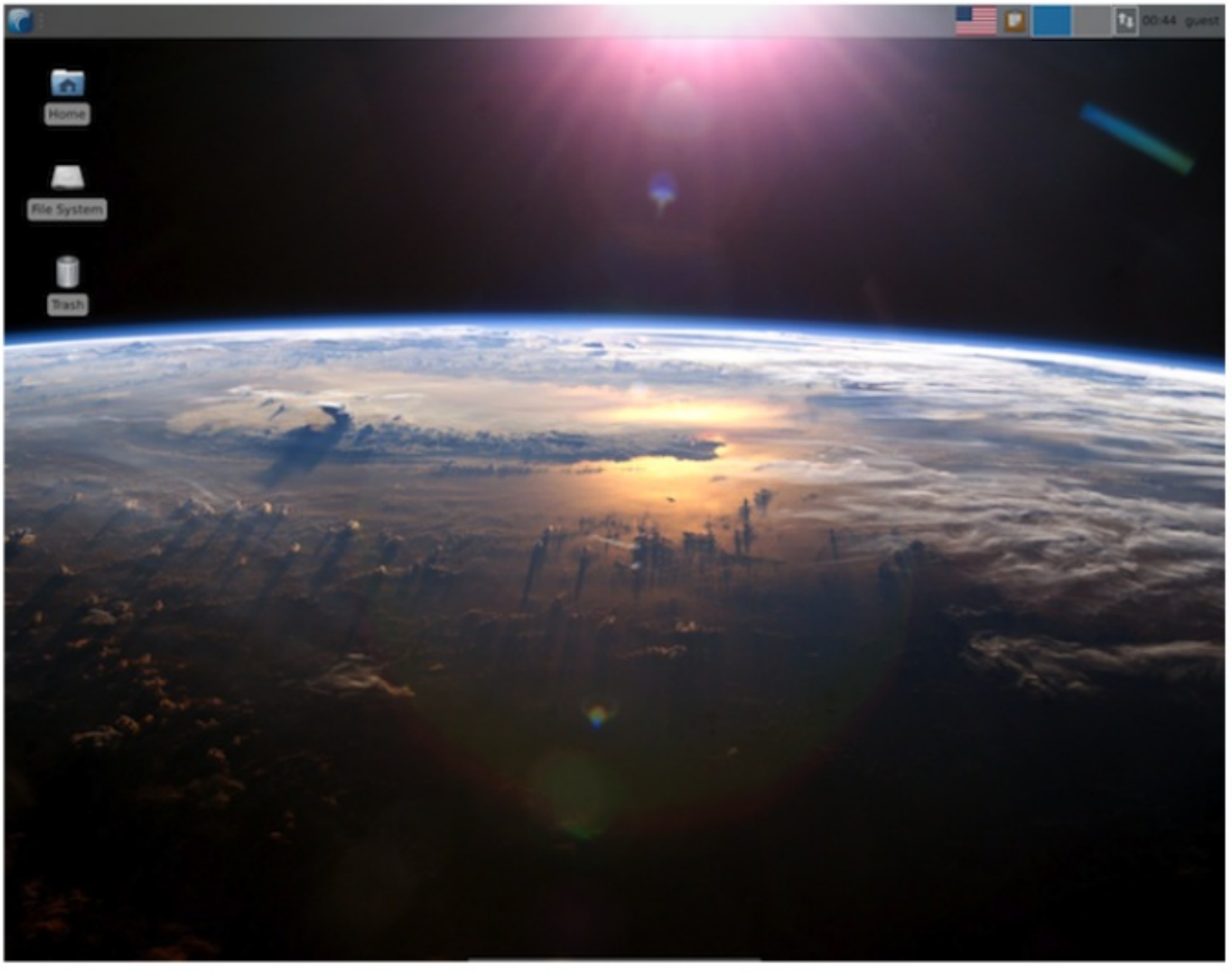
coolda

64bit

だってクールだ Porteus 64

porteus



https://porteus.org/

ポーティアス**v1.2 rc2 64**を、アップルで試す。

アップルでは、USBペンからのブートをサポートしていません。
しかし、ポーティアスをUSBペンに入れた場合、
別途、スタート用のCDを用意すれば、使えるようになります。
これを使って、
1. USBペンを差し込む。
2. アップルを立ち上げる。
3. CDトレイにスタートCDをマウント。
4. 再起動。
5. ファンファーレの直前からオプションキーを押したままにする。
6. 小さな矢印が出たら離す。
7. 左にHDD、右にCDのイメージが出たら、アローキーで、右のCDをセレクト、リターン。
以上、書くと長いですが、具体的には、5, 6, 及び7, を行います。

