# cool-da

だってクールだ SLAX REMIX

# slax-remix


VLA & Moon: courtesy of NRAO/AUI
http://images.nrao.edu/Telescopes/VLA/93

# スラックス　リミックスは進化して、ポーティアス。


スラックスコミュニティーから飛び出してきたスラックス　リミックス、
今、すごい勢いで、進歩しています。
2011年1月、そのスラックス　リミックスに、新しい名前がつきました。
ポーティアス==Porteus
ポータブルとプロテウスを合体させた造語で、変幻自在の小海神のような意味とのこと。
山の神だとギョッとする方も、Proteusは、可愛い海の神様なので、安心です。

http://www.porteus.org/distro-download/latest-porteus-release-32bit.html
porteus-v09-i486 32ビット用は、ここから、archiveをクリックをします。
64ビットバージョンは、こちらにあります。
http://www.porteus.org/distro-download/download-latest-64-bit.html
archiveをクリックして、porteus-v09-x86_64を入手して下さい。
見た感じ、とっても洗練されて、きれいになりました。
すごい勢いで、進歩しており、快調です。
V1.0もかなり安定しており、気分よく使えます。

# ポーティアス　バージョン０９は。


porteus-v09 は、スカッシュ４ファイルシステムを採用しています。
sq3からsq4への変換は、カーソルをsq3ファイル上に置いて右クリックなので、
簡単ですが、すでにsq4のファイルも用意されています。
ここには、1000以上のsq4モジュールがリストされています。
http://code.google.com/p/fidoslax/downloads/list

この他、右クリックでsq4に変換できるのは、スラックウェアのtgz、txz。
レッドハット系のrpm、デビアン系のdebとなっています。
スラックス６にくらべても、便利な機能も加わっており、
実際に動かしてみること、おすすめです。
そのために、必要な部品は、

1. 未使用のCD、２～３枚。
2. GParted ライブCD、もしくは、確実にフォーマット、ブートフラグを立てる方法を確保。
3. USBペンドライブ(4GB, 16GB)

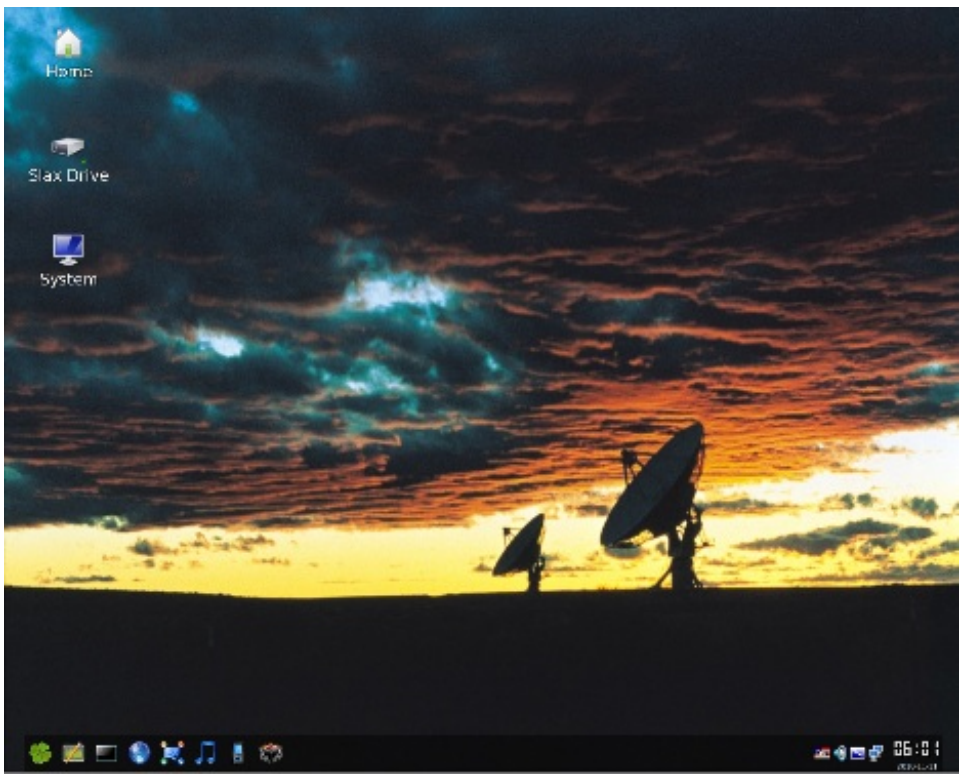
作業の流れとしては、

1. porteus-v09-i486.isoをダウンロード、CDに焼く。
2. GParted ライブCDを使って、USBペンドライブをフォーマット。
3. USBペンを差さないで、ポーティアスライブCDを立ち上げ、
   組み込みのプログラム、Porteus-2-usbを使って、USBペンドライブに、
   ポーティアスをインストール。
4. ブート専用のCDを作成。
5. 日本語入力のプログラムを整備。
   uimとanthyを使って、日本語入力用の簡単なlzmモジュールを作ります。

それでは、具体的な作業に入ります。

## ステージ１　ポーティアスを**CD**に焼く。


http://www.porteus.org/　にて、
porteus-v09-i486.iso　をダウンロード、アイソファイルをCDに焼きます。
サイズは、245.6MB。
日本語入力は、新たにモジュールを作る必要があります。
アイソファイルCDの焼き方を確認したい場合は、こちらを参照してください。
http://p.booklog.jp/book/9653/page/102703




VLA at Sunset: Image courtesy of NRAO/AUI
http://images.nrao.edu/Telescopes/VLA/92

## ステージ２　**USB**ペンドライブをフォーマット。


GParted ライブCDを作っておくと重宝します。
パーティションは、ウィンドウズ、リナックス、アップル、スラックス、
みんなが満足してくれないと、まずいことになります。
ライブCDからの操作は、かなり安定しています。この際、作っておきたい場合は、
http://sourceforge.net/projects/gparted/files/gparted-live-stable/0.7.1-5/gparted-live-0.7.1-5.iso/download
にて、gparted-0.7.1-5.isoを入手、CDに焼き、立ち上げます。

1. Gnome Partition Editor と大きなサインが出たらリタン。
2. Don't touch keymap と赤い字が出たら、リタン。
3. Which language do you prefer? で、15と入れ、リタン。
4. Which mode do you prefer? では、[0] となっているので、リタン。
5. これで、ジーパーテッドの画面が出ます。
   GParted(G)　-> デバイスを選択(D)  /dev/sdc (3.73 GIB)
6. フィールドは何本かあれば、１本ずつセレクトして、
   パーティション　-> delete をセレクト。
   すると、未割当て、として１本化されるので、Apply。
   本当に？というダイアログが出るので、Apply。完了のダイアログで、X close
7. 未割当て　3.73 GIB と出るので、このラインをセレクト、
   パーティション　->　New
   ext2 とある部分を、fat32 に変更、Add ボタン　をプッシュ。
8. 新規パーティション　fat32 3.73　GIB とある行をセレクト、Applyを押す。
   本当？ダイアログをApply、完了をcloseする。
9. /dev/sdc1 fat32 3.73GIB 7.47MIB 3.72GIBの行をセレクト。
   パーティション　　->　フラグを編集(A)　をセレクト。
   bootにチェックマーク、closeを押す。
10. ウインドウを閉じ、Exitボタンをダブルクリック。
    ダイアログのRebootをチェック、OKを押す。
11. CDを取り出し、トレイを閉じ、リタン。
12. アンマウントしてから、抜いておきます。

