Linuxによる組込みシステム開発
Fedora7
③

港湾職業能力短期大学校　神戸港

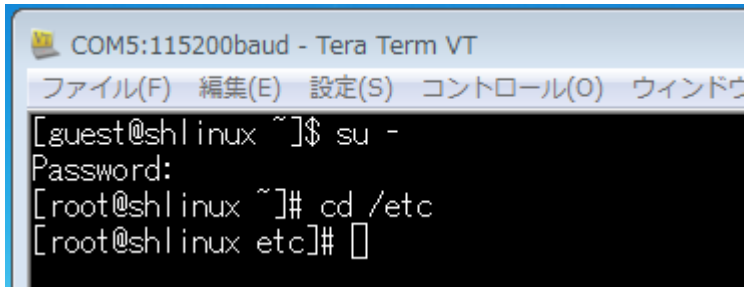
マイコンボード：T-SH7706LSR(TAC製)

マイコン：SH7706（SH3ファミリ）：ルネサスエレクトロニクス

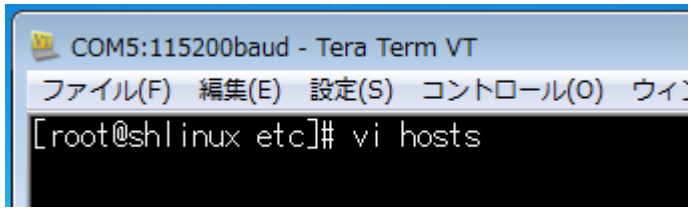
この教材は
みついわゆきお氏のmes2サイト
http://mes.sourceforge.jp/mes2/
およびLinux記事
http://gihyo.jp/dev/serial/01/micom-linux/0001
および、名もないページさんサイト（2015.4現在リンク切れです）
http://wave2.iobb.net/doc/summary/sh3wiki/wifky.cgi
の内容を参考に作成しています

Xクライアントのssh接続（1）

1. ターゲットボードのXサーバ設定を行う。/etcにrootユーザで入る。

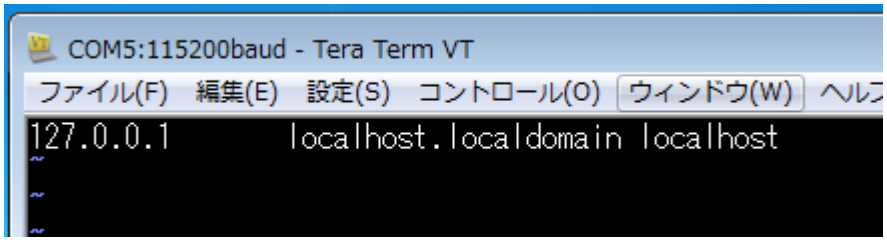


2. hosts ファイルを作成する。



3. hostsの内容は

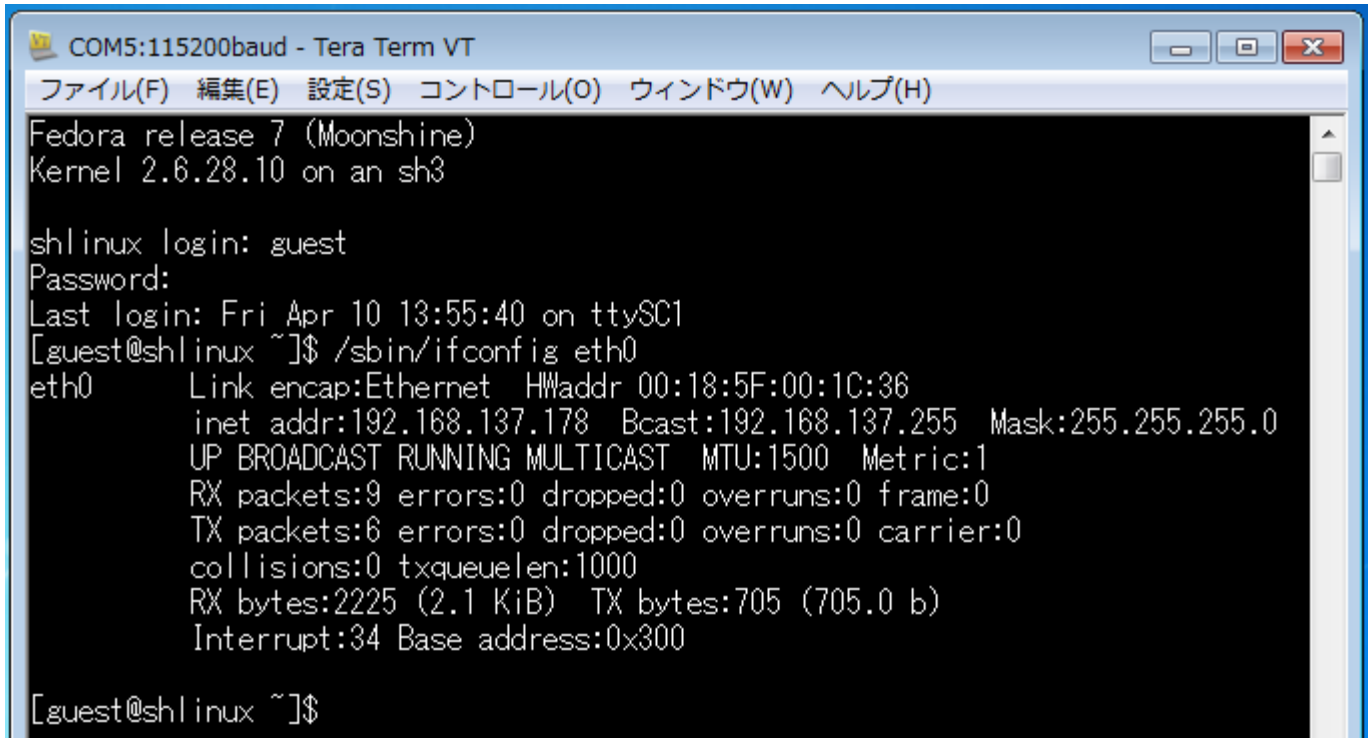
   127.0.0.1　　localhost.localdomain　localhost　とし、保存する。