スラックウェアOSは、メインはKDEとなっていますが、
XFCEも用意されています。
ここに来て、リーナストーバルスの、KDEではなく、XFCEにする発言、
もあって、XFCE好感度は、急上昇中です。
余計なものは、デスクトップに置かないというプロテイストが好感を
呼んでいます。
それでは、この新作を、アップルに立ち上げ、テストしてみましょう。
実際、このスタートCDがあれば、マシンを選びませんので、
インテル互換の64ビットCPU搭載の全てのマシンで、利用可能です、
と云いたいところですが、例えばマシンにhddが1台の時のみ、
という制限がつきますので、ご注意下さい。
必要なもの。
1. 未使用のCD、２枚程度。
2. USBペンドライブ、4GB以上がおすすめ。

ステージ**1** ファイルのダウンロード。

https://porteus.org/distro-download/download-latest-64-bit.html　にて、
testing から、
Porteus-v1.2-rc2/ をクリック。
Porteus-XFCE-v1.2-rc2-x86_64.iso　199.2M
をダウンロードします。
ホームにisoの名前でフォルダーを作りそこに入れておきます。
http://sourceforge.net/projects/gparted/files/gparted-live-stable/0.12.1-5/gparted-live-0.12.1-5.iso
129.6MB
をdownload、isoフォルダーに入れておく。
イソファイルをCDに焼きます。
ご注意: イソファイルの場合、右クリックでCDに焼く方法は使えません。
1. アプリケーション　-> ユーティリティ　-> ディスクユーティリティ
ディスクユーティリティを立ち上げ、
2. ファイル　-> ディスクイメージを開く
isoフォルダーへナビゲートして、開く。
3. 左サイドのリストにある、gparted-live-0.12.1-5.iso　をセレクト。
　　左上、ディスクを作成　をクリック、トレイがオープンされるので、
　　CDを入れ、閉じるをプッシュ。
　　ディスクを作成が青色になったらリタン。
現在、スラックスは、slax7を開発中ですが、かなり遅れています。
もしもslax6をお持ちでない場合、7が出た時点で、消える可能性がありますので、
必ずダウンロードしておくこと、おすすめです。
これは、見事な作品であり、一家に一台、必要なものです。
ここでCDに焼く必要はありませんが、
中に入っているslaxsave.zipファイルを使いたいので、
ダウンロードしておきます。
http://www.slax.org/get_slax.php
にて、slax-6.1.2.iso　を入手します。

ステージ**2** 作業手順。

1. ジーパーテッドライブを使い、USBペンをfat32で初期化。
2. Porteus-XFCE-v1.2-rc2-x86_64.isoをダブルクリック、
   中身のbootとporteusフォルダーをUSBペンにコピー。
3. slax-6.1.2.isoをダブルクリック、中身のslaxフォルダーをダブルクリック、
   中にあるslaxsave.zipを、slaxsave_datのようなフォルダーを作り、そこにコピー。
   slaxsave.zipをダブルクリック、
   slaxsaveというフォルダーが出来るので、ダブルクリック、
   中にある、save1024.zipをUSBペンにコピー。
   USBペンにコピーしたsave1024.zipを、ダブルクリック。
   少し時間を要しますが、slaxsave.datが生成されます。
   名前を、mysave.datに変更。これは、changes=など、mysave.datにしてあるので、
   お守り下さい。
4. スタート専用のブートCDですが、マシンにhddが1台の場合のイソファイルが、
   githubに用意してあります。
   githubを利用しない場合は、porteus.cfgの変更を行い、make_iso.shを使って、
   イソファイルを生成します。