## ステージ３　**USB**ペンに、ポーティアスをインストール。


1. ポーティアス　ライブCDをトレイに入れ、立ち上げます。
2. ポーティアスの画面が出ましたら、リタン、又は、何もしないでいると、始まります。
3. 立ち上がったら、K Menu ->　System　Porteus-2-usb をセレクト。
4. ターミナルが登場、 Press enter to continue を見たら、リタン。
5. Please plug your stick in and press enter when ready で、
   先ほどフォーマットしたUSBペンを、スロットに差し、リタン。
6. Press enter to continue を見たら、リタン。
7. Please provide size in megabytes for fat32 boot partition
   example : 64M　とありますが、fat32の部分にデータも保管したいので、
   512Mとしました。最低は32Mからとのこと。ここでは、512Mと入れ、リタン。
8. Please choose linux file system for the second partition
   currently supported are: ext2、ext3、ext4、xfs...
   example : xfs
   とあるので、xfs　とタイプ、リタン。
   Warning が出るので、y をタイプ、リタン。
9. Please remove your USB device ...
   press enter when done で、一旦ペンドライブを引き抜き、改めて差し込み、リタン。
10. Welcome to Porteus boot installer  press any key to continue、or Ctrl+C to abort で、
    a をタイプして、リタン。
11. Your USB device should now bootable...
    reboot now? Answer y/n
    インテルマックでは、ブート用のCDが必要なので、n を入れてリタン。
12. K Menu -> Log Out -> Restart Computer
13. デスクトップに見えるUSBペンは、Untitleとあるので、すべて大文字で、
    USB_PENのような名前をつけておきます。

## ステージ４　ポーティアスブート専用**CD**を作る。


1. Porteus live CD をダブルクリックして中身を見ると、
   boot と　porteus というフォルダーが見えます。
   これを、myporteusのようなフォルダーを作り、そこにコピーします。
2. porteusの方を開き、baseを開くと、そこには、全部で９ケのファイルがあります。
   これを全て削除します。
   baseは、これで空になりました。
   porteus-v09.sgn を削除します。
   これで、myporteusフォルダーは、約5.8MBになりました。
   このmyporteusフォルダーを、USBペンの中にコピーします。
   これで、USBペンの中身を見ると、bootとmyporteusを見ることができます。
3. Porteus live CD を立ち上げます。
4. Homeをダブルクリック、Homeを開いておきます。
   Systemをダブルクリック、
   System -> Storage Media -> USB_PEN -> data -> myporteus
   に到達したら、これを/rootにコピーします。
5. Konsoleを立ち上げ、/root/myporteus/porteusフォルダーをコンソールにドラッグ。
   ダイアログからcdをセレクト。
   lsすると、make_iso.sh　というファイルがあるのが、見えます。
   # ./make_iso.sh　/tmp/porteus_boot.iso
   として、テンプにアイソファイルを作成します。
   尚、アンダーラインは、イコールのキー位置をタイプすると出てくるでしょう。
   そこが、英語キーボードでのアンダーラインのキー位置になります。
6. Homeから、左上にある上向きアローキーにて、1段階上がり、/tmpに行くと、
   porteus_boot.isoが出来ているのが見えます。
   System -> Storage Media -> USB_PEN に行き、アイソファイルをmoveします。
7. ログアウトから、Restart Computerをセレクト。再起動します。
8. USB_PEN から、porteus_boot.isoを取り出し、CDに焼きます。
   ライブCD上にて、CDを焼くことも可能です。

## ステージ５　**USB**ペンをブート専用**CD**経由で立ち上げ。


通常、USBから立ち上がるように、マシンの設定を変更しても、
次回は、ちゃんと元に戻っていることになっているので、
自分以外のパソコンをちょっとだけ借りる場合でも、迷惑にならないのですが、
持ち主は、何かゴソゴソやっていると、心配するかもしれません。
その点、CD経由は、ディフォルトでそのように設定されていることが多いので、
わりと自然にブートすることができ、クールです。

1. ポーティアスブートCDをトレイに入れ、ブートします。
   作業はUSBペンに引き継がれポーティアスは立ち上がってきます。
2. K Menu から、コントロールセンターをセレクト、
   System Administration -> Date & Time に行きます。
   ディフォルトでは、UTCとなっていますので、これより８つ上に送り、
   Tokyoをセレクト。Applyを押すと、時計は自動的にJST表示に変更され、
   次回以降も、JSTが表示されます。
3. Regional & Language を、Defaultから、Asia, East -> Japan をセレクト。
4. Keyboard Layout をセレクト。
   まず、右サイドにあるリストから、USA us を残し、それ以外は、<< Remove します。
   Brazil、Czechia、France　などを、左サイドに移動させて下さい。
   今度は左サイドのリストをナビゲイト、Japan をセレクト、Add >> にて、
   右サイドに移動させます。
   いま移動した、Japan jp をセレクト、Layout variant のポップアップを開き、
   106をセレクトしておきます。これで、Applyボタンを押します。
5. Appearance & Themes にて、フォントのサイズ、Desktop -> Panels にて、
   パネルの様子などを、お好みで、設定なさって下さい。

# ステージ６　日本語の入力モジュール作成の準備。


日本語の入力を受け付けるには、scim、uim などのプログラムを利用します。
スラックスのサイトには、scimを利用したモジュールがありますが、
うまく動いてくれません。
同様に、japaneseと言う名前のモジュールも働いてくれません。
そこで、できるだけシンプルな形での、作り直しを行います。
scim、uim を較べると、uim の方が楽ですので、uimを使うことにします。
かなを漢字にしたりするのに、anthyも必要です。
作業の流れは、以下のようになります。

1. uim、anthy、および、依存ファイルをダウンロード。
2. すべてのファイルを、*.sq4.lzm ファイルに変換しておく。
3. 用意したlzmファイルをすべて、porteus/modules フォルダーにいれる。
4. コンフィギュア、つまり、設定のことですが、
   uimは、各国の言葉を扱うので、今回は、日本語を扱う、という設定をします。
   uimをいつどんな時に立ち上げるのか、これを、.xsessionファイルに書き込みます。
   # コンソールから、uim-pref-gtk にて、m17n.anthy が登場する設定を行う。
   そして、再起動をする。
5. bluefishを立ち上げ、シフト＋スペース　にて、かな入力を試みる。
6. 動作が確認できたら、myja のようなフォルダーを作り、
   今回作成した、lzmファイルをすべて、lzm2dir を使って、myja以下に展開しておきます。
   全部展開したら、今度は、dir2lzm にて、１ケのモジュールにまとめます。
   dir2lzm myja myja-0.0.1.sq4.lzm で、マイジャモジュールの完成です。
7. 個別のlzmファイルを、porteus/modules から削除、myja-0.0.1.sq4.lzmをいれる。
8. 動作テスト。

## ステージ７　日本語の入力モジュールを作成する。

uimのマニュアルを見る場合は、
http://anthy.sourceforge.jp/cgi-bin/hikija/hiki.cgi?InstallUim#l1

そこに、uim　が必要とするものがリストされています。
ポーティアスにすでに入っているものは、５本で、残りを調達します。
1. Autoconf（>= 2.60b）インストール済み。
2. Automake（>= 1.10）インストール済み。
3. Libtool （>= 1.5.22）インストール済み。
4. intltool（>= 0.35.2）
5. GNU make インストール済み。
6. Perl インストール済み(5.10.1)。
7. Ruby
8. librsvg
9. AsciiDoc
10. ed

ところで、出来上がった日本語入力は、今の所、KWriteでは、動作しません。
そこで、動作するエディターを含め、以下のファイルを入手します。
そして、これらを、どこから取ってくるか検討します。