Xクライアントのssh接続（2）

3. 再起動し、下記の例ではターゲットボードが192.168.137.178に割り振られていることを確認する。

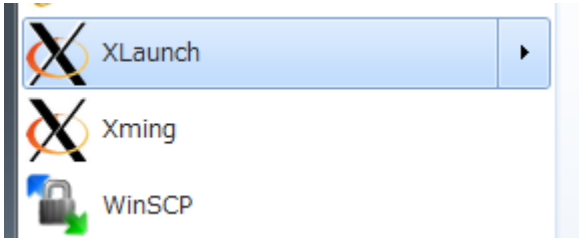
$ /sbin/ifconfig　eth0



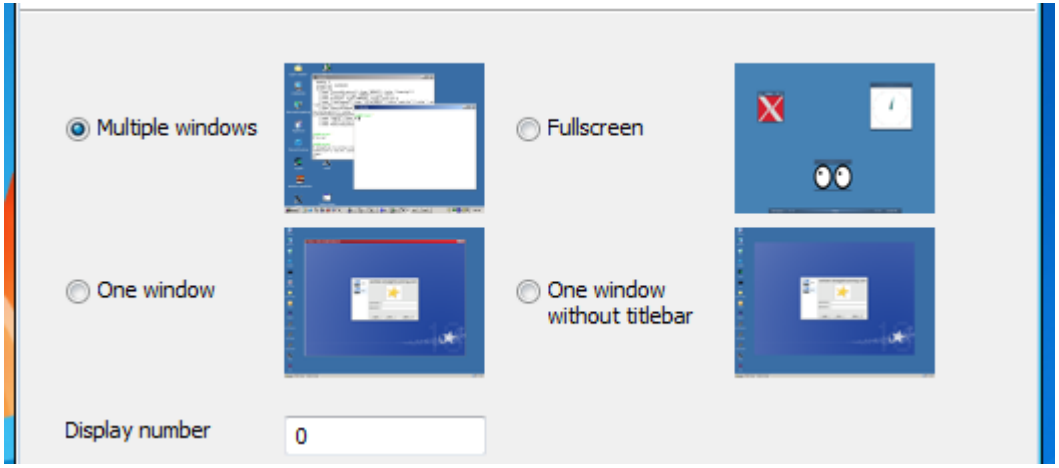
Windowsパソコンでのssh接続（1）

1. ターミナルとしては低速だが、Windowsマシンでターゲットボードのxウインドウを開き操作することができる。

   XLaunchを起動する。（Xmingのインストールについては附1参照）

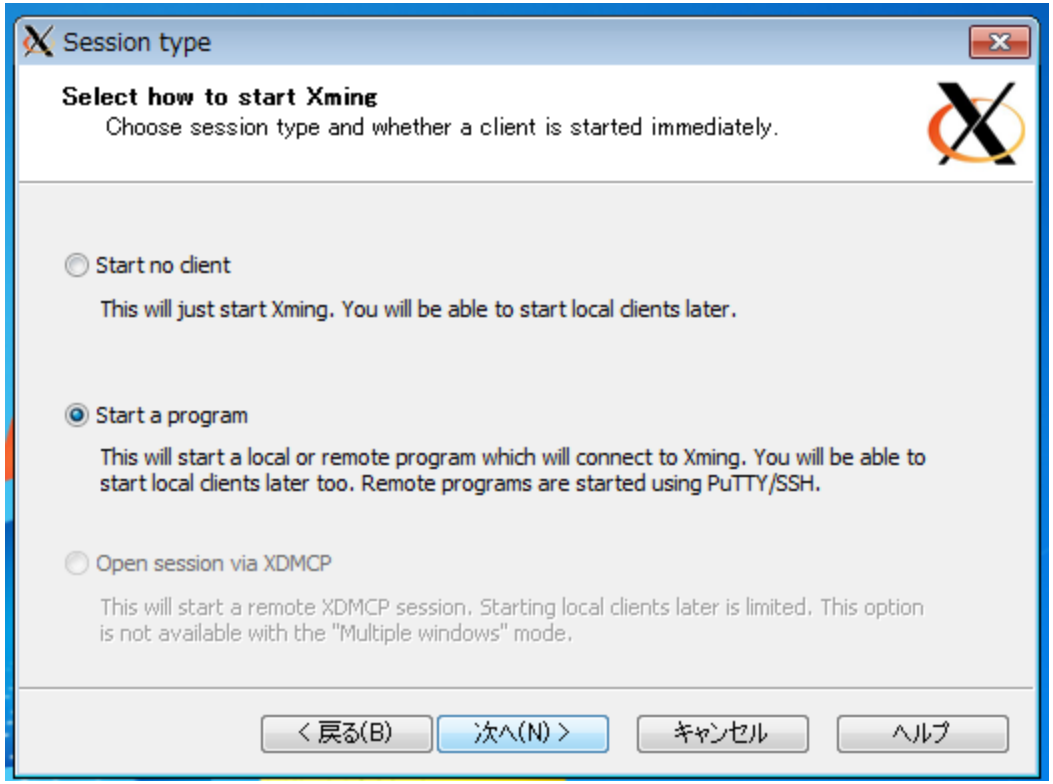


2. 標準的なMultiple windowsを選択する。この選択ではWindows画面を背景にxtermが使用できる。



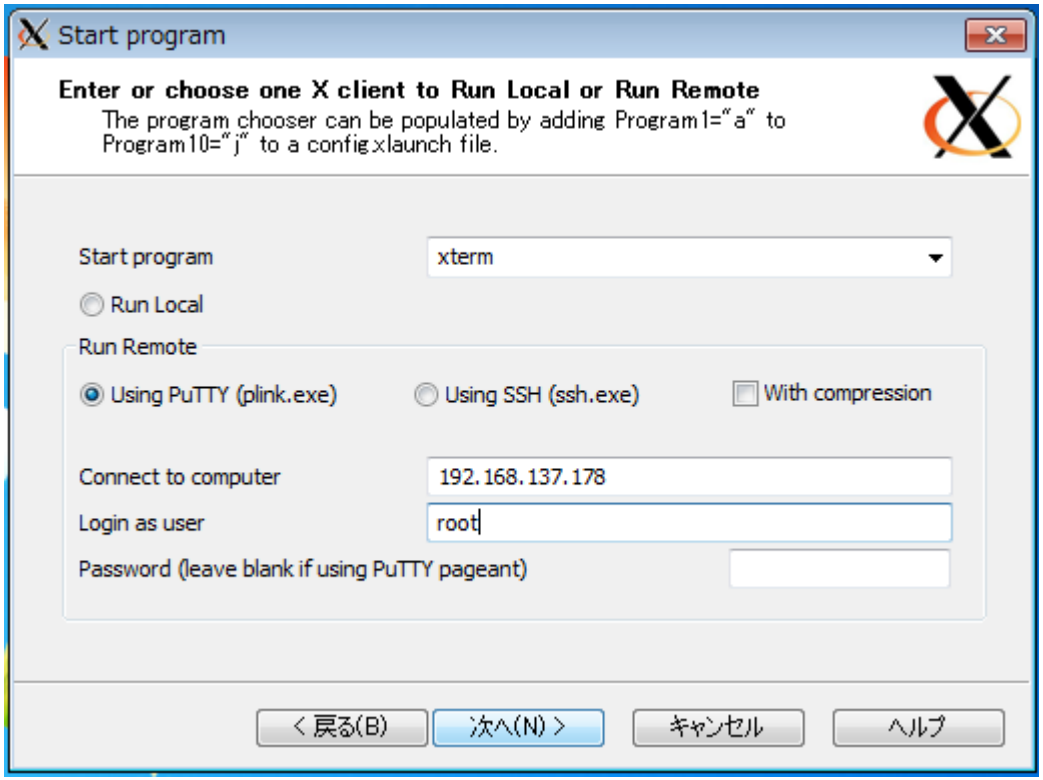
Windowsパソコンでのssh接続（2）

3．Start a program を選択して「次へ」