それでは、作業開始です。

ステージ３　ジーパーテッドライブを使い、**USB**ペンを**fat32**で初期化。

1. ジーパーテッドライブCDをF12でトレイを開け、マウント、F12で閉めます。
2. 左上、リンゴポップダウンから再起動。
3. オプションキーを押したままにする。
4. 小さな矢印が出たら離す。
5. 左にHDD、右にCDのイメージが出たら、アローキーで、右のCDをセレクト、リターン。
6. 下の部分に、Gnome Partition Editor と大きなサインが出たらリターン。
7. Don't touch keymap と赤い字が出たら、リターン。
8. Which language do you prefer? で、15と入れ、リターン。
9. Which mode do you prefer? では、[0] となっているので、リターン。

これで、ジーパーテッドの画面が出ます。
現在は、マシン本体のhddの状態がでていますので、左上、
GParted(G)　-> デバイスを選択(D) /dev/sdb (3.73 GIB)
をセレクトします。
USBの状態が示されます。

1. 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、 close。
2. 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。
3. 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。
4. /dev/sdb1 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。
5. 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にチェックマーク、OKを押す。
6. CDを取り出し、リターン。

7. 初期化の済んだUSBペンに名前をつけておきます。

ステージ４　ギットハブからイソファイルを入手

https://github.com/coolda/bootpo/downloads　にて、
Download as zip または、Download as tar.gz　を押して、
ダウンロードしておきます。
この際ギットハブに登録してみようかの場合、その方法をご紹介しておきます。
https://github.com/
にて、ど真ん中の、Plans, Pricing and Signup をプッシュ。
一番上の$0/mo Free for open source の　右にある　Create a free account をクリック。
入力は、
1. Username : このサイトで使う、あなたのユーザー名を入力します。
2. Email Address : あなたのEメールアドレス。
　　ご自分で作成したプログラムも、自分の場所に置けますので、
　　パソコンのメールアドレスが便利かと思います。
3. Password : パスワードは、７文字以上、その中には、
　　最低、小文字アルファ１つ、数字１つを含めます。
4. Confirm Password : 3を繰り返す。
そして、Create an account を押します。
以上で終了です。
これで、次回から、ログインすることができます。
ログインしましたら、
https://github.com/coolda/bootpo/downloads
にアクセスして、ページの５行目辺りにある、
Download as zip または、Download as tar.gz どちらかをクリックします。
ダウンロードしてきたファイルをダブルクリックし、
生成されたフォルダーの中にある、boot12rc2_64.iso を、ディスクユーティリティで、
CDに焼きます。
これで、冒頭に掲げた手順で、立ち上げて下さい。
/home/guest に、workのようなフォルダーを作り、再起動して、
これが、ちゃんと存在していれば、成功です。

ステージ５　パッケージマネジャーを使う。

ポーティアスのレポは、まだそれほど大きくはありませんが、
先に、パッケージを取り込むプログラムが完成しています。
依存ファイルも取り込んでくれますので、便利になりました。
System -> Porteus Package Manager で、ダイアログが出ます。
ルートでの操作になりますので、'toor' とタイプ。
パッケージの収納場所は、マシーンにHDDが1台、USBペンはFAT32で、初期化、
そして、パーティションを切ってなければ、 sdb1となります。
この場合、/mnt/sdb1/porteus/modules とタイプします。
試しに、下の画面にあるSlackwareをプッシュしてみます。
データベースを作るので、200Mb程必要です、と聞いてきます。
Yesとします。以降何かダイアログが出ましたら、Yesとして下さい。
ルビーは、ディフォルトで入っていないので、インストールしてみます。
サーチにrubyと入れ、探します。12.0からサーチすると、ruby-1.8.7_p72-i486-2
が出ました。これをセレクトしますと、インフォが出ます。
Download nowをプッシュ、convert after downloadもセレクト。
これで、xzmモジュールは/mnt/sdb1/porteus/modules に収納されます。
/mnt/sdb1/porteus/modules に行き、右クリックすると、
open with activate...というのが 1 番上の方にありますので、セレクト。
これで、すぐに使えるようになります。