1. スカッシュ４のメインレポより、以下の６本を入手します。
サーチに、geany とタイプ、お目当てのファイルが出てきます。
http://code.google.com/p/fidoslax/downloads/list
マイナス符号２ケ使いで、sq4だと知らせてくれます。
geany--0.19--0.1.3.lzm
librsvg--2.32.1--0.1.3.lzm
asciidoc--8.4.5--0.1.3.lzm
python--2.6.5--0.1.3.lzm
tcl--8.5.7--0.1.3.lzm
tk--8.5.7--0.1.3.lzm

スラックスのモジュールサイトからは。

2. スラックスのモジュールサイトからは、以下の１本を入手。
   取ってきたらカーソルをあてて、右クリックして、
   Convert Slax sq3 module(s) to sq4...をセレクトします。
   すると、名前にも、sq4の文字が追加されます。
   http://www.slax.org/modules.php?action=detail&id=1779　にて、
   anthy-9100e-1.lzm　を入手。
3. スラックウェアのパッケージ
   http://packages.slackverse.org/　より、
   以下の２本を入手します。
   m17n-lib-1.5.4-i486-1.txz を右クリック、Convert tgz/txz to sq4...をセレクト、
   m17n-lib-1.5.4-i486-1.sq4.lzm　に、変換する。
   slackwareの13.0で、rubyをサーチ、
   少し古めの、ruby-1.8.7_p174-i486-1.txz　を入手、
   右クリック、Convert tgz/txz to sq4...をセレクト。
4. デビアンのパッケージ、
   http://www.debian.org/distrib/packages　にて、intltoolでサーチ、
   intltool_0.41.1-1_all.deb　を入手、右クリック、Convert deb to porteus。
5, http://sourceforge.jp/projects/sap/releases/　にて、
   スラックウェアのパッケージを入手。
   uim-1.5.4-i486-2.tgz　を右クリック、Convert tgz/txz to sq4...をセレクト、
   uim-1.5.4-i486-2.sq4.lzm　に、変換する。
6. http://sourceforge.jp/projects/slackware/downloads/49030/ipa-fonts-ttf-00203-noarch-2.tgz/?use_default=none
   ipaフォントのスラックウェアのパッケージを入手。
   ipa-fonts-ttf-00203-noarch-2.tgz　を右クリック、Convert tgz/txz to sq4...をセレクト、
   ipa-fonts-ttf-00203-noarch-2.sq4.lzm に、変換。

## ステージ８　**Ruby1.8.7**のソースを、**lzm**に変換する。


ソースからlzmモジュールを作ります。今回は、ルビー1.8のターファイルをlzmに変換します。
この時、ルビーのライブラリーも作成しておきます。

1.　http://www.ruby-lang.org/ja/downloads/　にて、ruby-1.8.7-p330.tar.gzを入手、
　　rb187_330のようなフォルダーを作り、そこに入れておきます。

2. 以下のスクリプトを rb2lzm.sh と言う名前にて、ターファイルと一緒に置きます。

```
#!/bin/bash
### rb2lzm.sh
mkdir myrb187
mkdir -p mylib187/usr/local/lib
tar xvzf ruby-1.8.7-p330.tar.gz
cd ruby-1.8.7-p330
./configure --enable-shared
make
make install DESTDIR=../myrb187
cd ../
mv myrb187/usr/local/lib/libruby-static.a  mylib187/usr/local/lib
mv myrb187/usr/local/lib/libruby.so  mylib187/usr/local/lib
mv myrb187/usr/local/lib/libruby.so.1.8  mylib187/usr/local/lib
mv myrb187/usr/local/lib/libruby.so.1.8.7  mylib187/usr/local/lib
dir2lzm mylib187 libruby-1.8.7-p330.sq4.lzm
chmod 777 libruby-1.8.7-p330.sq4.lzm
dir2lzm myrb187 ruby-1.8.7-p330.sq4.lzm
chmod 777 ruby-1.8.7-p330.sq4.lzm
```

3. rb187_330フォルダーに移動、以下の操作を行います。

```
# cd rb187_330　　リターン。
# chmod 755 rb2lzm.sh　リターン。
# ./rb2lzm.sh　　リターン。
```

これで、libruby-1.8.7-p330.sq4.lzm と、ruby-1.8.7-p330.sq4.lzm が作成されました。

## **lzm**モジュールが揃ったか、確認をします。

　日本語入力モジュール関係で、必要とされるファイルを確認します。
1. anthy-9100e-1.sq4.lzm
2. asciidoc--8.4.5--0.1.3.lzm
3. intltool_0.41.1-1_all.deb
4. ipa-fonts-ttf-00203-noarch-2.sq4.lzm
5. librsvg--2.32.1--0.1.3.lzm
6. m17n-lib-1.5.4-i486-1.sq4.lzm
7. python--2.6.5--0.1.3.lzm
8. ruby-1.8.7_p174-i486-1.sq4.lzm
9. tcl--8.5.7--0.1.3.lzm
10. tk--8.5.7--0.1.3.lzm
11. uim-1.5.4-i486-2.sq4.lzm

以上の11本は、myjaというモジュールとして１本化されます。

これ以外では、
http://www.slax.org/modules.php?action=detail&id=1913 にて、
sazanami-fonts-ttf-20040629-1.lzm を入手、右クリックにて、
Convert Slax sq3 module(s) to sq4... で変換します。
1. sazanami-fonts-ttf-20040629-1.sq4.lzm

http://www.slax.org/modules.php?action=detail&id=4159にて、
bluefish-1.0.7-i486-2sl.lzm　を入手、右クリックにて、
Convert Slax sq3 module(s) to sq4...　をセレクト、sq4に変換します。
2. bluefish-2.0.3.lzm

3. 先ほど入手した、geany--0.19--0.1.3.lzm。

これらすべてのlzmファイル１４本を、ダブルクリックしていきます。
すると、porteus/modules に取り込まれます。
ですから、念のため、どこかにあらかじめコピーを保存しておいたほうがいいかもしれません。

## ステージ９　**uim**設定の詳細。


1. コンソールを立ち上げ、# nano　.xsession　にて、
　　不可視ファイル .xsession を作成。
　　そこに、以下を、コピーしておきます。
　　ファイル名の先頭に、ドットをつけておくと、
　　そのファイルは、透明人間になるというわけです。

```
export XMODIFIERS=@im=uim
export GTK_IM_MODULE=uim
export UIM_IM_ENGINE=anthy
if type uim-xim & > /dev/null ; then
uim-xim &
fi
```

# Ctrl+O で、セーブ。
# リタン。
# Ctrl+X で、脱出。
もし、ナノの 使い方が分からない場合は、
ミニマルナノを 参照してください。
これで、コンソールを終了します。
# exit　リタン。

**uim**のグローバル設定を行う。


2. 再度コンソール をたちあげます。
# uim-pref-gtk で、リタン。
すると、最初に、
libmimx-ispell.so: cannot open shared object file:
No such file or directory
が出ますが、続いて、
Global settings というリストが出ます。