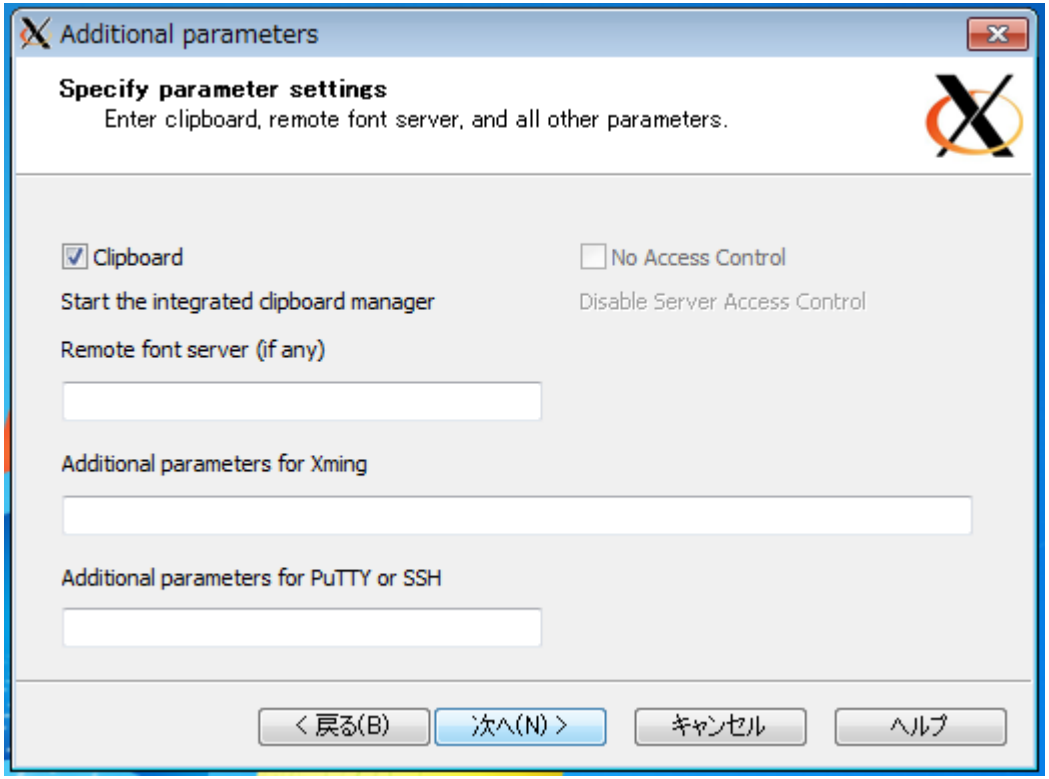
Windowsパソコンでのssh接続（3）

4．Start program に「xterm」、Run Remote を using PuTTYを選択し、
Connection to computer にターゲットボードのIPアドレスここでは192.168.137.178
Login as userは”root”。（xtermはデフォルトでrootユーザでしか起動しない）
Passwordは「ブランク」にして「次へ」をクリックする。



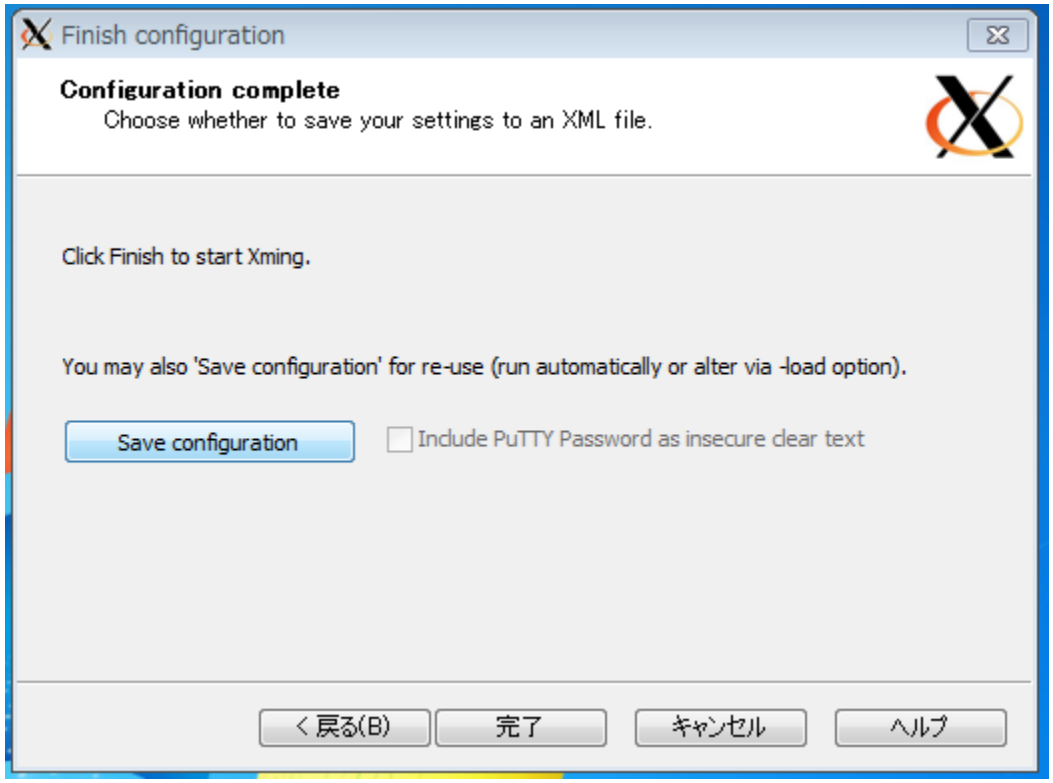
Windowsパソコンでのssh接続（4）

5. 次の画面はそのまま「次へ」



Windowsパソコンでのssh接続（５）

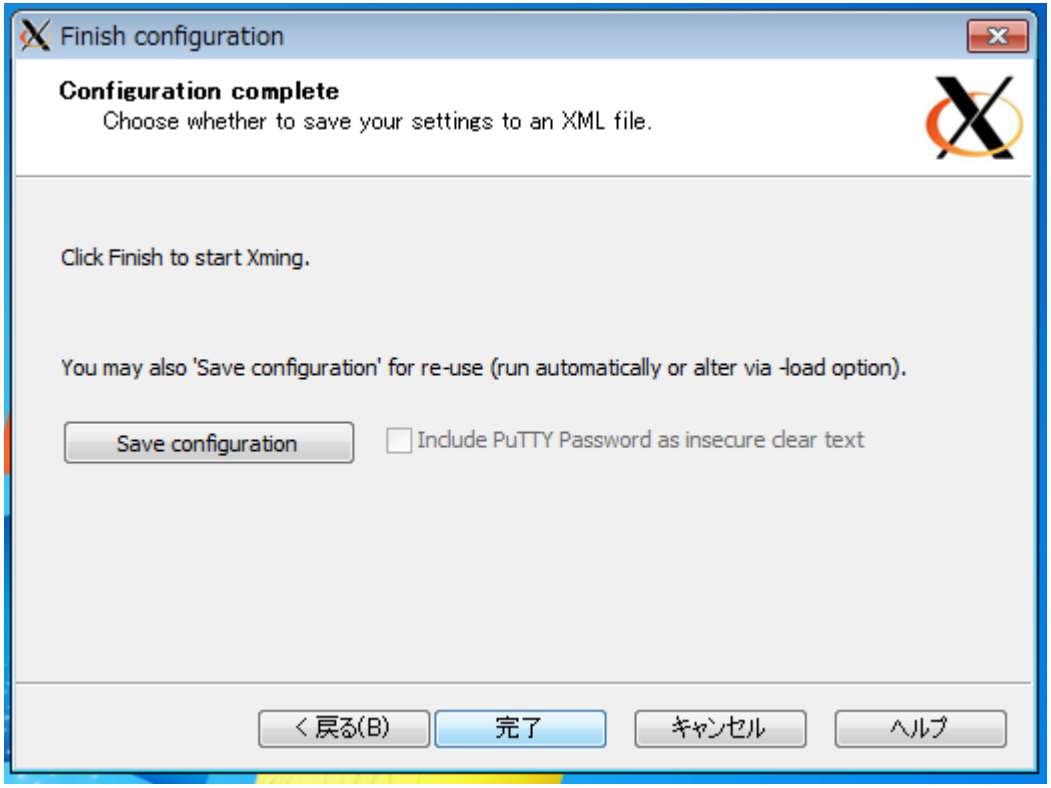
６．Configurerationで「Save configuration」をクリック
　　２回目以降は「完了」クリックでOK