ステージ６　**NASM**を使ってみる。

64ビットのOSでは、マシンが用意している、
64ビットレジスタを、利用することができます。
32ビットのみ対応のCPUには、この部品がついていません。
従って、64ビットレジスタを使用したプログラムは、動きません。
この事実を前にした時、64ビットのOSを使うという意味が、
明確になります。

1. マシンが64に対応している場合、その部品代は、すでに支払っている。
2. 32ビットOSでは、64ビットレジスタをフルに使えません。
   ただし、32ビット用として、その半分を使うことはできます。
   この場合、CPUは、64に対応している必要があり、
   純然の32ビットマシンでは動かないことになります。
   なので、この方法には、無理があります。
3. マシンが64に対応しているのに、32ビットOSを使用する場合、
   古いマシンと同列の扱いとなります。

64ビットになることで大きく変わったことは、

1. レジスタとメモリ間を移動する時の単位あたりのサイズが
   ４バイトから８バイトになった。
2. ポインターのサイズは、通常、８バイトになった。
2. CPUが用意する、64ビットレジスタをフルに使える。

インテルの80x86では、わたしたちが利用できる汎用レジスタは、
ずっと８本でした。
現在は、64ビット汎用レジスタが16本、浮動小数点用も16本、
用意されており、これをバンバン利用することができます。
64ビット汎用レジスタの名前は、先頭の文字がrになりました。
rax, rbx, rcx, rdx, rbp, rsp, rsi, rdi
r8, r9, r10, r11, r12, r13, r14, r15
浮動小数点用は、xmm0..xmm15
それでは、nasmを使って、わたしたちもこの64汎用レジスタを利用できるか、
確認します。

NASMは、すでにインストールされています。

nasmは、ポーティアスに組み込みで入っていますので、
モジュールを取り込んでくる必要はありません。コンソールから、
$ nasm -v リターンで、
バージョンナンバーが出てくれば、すぐに利用可能です。
動作確認のプログラムは、ハローワールド。
プリントエフ関数を呼び出すことが、全てです。
関数を呼び出すとき、64ビットのデータとして引数を渡す時は、
以下のレジスタを使います。
整数型のデータ／ポインターの場合、
rdi　　1番目の引数に使う。
rsi　　2番目の引数。
rdx　　3番目の引数。
rcx　　4番目の引数。
r8　　5番目の引数。
r9　　6番目の引数。
6本以上の場合は、スタックに積みます。
関数からの戻り値は、スタックではなく、raxに戻ります。
戻り値が浮動小数点の場合、xmm0 に戻ります。
ストリングはCのストリングなので、終了のマークとしてゼロを置きます。

```
;;; hello.asm
global main
extern printf
section .text

main:
    mov rdi, msg        ;;; rdi use to pass 1st argument to functions
    xor rax, rax
    call printf
    ret
msg:
    db 'Hello, World', 10, 0    ;;; newline, null terminator
```

コンパイルします。

mainの前にアンダーラインは、必要ありません。
必要なら、nasmが加えるとのこと。
gccを使う場合、_startのような記述は、mainに変更するように、
という指示が出ますので、変更して下さい。
１番目の引数として、msgをrdiに収納します。
xor rax, raxは、レジスタをゼロで初期化するときの定番です。
これが、２番目の引数。
これで、printf()を呼びます。
pushとかpopがありませんので、とてもすっきりしています。
retが戻りの操作をやってくれるので、簡単です。
コンパイルは、コンソールから、
$ nasm -f elf64 hello.asm
これで、hello.oファイルができます。
$ gcc -m64 -o hello hello.o
これで、hello　というバイナリファイルができます。
実行は、
$ ./hello

ステージ７　**C**から**NASM**のファイルを呼び出す。

Cから、nasmのコードを呼び出す方法を見ます。
引数として、イントを３つ渡し、最も大きい値を返すという例を試します。

```
;;; max3.asm

global max3
section .text
max3:
  cmp edi, esi
  cmovl edi, esi
  cmp edi, edx
  cmovl edi, edx
  mov eax, edi
  ret
```