先頭 の Specify default IM に、チェックマークをつけます。
３行目の、Enable input methods のお尻にある、Edit... ボタンを押します。
すると、Enable　Disable と言う2つのリストがある、
Enable input methods　が、登場します。
ここで、Anthy, m17n-en-ispell, m17n-ja-anthy の3つを残し、
後は、すべて、右側のDisabledの方に移動させます。
そして、m17n-ja-anthyを、リストのもっとも下に置きます。
これで、close を押します。
Global settingsの2行め、
Default input method のポップアップを開き、
m17n-ja-anthy を選択します。
これにて、 Apply, OK とします。
これで、 設定は、完了しました。
再起動し、bluefish、geany、leafpad などを立ち上げ、
シフト＋スペースバーを同時に押してから、
キーボードでタイプすると、ひらがな、が出ます。
後は、スペースバーで、かたかな、漢字変換ができます。
英字に戻す場合は、シフト＋スペースバーを同時に押します。
右下のパネルのusをクリックしてJpに変えておくと、日本語キーボード対応となります。

## ステージ１０　日本語入力関係のモジュールを１本化する。


以下のファイルを統合して、myjaモジュールを作成します。

1. anthy-9100e-1.sq4.lzm
2.  asciidoc--8.4.5--0.1.3.lzm
3. intltool_0.41.1-1_all.sq4.lzm
4. ipa-fonts-ttf-00203-noarch-2.sq4.lzm
5. librsvg--2.32.1--0.1.3.lzm
6. m17n-lib-1.5.4-i486-1.sq4.lzm
7. python--2.6.5--0.1.3.lzm
8. ruby-1.8.7_p174-i486-1.sq4.lzm
9. tcl--8.5.7--0.1.3.lzm
10 tk--8.5.7--0.1.3.lzm
11. uim-1.5.4-i486-2.sq4.lzm

/root 以下に、myja のようなフォルダーを作り、上記lzmファイルを入れておきます。
そこに、myja1 という名前でフォルダーをつくり、
その中に、lzmモジュールを展開していきます。

```
# lzm2dir  asciidoc--8.4.5--0.1.3.lzm myja1
# lzm2dir intltool_0.41.1-1_all.sq4.lzm myja1
# lzm2dir ipa-fonts-ttf-00203-noarch-2.sq4.lzm myja1
# lzm2dir librsvg--2.32.1--0.1.3.lzm myja1
# lzm2dir m17n-lib-1.5.4-i486-1.sq4.lzm myja1
# lzm2dir python--2.6.5--0.1.3.lzm myja1
# lzm2dir ruby-1.8.7_p174-i486-1.sq4.lzm myja1
# lzm2dir tcl--8.5.7--0.1.3.lzm myja1
# lzm2dir tk--8.5.7--0.1.3.lzm myja1
# lzm2dir anthy-9100e-1.sq4.lzm myja1
# lzm2dir uim-1.5.4-i486-2.sq4.lzm myja1
```

そして、１本化します。

```
# dir2lzm myja1 myja--0.0.1.lzm
```

slax/modules に存在する、上記１１本のファイルを、Package Manager で、removeし、
かわりに、今作った　myja--0.0.1.lzm　をダブルクリックして投入します。
上記の操作は、テキストに書いた方が良いかもしれません。

## ステージ１１　**PyGTK**をインストールする。


PyGTKは、GTK+のウインドウ、ボタンなどを、パイソンから利用できるプログラムです。

1. http://packages.slackverse.org/　にて、pygtk　をサーチに入れると、
　pygtk-2.16.0-i486-1.txz　が出てきます、これをダウンロード、右クリックにて、lzmファイルに変換。
2. pygobject　でサーチ、 pygobject-2.20.0-i486-1.txz　を入手、右クリックにて、lzmファイルに変換。
3. pycairo　でサーチ、 pycairo-1.8.8-i486-1.txz　を入手、右クリックにて、lzmファイルに変換。

これで、3本とも、lzmファイルになったので、
3本ともダブルクリックを行い、インストールを完了します。

パイソンのハローを用意する。

---


以下のプログラムを、hello.pyと名前をつけて、
pygtk_tutのようなフォルダーにコピーしておきます。

```
#!/usr/bin/python
### hello.py
import pygtk
pygtk.require('2.0')
import gtk

class Base:
    def __init__(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.show()
        self.window.connect("destroy", self.destroy)

    def destroy(self, widget, data=None):
        gtk.main_quit()

    def main(self):
        gtk.main()

if __name__ == "__main__":
    base = Base()
    base.main()
```

コンソールで、pygtk_tutフォルダーに行きます。
# cd pygtk_tut　リターン。
# chmod 755 hello.py で、リターン。
続いて、
# ./hello.py で、リターン。
空のウインドウが出現します。

## ステージ１２　**FXRuby**をインストールする。


fxrubyは、C++で実装されたfox-tkを、ルビーから利用できる、
技術的にも非常にしっかりしたGUIプログラム。
rubygemsは、ルビーのファイル管理プログラムですが、
これををlzmパッケージにする場合、例によって、人力になってしまいます。
そこで、rubygemsは、とりあえず、モジュールにしないで、
普通のLinuxのように、ただ、そこに置いておくことにします。
最初に、フォックスを入手します。

1. http://code.google.com/p/fidoslax/downloads/list にて、
   foxで、サーチ。fox--1.6.37--0.1.3.lzmを入手。
   ダブルクリックで、インストール。
2. foxの動作テスト用に、hello.cppを用意します。

```
// hello.cpp
#include "fx.h"
#include "FXExpression.h"
int main(int argc, char *argv[]) {
   FXApp application("Hello","FoxTest");
   application.init(argc,argv);
   FXMainWindow *main = new FXMainWindow(&application, "Hello", NULL, NULL,
DECOR_ALL);
   new FXButton(main, "&Hello, World!",
   NULL, &application, FXApp::ID_QUIT);
   application.create();
   main->show(PLACEMENT_SCREEN);
   return application.run();
}
```

メイクファイルは。


メイクファイルは以下のようにします。

```
#
# Makefile for fox
#
CC=g++
CFLAGS=-Wall
PROGRAM=hello
FILES=hello.cpp
INCLUDES=-I/usr/include/fox-1.6

$(PROGRAM):$(FILES)
	$(CC) $(CFLAGS) $(INCLUDES) `fox-config --libs` -o $(PROGRAM) $(FILES)

clean:
	rm -f *.o $(PROGRAM)
.PHONY: clean
```

ファイル名をMakefileとし、fox_tutに、hello.cppと一緒に置きます。
コンソールから、
# cd fox_tut　リターン。
# make　リターン。
# ./hello　リターン。
スクリーンのセンターに、かわいいのが出現します。
ここで、ご注意。
$(CC) $(CFLAGS) $(INCLUDES) `fox-config --libs` -o $(PROGRAM) $(FILES)　の行、
rm -f *.o $(PROGRAM)　の行、の先頭には、タブを使う必要があります。
これは、そのような仕様になっているので、スペースではなく、タブを使って下さい。

## **rubygems**のインストール。


マイジャを作成しなかった場合、ルビー本体が入っていないことになるので、
ステージ８で作ったruby-1.8.7を、インストールしておきます。

http://rubyforge.org/projects/rubygems/ にて、rubygems-1.3.7.tgz を入手。
gems137のようなフォルダーを作り、そこに入れておきます。
コンソールから、
# tar xvzf rubygems-1.3.7.tgz リターン。
# cd rubygems-1.3.7 リターン。
# ruby setup.rb
これで、/usr/binにある、gemという小さなファイルを、操作できるようになります。

gemからfxrubyをインストールします。
# gem install fxruby
Successfully installed fxruby-1.6.20-x86-linux
1 gem installed
Installing ri documentation for fxruby-1.6.20-x86-linux...
Installing RDoc documentation for fxruby-1.6.20-x86-linux...

