     24
  SubSection "Display"
    Depth     16
    Modes    "1024x768" "800x600" "640x480"
  EndSubSection
  SubSection "Display"
    Depth     24
    Modes    "1024x768" "800x600" "640x480"
  EndSubSection
EndSection
```

Identifier 行では、後述の ServerLayout セクションで画面を識別するための ID（認識名）を設定します。

Device 行と Monitor 行では、画面を構成するビデオカードとディスプレイを指定します。Device, Monitor セクションで設定した ID を指定してください。

DefaultDepth 行では、デフォルトの色深度をビット単位で指定します。ビデオチップ自体が対応している色深度でも、Xorg のドライバが対応していない場合もありますので、使える色深度に関しては各ビデオチップの man で確認してください。

Display サブセクションには、画面の色深度ごとに使う解像度を指定します。例のように Depth に色深度をビット数で、Modes に解像度を指定します。解像度は最初に指定したものが初期値で使われ、その他の解像度には Ctrl+Alt+ テンキー側の +,-（プラス・マイナス）キーで切り替えられます。

色深度は表示可能な色数の「容量」を指す用語です。例えば色深度が 8 ビットなら 256 色、16 ビットなら 65,536 色を表示する能力があることを表します。

Modes の指定は、正確には解像度ではなく "ModeLine" の名前を記します。

ServerLayout, InputDevice, Monitor, Device, Screen セクションは ID さえ違えれば、複数指定することができます。これにより、レイアウト・入力機器・ディスプレイ・ビデオカード・画面設定を使い分けることができます。

また、各セクションの間にある依存関係をまとめると、下記のようになります。

つまり、Xorg には統一設定の他に複数のレイアウトを定義でき、各レイアウトは入出力機器（Screen, InputDevice）の組み合わせからなり、出力機器はビデオカードとディスプレイの組でできていると捉えることができるわけです。

```
Xorg -+- Files, Modules, ServerFlags, DRI（全体に関わる設定）
        +
        `- ServerLayout -+- Screen ------+- Monitor
                         |               `- Device
                         `- InputDevice
```

3.9.4 ~/.Xresources

kterm, gv, XEmacs のような古典的な X のアプリケーションは、リソースファイルをデフォルトのパラメータとして使用します。システム全体のリソース設定は /etc/X11/app-defaults 以下に配置されます。ユーザ毎のリソース設定は ~/.Xresources です。これはシステム全体のリソース設定より優先されます。

ここでは詳しい設定方法については述べません。書き換えた設定を X を再起動せずに有効にするには、xrdb -m ~/.Xresources として下さい。

ARMA2.2 までは標準のリソースパスは ~/.Xdefaults でした。アップデート後にこちらを移行する場合は、~/.xinitrc-desktop に xrdb -m ~/.Xdefaults と記述してください。

3.9.5 テキストのコピー&ペースト

多くの X アプリケーションでは、テキスト部分をマウスで左ドラッグして選択すると反転した範囲の内容がバッファにコピーされます。そして、テキストを入力できる場所で中クリックすると、バッファの内容がマウスカーソルの位置にペーストされます。

3.10 時刻合わせの設定

3.10.1 時刻を正確に合わせる - NTP

PC 互換機はマザーボード上にボタン電池で動く時計を持っており、電源を切っても時刻設定が消えないようになっています。しかし一般的にはマザーボードの時計の精度は高くありませんので、定期的に時刻合わせをする必要があります。特に NFS を使っている場合などマシンごとに時計が違っていると、ネットワークでファイルをやり取りする際にどれが新しいか分からず間違って上書きしてしまうなど問題が出る可能性がありますので、マシンの時計はなるべく正確に合わせておいた方がよいでしょう。

ここではインターネットに接続されたコンピュータで一般的な NTP（Network Time Protocol）について説明します。NTP の仕組みは単純で、基本的に NTP サーバは時刻を知らせるだけ、NTP クライアントはそれに基づいて時刻を合わせるだけですが、通信時の遅延を計算に入れるなど、正確さのための工夫がされています。また、NTP サーバも自身がクライアントとして、より上位の NTP サーバとの間で時刻を合わせており、最上位の「ルート NTP サーバ」はたいてい原子時計などに直結しています。

通常の範囲では管理ツール（ogl-admin）による時刻設定で cron による定期的な ntpdate 設定をすれば十分でしょう。下記ではコマンドレベルの ntp の使用方法について説明します。

ntpdate

ARMA では NTP クライアントに `ntpdate` を採用しています。`ntpdate` の使い方は非常に簡単で、NTP サーバのホスト名を指定するだけです。

```
# ntpdate < ホスト >
```

NTP サーバの所在ですが、2009 年 11 月現在では http://www2.nict.go.jp/w/w114/tsp/PubNtp/index.html で告知されている `ntp.nict.jp` が国内の公開 NTP サーバとして有名です。自分の所属するプロバイダや大学などが NTP サーバを提供している場合はそちらを利用した方がよい場合もあります。Google などの検索サービスも活用し、なるべくネットワーク的に近く安定して通信できるサーバを探してみてください。

さて、正常に時刻を合わせられると以下のようなメッセージが表示されます。

```
# ntpdate ntp.nict.jp
8 Aug 08:08:08 ntpdate[8888]: step time server X.X.X.X offset +1.234567
sec
```

この秒数が正ならば時刻を合わせるために時計を進めたことを、負ならば逆に遅らせたことを意味します。この例では、時計を 1.2 秒余り進めています。