すべて32ビットのレジスタを使っています。
これなら全く同じファイルを32、64で変更なしで使えるというわけです。
コンパイルは、

```
$ nasm -f elf64 max3.asm
```

これを呼び出して使う側のCファイルは、

```
// usemax.c
#include<stdio.h>

int max3(int, int, int);
int main() {
  printf("%d\n", max3(16, 256, 32));
  return 0;
}
```

これをコンパイルすると、

コンパイルは、
$ gcc -m64 -o use usemax.c max3.o

実行は、
$ ./use

64ビットの実行ファイルを作るということは、
例えば、Cでは、gccに、-m64　フラッグを立ててコンパイルする。
これだけですので、利用できるプログラムもどんどん増えてくると思います。
なので、64ビットOSはおすすめです。

ステージ８　最強スクリプト、**cint**をインストール。

cintは、C及び、C++の、スクリプト/インタプリター版です。
これを達成するために、40万行のコードを書き込んだとのこと。
C++は、そういうことはできません、とは、絶対言いたくない言語ですから、
クローン化を目指した場合、このような恐ろしいことになるのかと思います。
cintは、後藤正治さんの作品で、大変な努力の成果がここにあります。
http://root.cern.ch/drupal/content/cint　にて、
cint-5.18.00.tgz　を入手します。
今回、そういえば、と思ってインストールして、見事に動作していますが、
例えば、アップルでは、コンパイルの通りにくいアイテムなので、
ちょっとビックリしています。

1. 先に、依存ファイル、readline-5.2-x86_64-4.txz　を入手します。
   http://packages.slackverse.org/　にて、
   readline と入れ、Slackware64 13.37を選択、サーチします。
   ダウンロードできましたら、右クリックで、
   Convert tgz/txz to xzmをセレクト。
   これを /porteus/modulesフォルダーに収納、
   右クリックの先頭、Open with "activate"をセレクト。
   今、マウスでドロップしますと、コピーの形になります。
   つまり、元フォルダーには、データが残ります。
   なので、ゴミ箱に送るなりしてください。
2. cint-5.18.00.tgzを、/home/guestに置き、
   ターミナルから、
   # tar -xvfz cint-5.18.00.tgz とします。
   cint-5.18.00フォルダーができますので、そこに移動します。
   # cd cint-5.18.00
   # ./configure
   # make

3. 以下のスクリプトをテキストにコピーしましたら、
　　set_cint.sh　の名前で、/home/guest などに置き、
　　# chmod 755 set_cint.sh
　　# ./set_cint.sh　と操作して下さい。

```
#!/bin/bash
### set_cint.sh
mkdir -p mycint/usr/local
mv /home/guest/cint-5.18.00/bin mycint/usr/local/
mv /home/guest/cint-5.18.00/lib mycint/usr/local/
dir2xzm mycint /mnt/sdb1/porteus/modules/cint518.xzm
```

/porteus/modules/ に出かけ、cint518.xzmを右クリック、
先頭の、Open with "activate" をセレクト。
4. nanoを入手します。viを苦手としないのであれば、
　　viは、入っていますので、ここは、スキップしてください。
　　http://packages.slackverse.org/　にて、
　　nano と入れ、Slackware64 13.37を選択、サーチします。
　　nano-2.3.1-x86_64-1.txz　が出ますので、ダウンロード。
　　ダウンロードできましたら、右クリックで、
　　Convert tgz/txz to xzmをセレクト。
　　これを /porteus/modulesフォルダーに収納、
　　右クリックの先頭、Open with "activate"　をセレクト。
5. ドットバッシュアールシーの編集。
　　ターミナルをホームで立ち上げ。ナノで.bashrcをオープン。

```
# nano .bashrc
```

　　すでに、ブロークンマンが何か書いていますので、
　　アローキーで、カーソルを下に送り、
　　新しいところに、以下の３行をコピーします。

```
export CINTSYSDIR=/home/guest/cint-5.18.00
export PATH=$PATH:$CINTSYSDIR
export LD_LIBRARY_PATH=.:$CINTSYSDIR:$LD_LIBRARY_PATH
```