シンボリックリンクを 1 つ追加します。
# cd /usr/lib
# ln -s libfiff.so.3.9.4 libtiff.so.4

ルビーのハローを用意する。

---


fxruby_tutのようなフォルダーに、以下のプログラムをいれておきます。

```
#!/usr/bin/ruby
### hello.rb
require 'rubygems'
require 'fox16'
include Fox

theApp = FXApp.new
theMainWindow = FXMainWindow.new(theApp, "Hello")
button = FXButton.new(theMainWindow, "Hello, World!")
button.tipText = "Push Me!"
button.connect(SEL_COMMAND) { exit }
FXToolTip.new(theApp)

theApp.create
theMainWindow.show
theApp.run
```

コンソールから、
# chmod 755 hello.rb リターン。
# ./hello.rb リターン。
foxの時と同じウイジェットが出現します。

## ステージ１３　**RubyGems**自身を**lzm**に変換する。


パッケージ管理のしくみは、通常、ディストロが最初から組み込んでいるものを使います。
パッケージには、*.deb　*.rpm のようなサフィックスがついているので、それと分かります。
しかし、これは、基本的に、Cのプログラムを想定したサービスです。
なので、ルビーは、自前のパッケージ管理のしくみを持っています。
その名前がルビージェムスで、管理されるパッケージの方には、しっぽに、*.gemをつけます。
本来、ルビーのスクリプトは、その辺に置いておきますが、
自分以外のまともな方が作ったプログラムは、説明書、宣伝文などもつけたパックにして、
所定の場所で、管理いたします、というのです。
わたしたちは、ああそうですか、とは言わず、lzmモジュールに変換したい、と思うわけです。
尚、ルビー本体は、いくつも入れないで、ここでは、ルビー1.8.7-p302１本にします。
そして、ruby1.9は、別のペンドライブを、1.9専用として用意すること、おすすめです。
1.9しかいない経験、この壮快さは一体何なのでしょうか、実にクールです。
1. マイジャを入れてない場合、ステージ8で作った、ルビー1.8.7-p302を
　インストールしておきます。
2. すでにfxRubyなどを、ルビージェムス経由で、利用している場合、
　gem uninstall fxruby のように、削除しておきます。
3. http://rubyforge.org/frs/?group_id=126 より、
　rubygems-1.3.7.tgzを入手、rgems_packという名前のフォルダーに入れておきます。
4. 次ページにあるテキストをコピーし、dogems.sh という名前で、rgems_packフォルダーに収納
。
5. すでに、ルビージェムスをインストールしている場合は、
　３行目と４行目の先頭に#をつけておきます。

```
#tar　xvzf rubygems-1.3.7.tgz
#ruby　rubygems-1.3.7/setup.rb
```

ルビージェムスを**lzm**に変換するスクリプトは。


```
#!/bin/bash
### dogems.sh
tar       xvzf  rubygems-1.3.7.tgz
ruby     rubygems-1.3.7/setup.rb
mkdir   -p  madir/usr/local/lib/ruby/site_ruby/1.8
mkdir   madir/usr/local/bin
cp       /usr/local/bin/gem  /root/rgems_pack/madir/usr/local/bin
cp       -r  /usr/local/lib/ruby/site_ruby/1.8    \
         /root/rgems_pack/madir/usr/local/lib/ruby/site_ruby
dir2lzm  madir  rubygems--1.3.7.lzm
chmod   777  rubygems--1.3.7.lzm
rm    /usr/local/bin/gem
rm    -r  /usr/local/lib/ruby/site_ruby/1.8/rbconfig
rm    -r  /usr/local/lib/ruby/site_ruby/1.8/rubygems
rm    /usr/local/lib/ruby/site_ruby/1.8/gauntlet_rubygems.rb
rm    /usr/local/lib/ruby/site_ruby/1.8/rubygems.rb
rm    /usr/local/lib/ruby/site_ruby/1.8/ubygems.rb
```

6. コンソールから、
# cd rgems_pack リタン。
# chmod 755 dogems.sh リタン。
# ./dogems.sh リタン。
7. この時点で、コンソールから、
# gem --version リタン。すると、bashは、gemを知りません、などと云います。
8. rubygems--1.3.7.lzm　が出来上がっているので、これのコピーをインストールします。
　　ここで、コンソールから、
　　# gem --versiom リタン。1.3.7 と表示されます。
以上で、ルビージェムスをlzmに変換する作業は、終了です。

# ステージ１４　ルビーの**gem**ファイルを**lzm**に変換する。


ルビー本体をソースからlzmにする過程を経験して、lzmファイルの元ネタは、
ファイルのハイエラキーをそのまま再現したものに、
お目当てのデータのみを入れたフォルダーだ、とわかります。
このフォルダーに対し、dir2lzm というスクリプトをあてると、
*.lzmファイルが出来上がります。
練習のため、プログレスバーをインストールします。
今、gem install progressbar にて、
progressbar-0.9.0.gem をインストールすると、
ジェムファイルの管理屋さん、ルビージェムスは、
/usr/local/lib/ruby以下に、gemsというフォルダーを作り、
ここから、ファイルを構築して行きます。
これを、lzmにするには、
1. /usr 以下、フォルダーを順に並べたものを用意して、
2. プログレスバー関連のファイルを、ここそこにコピーする。
3. dir2lzmで処理。
今回は、ルビーのメソッドを１つ用意しました。

Coming Up

ジェムファイルを**lzm**に変換するメソッドは。

```
#!/usr/local/bin/ruby
### gem2lzm.rb
require 'fileutils'
def to_dir(na, lzna)
   p  gname    = na+".gem"
   p  specname = na+".gemspec"
   gemdir      = "/usr/local/lib/ruby/gems/1.8/gems"
   cachedir    = "/usr/local/lib/ruby/gems/1.8/cache"
   docdir      = "/usr/local/lib/ruby/gems/1.8/doc"
   specdir     = "/usr/local/lib/ruby/gems/1.8/specifications"
   p cache_path = cachedir + '/' + gname
   p doc_path   = docdir + '/' + na
   p gem_path   = gemdir + '/' + na
   p spec_path  = specdir + '/' + specname
   p mycache    = '/tmp/mydir/usr/local/lib/ruby/gems/1.8/cache'
   p mydoc     = '/tmp/mydir/usr/local/lib/ruby/gems/1.8/doc'
   p mygems     = '/tmp/mydir/usr/local/lib/ruby/gems/1.8/gems'
   p myspec     = '/tmp/mydir/usr/local/lib/ruby/gems/1.8/specifications'
   FileUtils.mkdir_p('/tmp/mydir/usr/local/lib/ruby/gems/1.8/')
   FileUtils.mkdir(mycache)
   FileUtils.mkdir(mydoc)
   FileUtils.mkdir(mygems)
   FileUtils.mkdir(myspec)
   FileUtils.cp(cache_path, mycache)
   FileUtils.cp_r(doc_path, mydoc)
   FileUtils.cp_r(gem_path, mygems)
   FileUtils.cp(spec_path, myspec)
   system("/usr/bin/dir2lzm /tmp/mydir /tmp/#{lzna}")
   system("chmod 777 /tmp/#{lzna}")
   system("rm -r /tmp/mydir")
end
to_dir(ARGV[0], ARGV[1])
```