７．config.xlaunchファイルはそのままディフォルトにて「保存」

Windowsパソコンでのssh接続（6）

8.「完了」をクリック



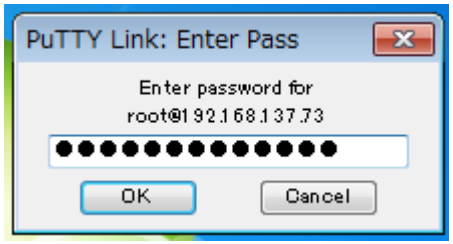
Windowsパソコンでのssh接続（7）

9.「はい」をクリック

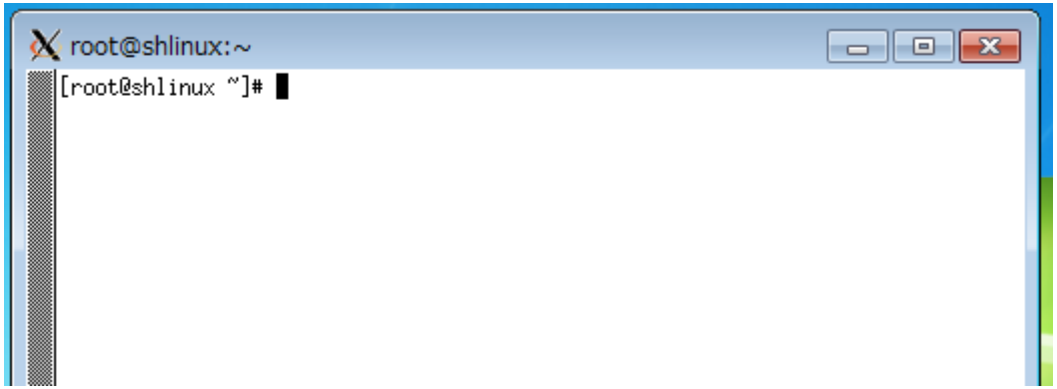
2回目以降は表示されない

Windowsパソコンでのssh接続（8）

10. 結構な時間待たされた後、下記画面が現れるのでrootユーザのパスワード入力する。



11. しばらくするとWindowsディスクトップにxtermが立ち上がる。

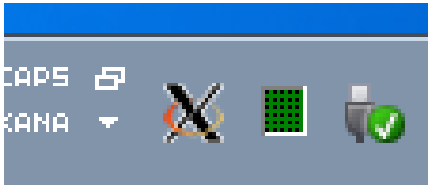


Windowsパソコンでのssh接続（９）

１２．結構な時間待たされ、人知れず本当に接続が切れている場合もある。

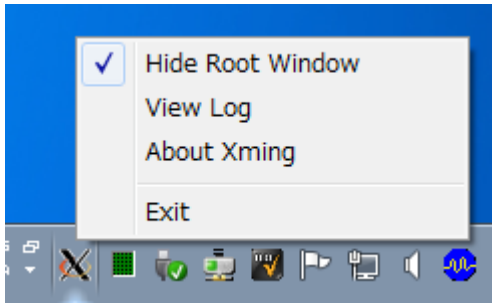
その判断方法としてタスクバーにある以下のアイコン「X」表示で、接続処理が不調に終わると何のメッセージも残さず消えてゆく。

（これでは何が起こったか状況がわからないので、メッセージを知る方法は後述の「Linuxマシンでのssh接続」にてメッセージを確認することを勧める）



Windowsパソコンでのssh接続（10）

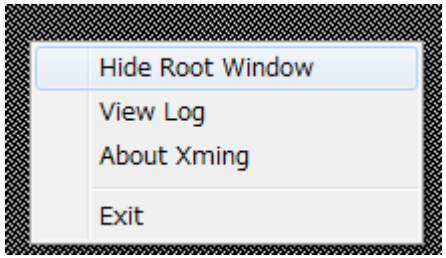
13．アイコンがあると操作したくなるもので、「Hide Root Window」のチェックをはずすと全面　Xウインドウの格子画面と化す。（注：チェックをはずす前に→次の14．に目を通しておくこと。そうでないと戻し方がわからなくなる。）



Unixライクな画面で作業したい人向き用でのxtermが表示できる。

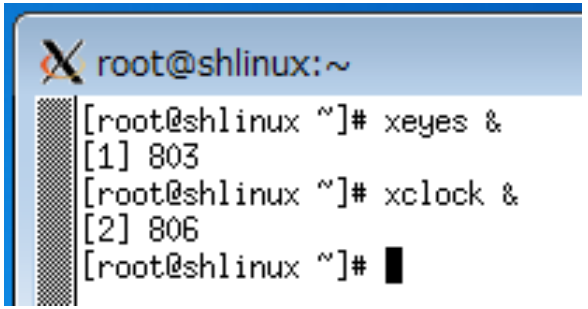
Windowsパソコンでのssh接続（11）

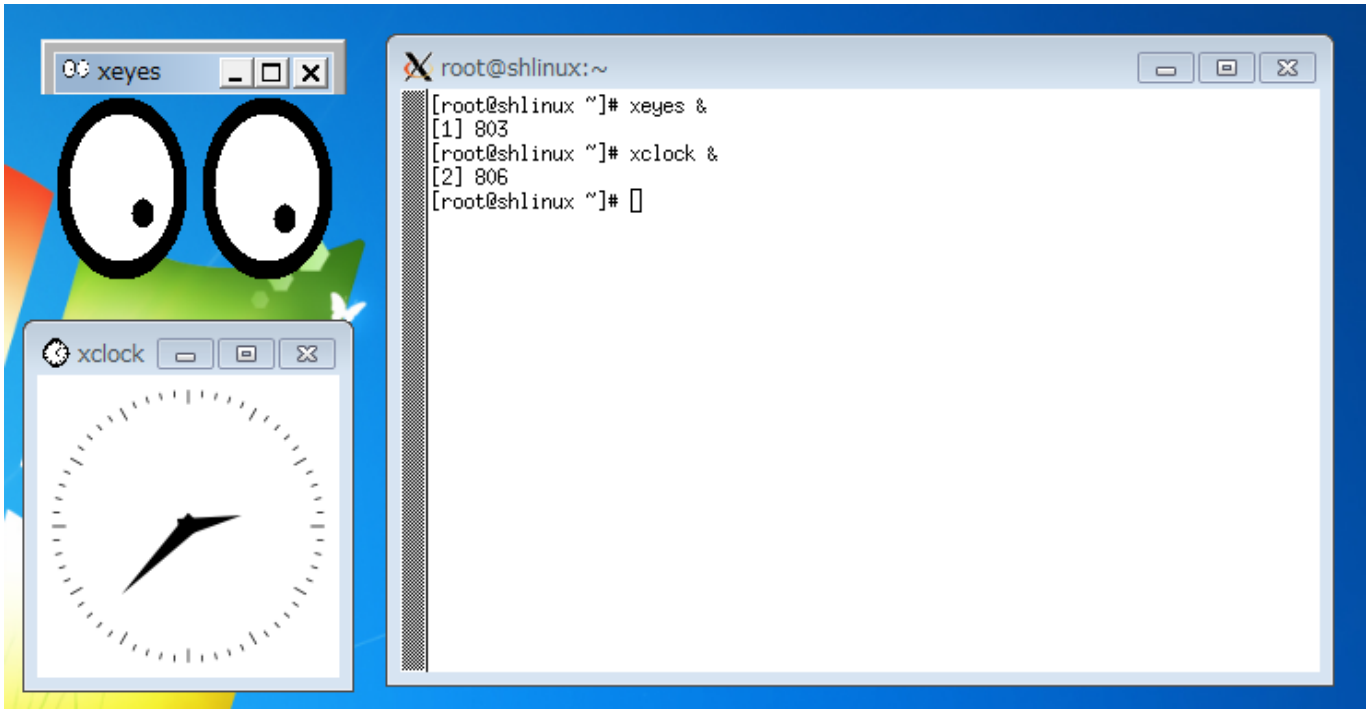
14．Unixになじみのない人には、突然格子模様になるとびっくりするが、元にもどすには[Ctrl]+[Alt]+[Del]をクリックし、「タスクマネージャーの起動」を選択するとタスクバーが現れるので改めて右クリックで下記ポップアップウインドウを開き「Hide Root Window」にチェックを入れればよい。