コントロールと'O' リターン、
コントロールと'X'。
以上で作業はすべて操作できました。
続いて、テストプログラムを作成します。

テストプログラムを作成します。


```
// hello.cpp
#include<iostream>
int main() {
    cout << "Hello" << endl;
}
```

これを、テキストに書き込み、hello.cppとして、ctesのようなフォルダーに置きます。

```
$ cd /home/guest/ctes
$ cint hello.cpp
guest@porteus:~/ctes$ cint hello.cpp
Hello
```

1. ファイルを呼ぶ時、./hello.cppとしなくてよい。
2. int main() の先頭のintは、省略できる。
3. using namespace std;　は、省略できる。
4. g++のようなコンパイラーを使う場合は、
   using namespace std;を書き加える。

個人的な感想ですが、通常のC++プログラムで、
ほぼまともに動いてくれれば、
これは実際、史上最強のインタープリタだ、と思っています。

ステージ９　メニューを日本語にする。

日本語でメニューなどを表示する部品は、ポーティアスのサーバーにあります。
注文を入れて、ダウンロードするという方法で、ライブCDのサイズを軽減しています。
最初に日本語のフォントを入手しておきます。
http://sourceforge.jp/projects/slackware/downloads/52999/ipa-fonts-ttf-00203-noarch-2.txz
/home/guest/Downloadsにダウンロードできましたら、
右クリックで、Convert tgz/txz to xzm をセレクト、xzmに変換。
このモジュールを所定の場所、/mnt/sdb1/porteus/modulesに移動します。
mv /home/guest/Downloads/ipa-fonts-ttf-00203-noarch-2.xzm /mnt/sdb1/porteus/modules
Menu -> System -> Porteus Language Selection Tool をクリックします。
ルートでの作業ですので、toorとタイプ。ダイアログが現れます。

1. 出来上がりサイズの説明があります。
   本体が24Mb 程度になるとのこと。
   フレームを少し拡大して、右下のnextボタンを見えるようにして、
   nextをクリック。
2. I want UTF-8 encoding にチェックを入れます。
   すぐ下のフィールドからナビゲート、ja_JP.utf8 まで下り、セレクト　next。
3. KDE environment language
   今回はXFCEですので、左側は触らず、右側の、Choose system languageに
   チェックを入れ、ナビゲートして、ja をセレクトして下さい。
4. Tweaking the KDE keyboard layout
   今回は、関係ありませんので、何もせず、next。
5. Windows support option
   何もせず、next。
6. The final stage
   BUILD をクリックします。

7. OFFLINE MODE
   この設定操作を以前にやっていて、今回はその修正などのケースの場合、
   以前のデータを保管してある場所までナビゲートしてくれれば、
   それを使いますよ、と云っています。
   今回が最初のトライの場合は、Cancelボタンをクリックします。
   すると、glibc-i18n　などのダウンロードが開始されます。
8. Manual pages languages
   今回は、ja　を入れます。
9. ダウンロードしたものをモジュール化すると云います。OK。
10. パッケージは、/porteus/modules フォルダーに収まりました。
   再起動が必要です。OK。
11. create OFFLINE module?
   次回のために、データとして作っておくか、聞いています。
   Yes。
   デスクトップに、lst-offline-bundle.xzm　というモジュールが作成されます。
   これを、/home/guestに、offline_moduleのようなフォルダーを作成して、
   保管しておきます。
   root@porteus:/home/guest# mv Desktop/lst-offline-bundle.xzm /home/guest/offline_modul