## **gem2lzm**という名前でコピー。

前のページのテキストをコピーしたら、gem2lzm.rbという名前で、
gem2lzm_packのようなフォルダーに入れておきます。
gem2lzmを使うときには、ジェムファイルの名前を間違えないようにします。
ヘルパーを 1 ヶ作っておきましょう。

```
#!/usr/local/bin/ruby
### gemlist.rb
Dir::foreach('/usr/local/lib/ruby/gems/1.8/gems') {|f|
  puts "#{f}"
}
```

gem2lzmを使ってみます。

1. #  chmod 755 gemlist.rb  リタン。
2. #  ./gemlist.rb  リタン。
3. プログレスバーなら、progressbar-0.9.0　とはっきりしたので、
   これで、gem2lzm.rbを呼び出します。
4. # chmod 755 gem2lzm.rb  リタン。
5. # ./gem2lzm.rb  progressbar-0.9.0  progressbar--0.9.0.lzm  リタン。
6. /tmp に、progressbar--0.9.0.lzm　が出来ています。
7. # gem uninstall progressbar  リタン。
8. # gem q --l リタン。
9. モジュールマネジャーにて、progressbar--0.9.0.lzmをインストール。
10. # gem q --l リタン。
11. プログレスバーが実際どんなものなのか、マニュアルの例を実行してみます。

```
#!/usr/local/bin/ruby
### pbar_app.rb
require 'rubygems'
require 'progressbar'
pbar=ProgressBar.new("testing",200)
200.times{sleep(0.1); pbar.inc}
pbar.finish
```

以上で、ルビーのジェムファイルをlzmに変換する作業は終了です。

## ステージ１５　**Shoes**を入れる。


Shoesは、ルビーの書式で、ボタンなどをつけたフレームを作り出す、GUIプログラム。
かんたん、軽い、親切、覚えやすい、センス良い。
５拍子揃った最高のプログラムなので、リミックスに入れます。
先に、依存モジュールを揃えておきます。
全部で７本になりました。
デビアンのサイト、
http://www.debian.org/distrib/packages にて、
サーチに、libgnutls26 と入れれば、１が出ます。
同様にして、以下の７本を入手します。
1. libgnutls26_2.4.2-6+lenny2_i386.deb
2. libkrb53_1.6.dfsg.4~beta1-5lenny6_i386.deb
3. libkeyutils1_1.2-9_i386.deb
4. libldap-2.4-2_2.4.23-7_i386.deb
5. libldap2-dev_2.4.23-7_i386.deb
6. libsasl2-2_2.1.22.dfsg1-23+lenny1_i386.deb
7. libtasn1-3_1.4-1_i386.deb

これらを、myshoesのようなフォルダーを作って、そこに収納、
右クリックにて、Convert deb to sq4 をセレクト、
sq4モジュールに変換、chmod 777 も行っておきます。
そして、次のように、pack.shと言う名前のシェルテキストにします。

依存ファイルのシェルテキストは。


```
#!/bin/bash
### pack.sh
mkdir      pack
chmod 777 libgnutls26--2.4.2-6lenny2_i386.lzm
chmod 777 libkeyutils1_1.2-9_i386.sq4.lzm
chmod 777 libkrb53--1.6.dfsg.4beta1-5lenny6_i386.lzm
chmod 777 libldap--2.4-2_2.4.23-7_i386.lzm
chmod 777 libldap2-dev--2.4.23-7_i386.lzm
chmod 777 libsasl2-2--2.1.22.dfsg1-23lenny1_i386.lzm
chmod 777 libtasn1-3--1.4-1_i386.lzm
lzm2dir libgnutls26--2.4.2-6lenny2_i386.lzm    pack
lzm2dir libkeyutils1_1.2-9_i386.sq4.lzm    pack
lzm2dir libkrb53--1.6.dfsg.4beta1-5lenny6_i386.lzm    pack
lzm2dir libldap--2.4-2_2.4.23-7_i386.lzm    pack
lzm2dir libldap2-dev--2.4.23-7_i386.lzm    pack
lzm2dir libsasl2-2--2.1.22.dfsg1-23lenny1_i386.lzm    pack
lzm2dir libtasn1-3--1.4-1_i386.lzm    pack
dir2lzm pack shoes_deps--0.0.1.lzm
```

このファイルを、myshoesフォルダーに収納、コンソールから、
chmod 755 pack.sh　リターン。
./pack.sh　リターン。
出来上がったshoes_deps--0.0.1.lzmを、ダブルクリック、インストールしておきます。
続いてシューズ本体を入手します。

# シューズの**run**ファイルを入手する。


http://shoes.heroku.com/downloads
shoes 3 novideo をクリック、shoes3.run.shを入手します。
さらに、ページの1番下にいるペンギンをクリックすると、
スクリプトが出ますので、 メニューのFile -> Save Page As にて
このファイルを取り込みます。
これは、shoes2.runファイルで、メイクセルフというファイル形式になっています。
この2本を、shoes_runのようなフォルダーにいれ、
# cd shoes_run　　リターン。
# chmod 755 shoes2.run　リターン。
# ./shoes2.run　　リターン。
これで、Shoesプログラムが、登場します。

この時、例えば、このようなものが、出るかもしれません。
/tmp/selfgz1464932377/shoes-bin: /usr/lib/libsasl2.so.2:
no version information available
(required by /usr/lib/libldap_r-2.4.so.2)

これは、libldap_r-2.4.so.2が期待するlibsasl2.so.2の、
サブバージョンと合っていないと云っています。
通常は、そのままでも動作しますので、問題ありません。
もし、上記とまったく同じ場合なら、
1. /usr/libに行く。
2. libsasl2.so.2.0.22があるのを確かめる。
3. libsasl2.so.2シンボリックリンクを削除。
4. # ln -s libsasl2.so.2.0.22 libsasl2.so.2 リターン。
普通はバージョンを上げるとうまくいくはずですが、今回のは、逆です。
SHOESの画面のRead the Manualをクリック、
Run this をクリックすると、プログラム例が登場します。
shoes3.run.shも同じように動くと思います。お試し下さい。

# ステージ１６　**HLA**高級アセンブリーを入れる。


アセンブリー言語で、ハローワールド以上のプログラムを作るのは、大変です。
データ構造がないので、どう作っていくというイメージ、構想が出にくいのです。
そこで、HLAの登場です。
HLAは、パスカル、Cのような高速タイプの言語と、
80x86アセンブリーが合体したような形になっています。
パスカルは、Cのように、何が何でもスタック、のような変則型でもなく、
変数、レコードなどは、使う前に、プログラムの先頭付近で、宣言しておく、
という伝統のスタイルが、アセンブリー言語と良くマッチします。
アセンブリー言語は、レジスターを直接操作することを主体にして、
プログラムを構成するので、いつものプログラミングとは、
相当違った雰囲気を味わうことになり、大いに楽しめます。

サイトから、Linux用のファイルを入手します。
http://webster.cs.ucr.edu/AsmTools/HLA/frozen.html
http://webster.cs.ucr.edu/AsmTools/HLA/dnld.html
Linux Users: までページを下がり、HLA Executables for Linux をクリック。
linux.hla.tar.gzを入手、バージョンは、2.106 でした。
myhlaというフォルダーを作り、その中にhla2のようなフォルダーを作り、
linux.hla.tar.gzを入れておきます。コンソールで、hla2に移動。
# cd /root/myhla/hla2　リターン。
# gzip -d linux.hla.tar.gz リターン。
# tar xvf linux.hla.tar リターン。
これで、usrからの階層フォルダーができますので、
myusrのようなフォルダーを作り、usrを入れます。そして、dir2lzmを行います。
# dir2lzm myusr hla--2.106.lzm
# chmod 777 hla--2.106.lzm
これで、hla--2.106.lzmが、できましたので、ダブルクリック、
モジュールフォルダーに入れておきます。