Windowsパソコンでのssh接続（12）

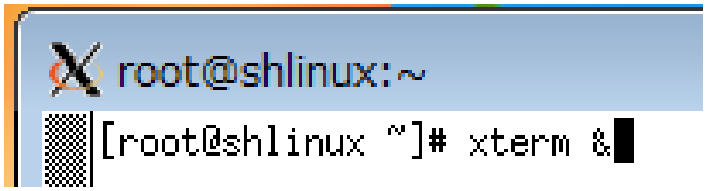
15．xeyesとxclockを起動する。

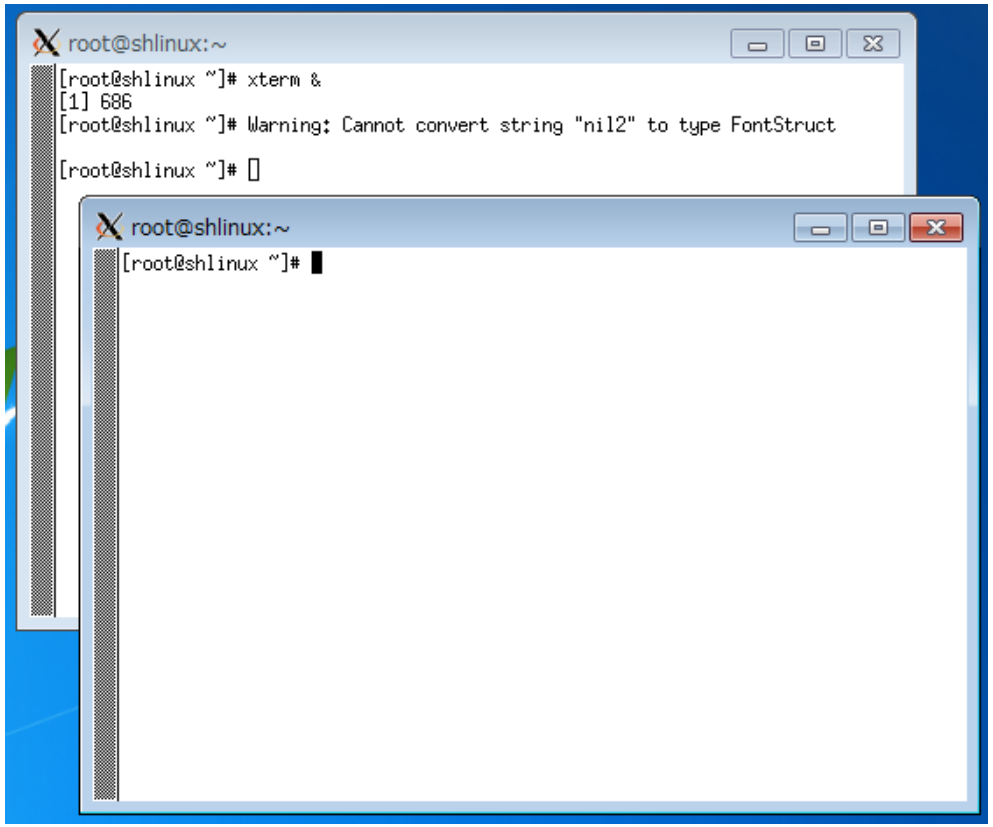




Windowsパソコンでのssh接続（13）

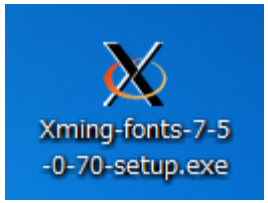
16. 複数のターミナルがほしい場合、xtermを追加起動する。

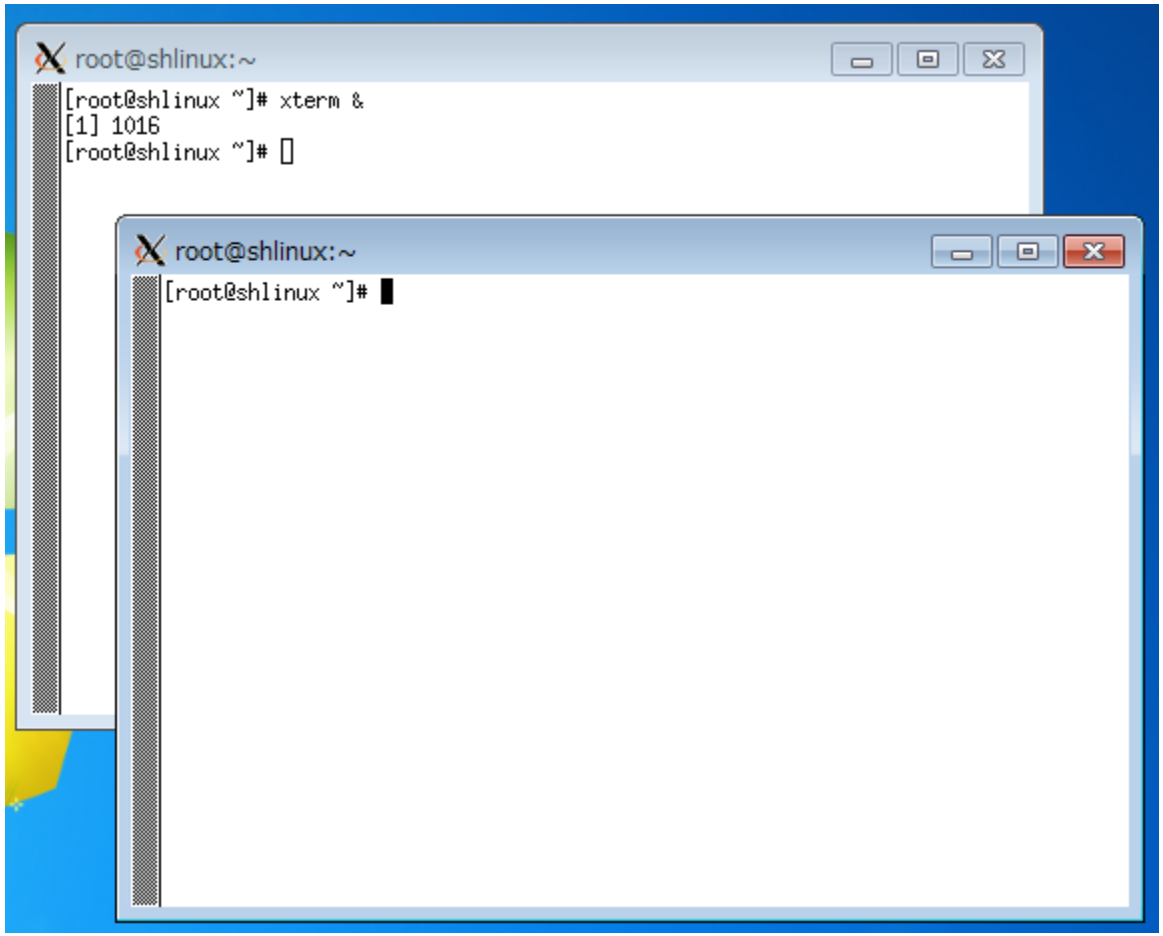




Windowsパソコンでのssh接続（14）

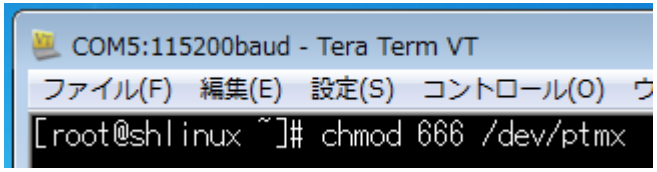
17．# xterm ＆ で…..”nil2“…のワーニングが煩わしいのでXming-fontsをインストールする。