以上で、全ての作業は終了しました。
再起動してから、メニューの様子を確認して下さい。
日本語になっていれば成功です。

ステージ１０　テキストの日本語入力をセットする。

http://packages.slackverse.org/　に出かけ、
以下のファイルを、slackware64 13.0 から探します。
1. anthy-9100e-x86_64-1.txz
2. scim-1.4.9-x86_64-4.txz
3. scim-anthy-1.2.4-x86_64-2.txz
4. scim-bridge-0.4.16-x86_64-4.txz
ブルーフィッシュが必要であれば、slackware64 13.37 から、
bluefish-2.2.1-x86_64-1sl.txz を入手します。
全て右クリックで、Convert tgz/txz to xzm をセレクトして変換します。
以下のファイルをモジュールを納めた/home/guest/Downloadsなどに置き、
chmod 755 mov.sh
./mov.sh にて、モジュールを/mnt/sdb1/porteus/modulesに移動させます。
#!/bin/bash
mov.sh
mv /home/guest/Downloads/anthy-9100e-x86_64-1.xzm /mnt/sdb1/porteus/modules/
mv /home/guest/Downloads/scim-1.4.9-x86_64-4.xzm /mnt/sdb1/porteus/modules/
mv /home/guest/Downloads/scim-anthy-1.2.4-x86_64-2.xzm /mnt/sdb1/porteus/modules/
mv /home/guest/Downloads/scim-bridge-0.4.16-x86_64-4.xzm /mnt/sdb1/porteus/modules/
mv /home/guest/Downloads/bluefish-2.2.1-x86_64-1sl.xzm /mnt/sdb1/porteus/modules/
再起動後、エディターを立ち上げ、コントロール+スペースバーで、Anthyが出れば成功です。
アルファに戻すには、再度コントロール+スペースバーをヒットします。

ステージ１１　ニューリスプをインストールする。

newLISP is a registered trademark of Lutz Mueller.
ラムダ式を使うなら、始めからラムダ式を使うようにデザイン
されている言語を使いたい。
それで、使い方のやさしいのがよい。
という要望に、真っ向から対応しているリスプ系の言語が、
最低でも２つ存在します。
１つは、clojureで、もう１つは、ニューリスプです。
今回はインストールするニューリスプは、
1. 組み込み関数を、400程度に抑え、スリムになっている(xzmで864kb)。
2. ジャバベースの、軽快なGUIが付属している。
3. C系の言語で使われるループなどは、殆ど同じものが用意されている。
4. 配列、ストリング、リストは、ほぼ同じような操作が出来る。
5. リスプのみならず、Schemeの良い部分も取り込んでいる。
ニューリスプは使えば楽しくなること間違いなしですが、
リナックスのx86_64では、ソースからビルトすることになります。
また、GUIがジャバのJREを必要とするので、
このかなり大きめのファイルをインストールすることになります。
しかし、それだけの価値はあります。
ダウンロードは合計４本行います。
1. http://www.newlisp.org/index.cgi?page=Downloads
　download from newlisp.org: newLISP v.10.4.3 source
2. http://sourceware.org/libffi/　にて、
　libffi-3.0.11.tar.gz　を入手。
3. http://ftp.gnu.org/gnu/readline/　にて、ファイルの下から５番目、
readline-6.2.tar.gz　13-Feb-2011 15:50　2.2M　を入手します。
4. http://www.oracle.com/technetwork/java/javase/downloads/index.html
Linux x64 (64-bit)　20.32 MB　　jre-6u32-linux-x64.bin
Linux x64 (64-bit)　31.36 MB　　jre-7u4-linux-x64.tar.gz
上記のどちらかをダウンロードします。
ここでは、jre7u4、すなわち、jre1.7.0_04を選択しました。