## **HLA**の環境変数を設定する。


HLAの環境変数を設定します。
ドットバッシュアールシーに書き込むことにします。
.bashrcを、ホームに不可視ファイルとして、作成します。
コンソールから、
# nano .bashrc　リターン。
ナノは、ファイルが存在しなければ、新たに作ります。
以下をコピーアンドペーストして下さい。

export PATH=/usr/hla:$PATH
export hlalib=/usr/hla/hlalib/hlalib.a
export hlainc=/usr/hla/include
export hlatemp=/tmp

これで、バッシュがhlaを認識するようになったか、調べます。
# hla -?　リターン。
no such file or directory と言ってきた場合は、
1. hla--2.106.lzm が、modulesフォルダーに入っていることを
　確認。
2. .bash_profile ファイルに次の 1 行を加えます。
. ~/.bashrc

**HLA**のハロープログラムを用意する。


ハロープログラムを用意します。

```
// hello.hla
program hello;
#include( "stdlib.hhf" )
begin hello;
    stdout.put( "Hello, world!", nl );
end hello;
```

1. コメントは、C、C＋＋スタイル。
2. プログラムは、予約語、programから始まる。（パスカル風味）
3. スタンダードライブラリーのインクルード。（C風味）
4. ブロックの開始は、begin プログラム名 セミコロン。（独自風）
5. スタンダードライブラリーの呼び出し文は。（ C＋＋風味）
6. ブロックの終わりは、end プログラム名 セミコロン。（フォートラン、モジューラ２風）
7. main関数は、なくてOK。（パスカル風味）
8. しかし、最低１つのbegin...endブロックが必要。（パスカル風味）

C　パスカルを見たことがあれば、違和感なく、すぐに使えます。

hello.hlaでセーブされたこのファイルをコンパイルします。

# hla hello リターン。

実行します。

# ./hello リターン。

これで、何でアセンブリー言語なのか、と思うかもしれません。

そこで、次に、３と４を足し算するプログラムを見ます。

３と４の足し算をアセンブリー風にすると。


３と４を足し算するプログラムを見ます。

```
// add1.hla
program       add1;
#include("stdlib.hhf")
begin add1;
   mov(%0011, ax);
   add(%0100, ax);
   stdout.put("result: $", ax, nl);
end add1;
```

1. 80x86のインストラクションを使うときは、
   オプコード（データのある場所、ターゲット）; の形になる。
2. ビットを表すには、％をつける。
3. mov(%0011, ax); は、即値の３を、１６ビットレジスター、AXにコピー。
4. add(%0100, ax); は、即値の４を、AXに加算。
5. レジスターの中身を表示する時は、１６進数になる。
6. 16進数は、先頭に$マークをつける。

これを実行すると、

```
root@slax:~/hla_tut/add# hla add1
root@slax:~/hla_tut/add# ./add1
result: $0007
```

## **while**文を**HLA**のアセンブリースタイルで書くと。


while文をアセンブリースタイルで作ります。

```
// while1.hla
program while1;
#include("stdlib.hhf");
begin while1;
   mov(0, ecx);
   mov(0, eax);
   whileLp:
      cmp(ecx, 10);
      jnl whileDone;    // jump not less  if bigger than10  then jump
      inc(ecx);
      add(ecx, eax);
      jmp whileLp;
   whileDone:
      stdout.put("result: ", (type uns32 eax), nl);
end while1;
```

1. 32ビットレジスターECX、EAXをゼロで初期化。
2. whileLp, whileDoneというレーベルを作る。
3. クンプは、ECXの中身と１０を較べ、結果を示すフラッグを何本か立てます。
4. フラッグを調べ、１０より大きいが１なら、
5. whileDoneへジャンプ、EAXの中身を１０進数に変換して、プリント。
6. そうでない場合、ECXに 1 加算。ECXをEAXに加算。
7. whileLpに戻って、作業を続行します。

コンパイルして、実行します。

```
root@slax:~/hla_tut/while1# hla while1
root@slax:~/hla_tut/while1# ./while1
result: 55
```

プログラムの舞台裏で何が起っているのか、少しずつ見えてくるので、
HLAは、実際頼りになります。

# ステージ１７　インクスケープをインストール。


以前は、イラストレーターを使っていた方も、
今は、マックで、インクスケープを使っていたりします。
日本語は、あまり得意ではありませんが、全体、操作性はかなり向上しています。
プログラムは大きく、依存ファイルも多めですから、
インクスケープ専門のUSBペンを調達するのがおすすめです。
現時点でのバージョンは、0.48.0、アーチLinuxから、調達します。
その前に、
http://code.google.com/p/fidoslax/downloads/list にて、
pkgtar2lzm--0.0.2--0.1.3.lzm または、lzm-utils--0.5--0.1.3.lzm を入手、
モジュールフォルダーに、 加えておきます。
http://www.archlinux.org/packages にて、以下のパッケージを入手。
1. atkmm-2.22.1-1-i686.pkg.tar.xz
2. bzip2-1.0.6-1-i686.pkg.tar.xz
3. cairomm-1.8.6-1-i686.pkg.tar.xz
4. gc-7.1-1-i686.pkg.tar.gz
5. glibmm-2.24.2-1-i686.pkg.tar.xz
6. gsl-1.14-1-i686.pkg.tar.xz
7. gtkmm-2.22.0-1-i686.pkg.tar.xz
8. gtkspell-2.0.16-1-i686.pkg.tar.gz
9. hicolor-icon-theme-0.12-1-any.pkg.tar.xz
10. imagemagick-6.6.4.10-1-i686.pkg.tar.xz
11. libsigc++-2.2.8-1-i686.pkg.tar.xz
12. openjpeg-1.3-3-i686.pkg.tar.gz
13. pangomm-2.26.2-1-i686.pkg.tar.xz
14. poppler-0.14.5-1-i686.pkg.tar.xz
15. poppler-data-0.4.4-1-any.pkg.tar.xz
16. poppler-glib-0.14.5-1-i686.pkg.tar.xz
17. renderproto-0.11.1-1-any.pkg.tar.xz
18. inkscape-0.48.0-4-i686.pkg.tar.xz

以上１８本、**sq4.lzm** に変換します。

---


```
#!/bin/bash
### pk2lzm.sh
pkgtar2lzm  atkmm-2.22.1-1-i686.pkg.tar.xz  atkmm-2.22.1-1-i686.lzm
pkgtar2lzm  bzip2-1.0.6-1-i686.pkg.tar.xz  bzip2-1.0.6-1-i686.lzm
pkgtar2lzm  cairomm-1.8.6-1-i686.pkg.tar.xz   cairomm-1.8.6-1-i686.lzm
pkgtar2lzm  gc-7.1-1-i686.pkg.tar.gz   gc-7.1-1-i686.lzm
pkgtar2lzm  glibmm-2.24.2-1-i686.pkg.tar.xz  glibmm-2.24.2-1-i686.lzm
pkgtar2lzm  gsl-1.14-1-i686.pkg.tar.xz   gsl-1.14-1-i686.lzm
pkgtar2lzm  gtkmm-2.22.0-1-i686.pkg.tar.xz  gtkmm-2.22.0-1-i686.lzm
pkgtar2lzm  gtkspell-2.0.16-1-i686.pkg.tar.gz  gtkspell-2.0.16-1-i686.lzm
pkgtar2lzm  hicolor-icon-theme-0.12-1-any.pkg.tar.xz   hicolor-icon-theme-0.12-1-any.lzm
pkgtar2lzm  imagemagick-6.6.4.10-1-i686.pkg.tar.xz  imagemagick-6.6.4.10-1-i686.lzm
pkgtar2lzm  libsigc++-2.2.8-1-i686.pkg.tar.xz  libsigc++-2.2.8-1-i686.lzm
pkgtar2lzm  openjpeg-1.3-3-i686.pkg.tar.gz  openjpeg-1.3-3-i686.lzm
pkgtar2lzm  pangomm-2.26.2-1-i686.pkg.tar.xz  pangomm-2.26.2-1-i686.lzm
pkgtar2lzm  poppler-0.14.5-1-i686.pkg.tar.xz   poppler-0.14.5-1-i686.lzm
pkgtar2lzm  poppler-data-0.4.4-1-any.pkg.tar.xz  poppler-data-0.4.4-1-any.lzm
pkgtar2lzm  poppler-glib-0.14.5-1-i686.pkg.tar.xz  poppler-glib-0.14.5-1-i686.lzm
pkgtar2lzm  renderproto-0.11.1-1-any.pkg.tar.xz   renderproto-0.11.1-1-any.lzm
pkgtar2lzm  inkscape-0.48.0-4-i686.pkg.tar.xz  inkscape-0.48.0-4-i686.lzm
```