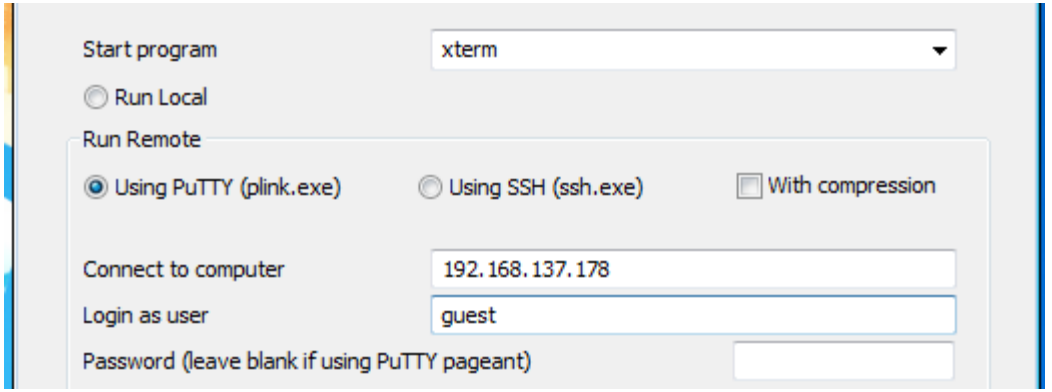




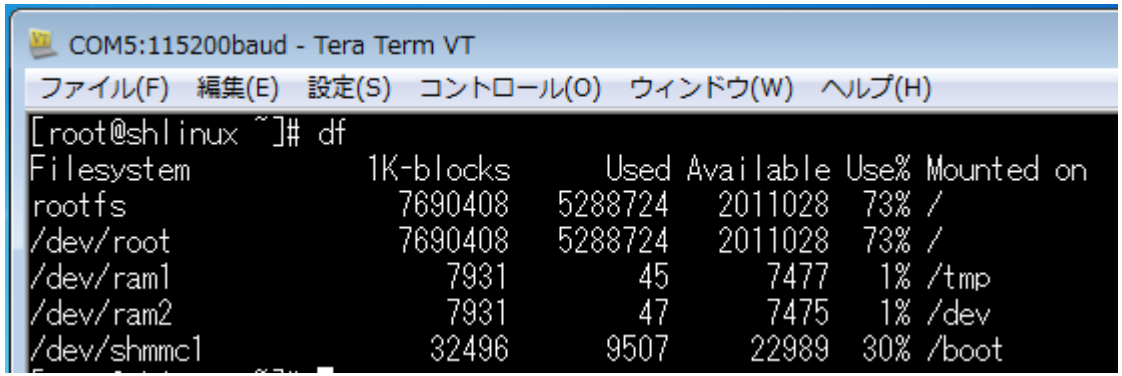
Windowsパソコンでのssh接続（15）

18．一般ユーザのguestとして接続し、xtermを起動させるには、ターゲットの/dev/ptmsのパーミションを666にする。



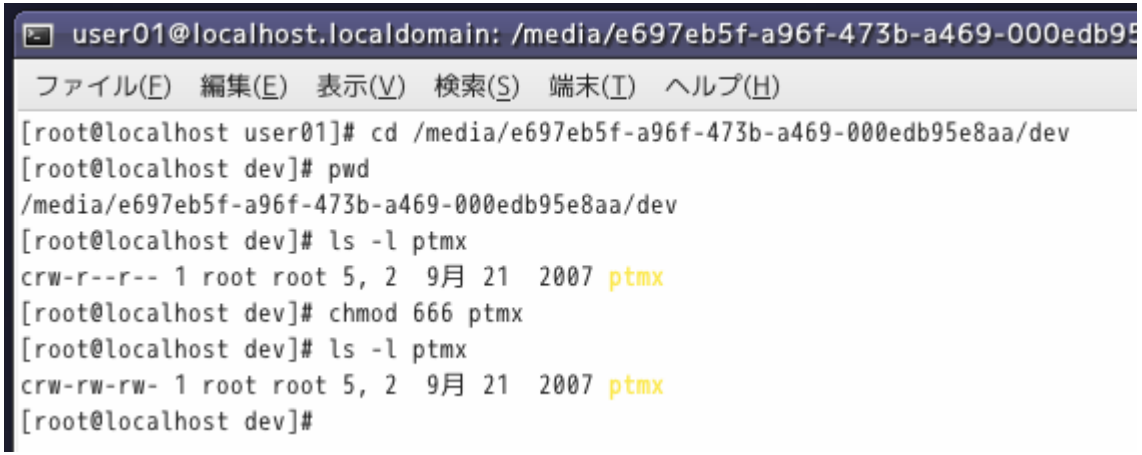


ただし、ターゲット上の/devはram2に展開されているので再起動で元に戻る。



Windowsパソコンでのssh接続（16）

19．SDカードの(SDカード)/dev/ptmsのパーミションを666にすれば再起動時にもパーミションは元に戻らない。



Linuxマシンでのssh接続（1）

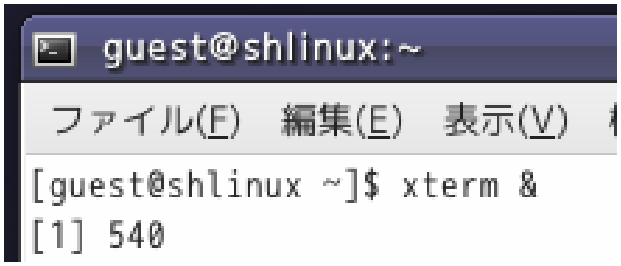
1．Linuxマシン同士なので相性がよく、下記ではターゲットボードのIPアドレスを192.168.137.145としてssh接続する。その時の引数に　-X（大文字”X”)と入れる。。