ナノをインストールしていない場合は。

以上の４本が、/home/guest/Downloads に収納されたものとして操作を進めます。
ところで、バッシュアールシーの操作などに、nanoを利用する場合
'シーイントをxzmモジュールにする'　を参照の上、インストールして下さい。
1. /home/guestに、workという名前で、フォルダーを作成。
2. 以下のファイルを、mydeps.shの名前で、収納します。

```
#!/bin/bash
### mydeps.sh
### ffi.sh part
mkdir /home/guest/ffi_tar
mv /home/guest/Downloads/libffi-3.0.11.tar.gz /home/guest/ffi_tar
cd /home/guest/ffi_tar
tar -xvzf libffi-3.0.11.tar.gz
cd libffi-3.0.11
./configure
make
mkdir /tmp/myffi
make install DESTDIR=/tmp/myffi
dir2xzm /tmp/myffi /mnt/sdc1/porteus/modules/libffi64-3.0.11.xzm
activate /mnt/sdc1/porteus/modules/libffi64-3.0.11.xzm
### readline.sh part
mkdir /home/guest/rline_tar
mv /home/guest/Downloads/readline-6.2.tar.gz /home/guest/rline_tar
cd /home/guest/rline_tar
tar -xvzf readline-6.2.tar.gz
cd readline-6.2
./configure
make
mkdir /tmp/rline
make install DESTDIR=/tmp/rline
dir2xzm /tmp/rline /mnt/sdc1/porteus/modules/readline64-6.2.xzm
activate /mnt/sdc1/porteus/modules/readline64-6.2.xzm
```

mydeps.shを実行します。

3. ルートでworkに行き、mydeps.shを実行。

$ su

toor

cd work

chmod 755 mydeps.sh

./mydeps.sh

次は、jreを操作します。

1. /home/guestにjre7u4_tarというフォルダーを作成。

2. ここにjre-7u4-linux-x64.tar.gzを移動。

mv /home/guest/Downloads/jre-7u4-linux-x64.tar.gz /home/guest/jre7u4_tar

cd /home/guest/jre7u4_tar

tar -xvzf jre-7u4-linux-x64.tar.gz

3. 出来たjre1.7.0_04フォルダーをmyjre7以下に移動。

cd ../

mkdir -p myjre7/usr/java

mv /home/guest/jre7u4_tar/jre1.7.0_04 /home/guest/myjre7/usr/java

dir2xzm myjre7 /mnt/sdb1/porteus/modules/jre64-1.7.0_04.xzm

activate /mnt/sdb1/porteus/modules/jre64-1.7.0_04.xzm

ニューリスプのターファイルを**xzm**に。

続いて、newlisp-10.4.3.tar.gzを操作します。

1. /home/guestにnewlis_tarというフォルダーを作成,
　ここにターファイルを移動させます。

```
# mv /home/guest/Downloads/newlisp-10.4.3.tar.gz /home/guest/newlis_tar
# cd /home/guest/newlis_tar
# tar -xvzf newlisp-10.4.3.tar.gz
```

出来たnewlisp-10.4.3フォルダーに移動、コンフィギュア、メイクを行います。

```
# cd newlisp-10.4.3
# ./configure-alt
# make
```

インストール用のフォルダーを/tmpに作成、インストール。

```
# mkdir /tmp/mylis
# make install DESTDIR=/tmp/mylis
```

xzmモジュールにする。

```
# dir2xzm /tmp/mylis /mnt/sdb1/porteus/modules/newlisp64-10.4.3.xzm
```

/usr/binにシンボルリンクを１つ作ります。

```
# cd /usr/bin
# ln -s /usr/local/bin/newlisp newlisp
```

nanoから.bashrcを編集します。

```
$ nano .bashrc
```

アローキーでテキストを送り、空いている所に以下の３行をコピーペーストします。

```
export NEWLISPDIR=/usr/local/share/newlisp-10.4.3
export JAVA_HOME=/usr/java/jre1.7.0_04
export PATH=$PATH:/$JAVA_HOME/bin
```

/porteus/modulesに出かけ、newlisp64-10.4.3.xzmを右クリックして、アクティベイト。

コマンドラインから、$ newlisp-edit を呼びます。

これで、GUIが立ち上がってくれれば成功です。

使い方は、http://p.booklog.jp/book/50646　をご参照下さい。

ステージ１２　ラップアップ。

XFCE版、200メガというサイズ、動き、快調です。
xfceは、まともすぎて、という声もありますが、
デスクトップを軽くしておきたいという、
プロ指向の方にとって、xfceは、強い味方となり得ます。
一方、32ビット版は、KDEバージョンも非常に高速ですので、
どちらを使うか迷ってしまいます。
ポーティアス、選択肢が拡がって、さらに魅力、上昇中。