出来上がったlzmファイルは、*.sq4.lzm にリネームします。
pkgtar2lzm の際に、*.sq4.lzm としても、なぜか無視されてしまうのです。

次に、http://www.debian.org/distrib/packages にて、
以下の３本を入手、右クリックで、sq4.lzmに変換します。

1. libboost-dev_1.34.1-14_i386.deb
2. libboost-serialization-dev_1.34.1-14_i386.deb
3. libboost-serialization1.34.1_1.34.1-14_i386.deb

## 依存ファイルをまとめて、**1**本にします。

これで、パッケージが揃いました。依存ファイルをまとめて、1本にします。
本体だけは、独立させておくことにしました。

```
#!/bin/bash
### inkpack.sh
mkdir           pak
lzm2dir         atkmm-2.22.1-1-i686.sq4.lzm    pak
lzm2dir         bzip2-1.0.6-1-i686.sq4.lzm       pak
lzm2dir         cairomm-1.8.6-1-i686.sq4.lzm       pak
lzm2dir         gc-7.1-1-i686.sq4.lzm       pak
lzm2dir         glibmm-2.24.2-1-i686.sq4.lzm       pak
lzm2dir         gsl-1.14-1-i686.sq4.lzm       pak
lzm2dir         gtkmm-2.22.0-1-i686.sq4.lzm       pak
lzm2dir         gtkspell-2.0.16-1-i686.sq4.lzm       pak
lzm2dir         hicolor-icon-theme-0.12-1-any.sq4.lzm       pak
lzm2dir         imagemagick-6.6.4.10-1-i686.sq4.lzm       pak
lzm2dir         libboost-dev_1.34.1-14_i386.sq4.lzm       pak
lzm2dir         libboost-serialization-dev_1.34.1-14_i386.sq4.lzm       pak
lzm2dir         libboost-serialization1.34.1_1.34.1-14_i386.sq4.lzm       pak
lzm2dir         libsigc++-2.2.8-1-i686.sq4.lzm       pak
lzm2dir         openjpeg-1.3-3-i686.sq4.lzm       pak
lzm2dir         pangomm-2.26.2-1-i686.sq4.lzm       pak
lzm2dir         poppler-0.14.5-1-i686.sq4.lzm       pak
lzm2dir         poppler-data-0.4.4-1-any.sq4.lzm       pak
lzm2dir         poppler-glib-0.14.5-1-i686.sq4.lzm       pak
lzm2dir         renderproto-0.11.1-1-any.sq4.lzm       pak
dir2lzm         pak       inkscape_deps--0.0.1.lzm
```

このファイルと用意のできたlzmファイルを、同じフォルダーに入れ、
# chmod 755 inkpack.sh　リタン。
# ./inkpack.sh　　リタン。
# chmod 777 inkscape_deps--0.0.1.lzm　　リタン。
inkscape_deps--0.0.1.lzm と、inkscape-0.48.0-4-i686.sq4.lzmを、ダブルクリックでインストール、
K Menu -> Graphics から、インクスケープを立ち上げます。

## Minimal nano


不可視ファイルの設定、編集には、ナノを使います。
その理由は、バグがない為、エディターが原因の不可解な現象というのを、
およそ考えなくてよい、ということです。
使い方は、かんたんです。
新しくファイルを作る時、名前を考える必要があります。
例
# nano myfile
すでに存在するファイルを呼ぶとき、
# nano myfile
この時、実は、ファイル名を間違えていたとします。
この場合、このファイル名で、新たに作成されてしまいます。
これを防止するには、とにかく　# ls を行って、
ファイル名を確認します。

カーソルの移動は、アローキーで行います。
コマンドキーは、２つ覚えます。
コントロールキー　と　アルファのオー　で、セーブ。
このとき、エディターの下部に名前が出ますので、リターンします。

終了して戻るとき、
コントロールキーとエックスを押します。

これらは、下に書いてありますので、忘れたら、それを見ます。

ナノの続き。


設定ファイルでは、１つの設定は、あくまで１行で書くという掟があります。
長くなると厳しいですが、リターンキーは打ちません。
ナノが自分で折り返しますが、あくまで、１行扱いになります。
リターンキーを読み込まないというオプションも存在しますが、
原則、これは使いません。
# nano -w .bashrc
長くなって、どうしても折り返したい場合は、エスケープシーケンスを
使います。
エスケープシーケンスは、バックスラシュですが、
これが厳しいのは、￥が出てしまう場合で、これは使えません。
従って、わたしたちは、折り返さないで、ナノにやってもらいます。

# ステージ１８　ラップアップ。


2011年, スラックス リミックスは、スラックスから飛び出して、
新たな方向へ踏み出しました。
ポーティアス、使ってみると、本当にいい感じ、今後の期待、高まります。
SLAX-REMIX AKA Porteus　の良い点についてまとめると、

1. ライブCDに、使いたいアプリを加えて、CDを作り直す、つまり、
   カスタムライブCDが、簡単に作れる。
   例えば、何らかのクラスで、フォントフォージを使ったフォントの作り方のセッションを
   行いたいとき、コンピューターには手をつけず、人数分のカスタムCDを用意します。
   セーブする場所は、だれかが代表してgmailに共有ファイルを作ったりすれば、
   何とかなるでしょう。
   コンピューター自体にインストールしないので、メンテは楽になります。
2. lzmファイルは小さい。
   例えば、インクスケープの場合、アップル用のdmgファイルのサイズは、79.4MB。
   これを展開したアップのサイズは、275.8MB。
   アーチリナックスのパッケージサイズは、13MB。
   これを変換したlzmファイルは20.4MB、依存ファイルは、全体で、11.8MB。
   で、lzmファイルの合計は、32.2MBとなります。
   これなら持ち運びにとても便利、ポータブルという名前もうなずけます。
3. パッケージをそのまま使うイメージ。
   普通、パッケージは、ダウンロードするまでの姿で、それは、展開し、インストールされます
。
   lzmファイルは、モジュールスフォルダーに置けば、それで、インストール、
   つまり、追加完了となり、そのまま利用可能な状態になります。
   このとき/usr/libフォルダーを開いておくと、
   ファイルがリアルタイムで出来ていくのを見ることができます。
   例えば、ジニーのように依存ファイルのないアプリでは、
   このlzmファイルを消去すれば、きれいに終わりとなります。
   通常、リナックスでHDDにインストールするケースでは、ことはこれほど簡単ではないかもしれません。