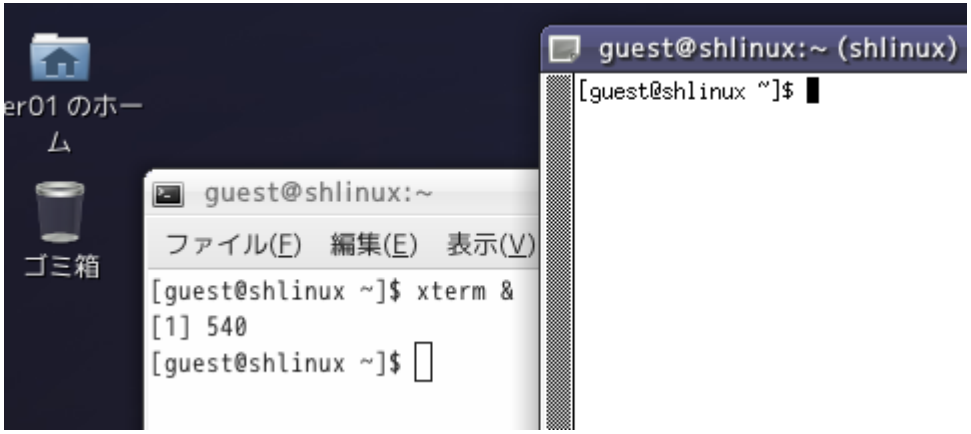
$ ssh -X guest@192.168.137.145

2．パスワード入力後、ターゲットボードコンソールにてxeyesとxclockを起動する。

Linuxマシンでのssh接続（2）

3．複数のターミナルがほしい場合、xtermを追加表示する。



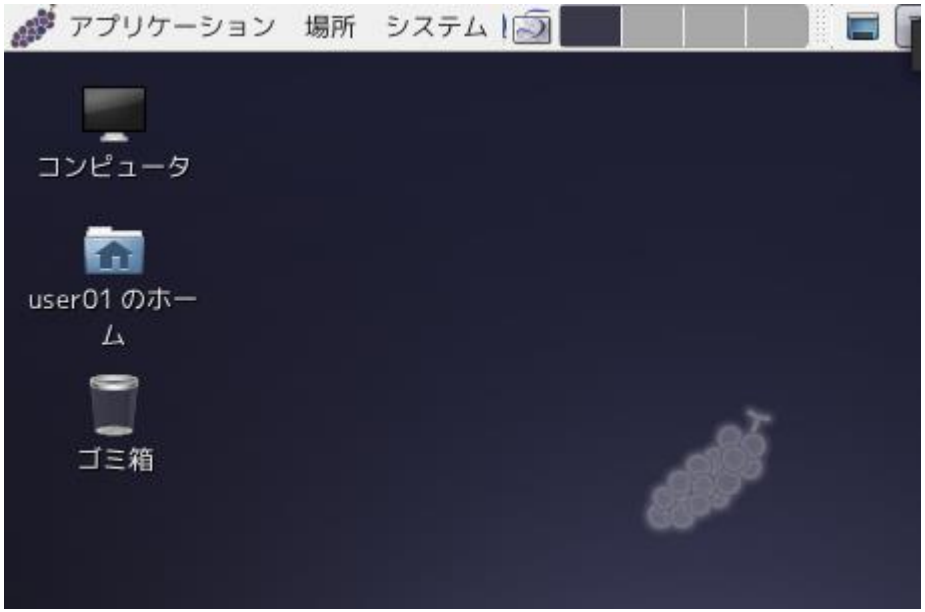


Linuxマシン本体と同一画面なのでターミナルがどちらのものか混乱するので、コマンドを書き込むときにはプロンプトのマシン名の識別に注意する。

クロスコンパイラとセルフコンパイラの生成（1）

1. サイトからダウンロードしたRPMで構築したsh3・Fedora7にはgccが構築されていないのでサイト記事を参考にcrosstoolを使ってATXマシン上にクロスコンパイラと、ターゲットボード上のセルフコンパイラを作成する。

2. まず使用するLinuxOSはVinelinux32ビット（またはCentOSの32ビット）の32ビット版であること。（CPUが64ビットであっても32ビット版でOSあればOK。64ビット版OSでは　コンパイル時にmalloc関数でエラーが出る）



クロスコンパイラとセルフコンパイラの生成（2）

3．サイト記事にしたがい、crosstool-0.43.tar.gzを下記ホームホームページからダウンロードする。

http://kegel.com/crosstool/

4．crosstoolは一般ユーザで使用する。crosstool-0.42.tar.gzを一般ユーザのホームフォルダに展開する。

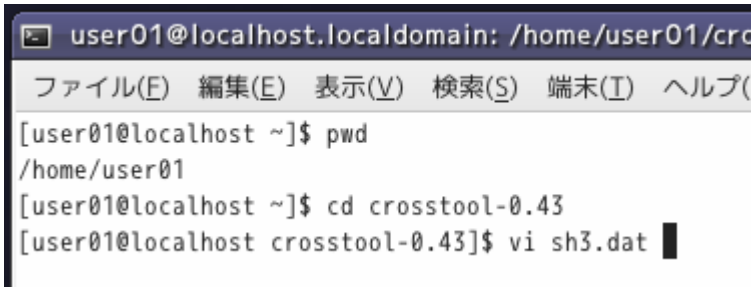
5．crosstoolはクロスコンパイラを構築する時、必要なファイルをネットから取り込みながら構築する。しかし時間を経ると目的のファイルが見つからない場合が生じてくる。ここではあらかじめ必要なファイルを配布データ”downloads”フォルダに収めてあるのでこれを利用する。同フォルダをホームのルートに作成する。

クロスコンパイラとセルフコンパイラの生成（3）

6．サイト記事に従い、必要なファイルは以下の通り、記事と異なるのは使用するkernelバージョンに合わせたlinux-2.6.28.10.tar.bz2（linux-2.6.39.4.tar.bz2は2.6.39.4のとき必要）に差し替えてある。

クロスコンパイラとセルフコンパイラの生成（4）

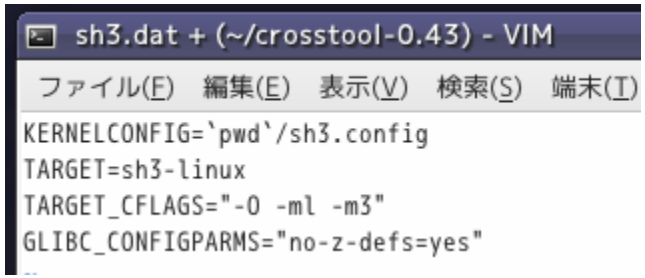
7．crosstool-0.43に入り、sh3.datを編集する。



たとえばクロスコンパイルのgccを “sh3-linux-gcc” といったコマンド表現にするためにはディフォルトのままではなく、

TARGET=sh3-unkown-linux-gnu → sh3-linux

に変更する。



クロスコンパイラとセルフコンパイラの生成（5）

8．続けてcrosstool-0.43のdemo-sh3.shを編集する。

```
user01@localhost.localdomain: /home/user01/crosstoo
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ cd crosstool-0.43
[user01@localhost crosstool-0.43]$ vi demo-sh3.sh
```

コンパイルに必要なファイルはユーザルートの“downloads”フォルダに用意され
生成されるクロスコンパイルはユーザールートの”crosstool”フォルダとするように
指定する。

```
set -ex
TARBALLS_DIR=$HOME/downloads
RESULT_TOP=$HOME/crosstool
export TARBALLS_DIR RESULT_TOP
GCC_LANGUAGES="c,c++"
export GCC_LANGUAGES
```

参照するデータベースは新たな行を追加し、gcc-4.1.2-glibc-2.3.6.datとする。
viコマンドで“yy”→”p”で一行コピー後編集。

```
#eval `cat sh3.dat gcc-4.1.0-glibc-2.3.2-tls.dat` sh all.sh --notest
#eval `cat sh3.dat gcc-4.1.0-glibc-2.3.5.dat` sh all.sh --notest
#eval `cat sh3.dat gcc-4.1.0-glibc-2.3.6.dat` sh all.sh --notest
eval `cat sh3.dat gcc-4.1.2-glibc-2.3.6.dat` sh all.sh --notest
```

クロスコンパイラとセルフコンパイラの生成（6）

9．追加したgcc-4.1.2-glibc-2.3.6.datデータベースファイルを作成する。雛形としてgcc-4.1.1-glibc-2.3.6.datをコピーして作成する。

```
user01@localhost.localdomain: /home/user01/crosstool-0.43
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[user01@localhost crosstool-0.43]$ pwd
/home/user01/crosstool-0.43
[user01@localhost crosstool-0.43]$ cp gcc-4.1.1-glibc-2.3.6.dat gcc-4.1.2-glibc-2.3.6.dat
```

10．gcc-4.1.2-glibc-2.3.6.datを編集する。

```
user01@localhost.localdomain: /home/user01/crosstool-0.43
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[user01@localhost crosstool-0.43]$ vi gcc-4.1.2-glibc-2.3.6.dat
```

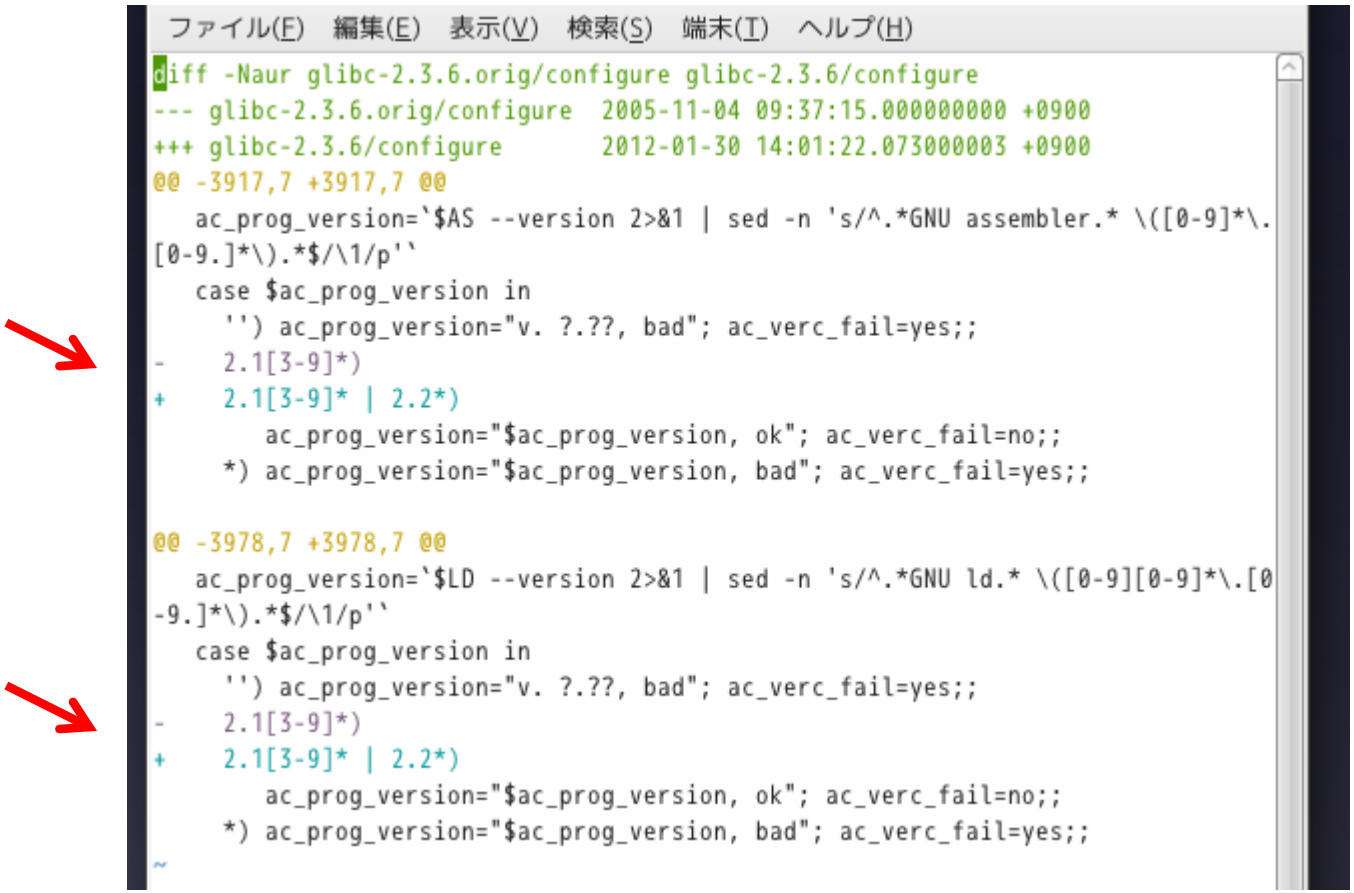
GCC-DIRを4.1.2に替え、LINUX_DIR使用カーネルを2.6.28.10（2.6.39.4の場合はその数字）に替える。その他は”downloads”フォルダにあるファイルのバージョンとの一致を確認する。

```
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
BINUTILS_DIR=binutils-2.16.1
GCC_CORE_DIR=gcc-3.3.6
GCC_DIR=gcc-4.1.2
GLIBC_DIR=glibc-2.3.6
LINUX_DIR=linux-2.6.28.10
LINUX_SANITIZED_HEADER_DIR=linux-libc-headers-2.6.12.0
GLIBCTHREADS_FILENAME=glibc-linuxthreads-2.3.6
GDB_DIR=gdb-6.5
```

クロスコンパイラとセルフコンパイラの生成（7）

11．サイト記事にあるようにglibc-2.3.6には$ASと$LDに対するバグがあり、これに対するパッチが必要で公開されている。ここでは適当な名前で ”glibc-2.3.6-as-ld.patch” ファイルとしてサイトの情報をそっくりコピーして配布資料に収録している。

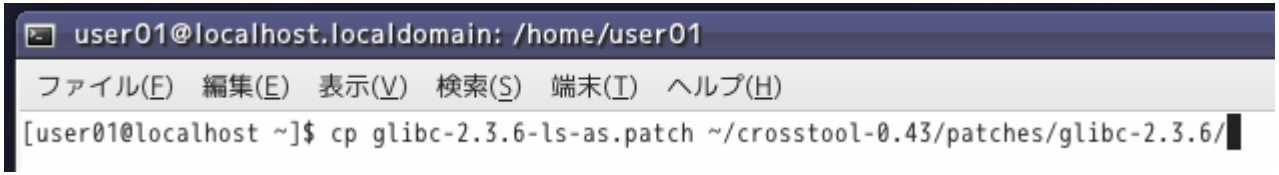
```
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[user01@localhost ~]$ vi glibc-2.3.6-ls-as.patch
```



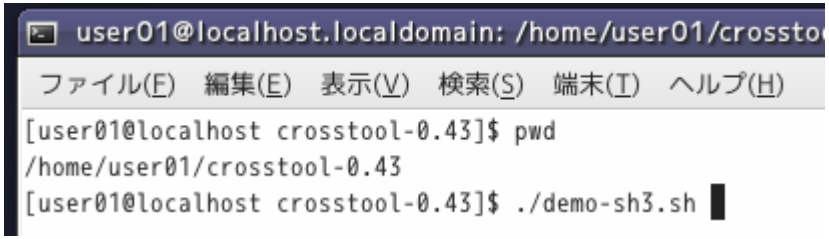
```
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
diff -Naur glibc-2.3.6.orig/configure glibc-2.3.6/configure
--- glibc-2.3.6.orig/configure  2005-11-04 09:37:15.000000000 +0900
+++ glibc-2.3.6/configure       2012-01-30 14:01:22.073000003 +0900
@@ -3917,7 +3917,7 @@
   ac_prog_version=`$AS --version 2>&1 | sed -n 's/^.*GNU assembler.* \([0-9]*\.
[0-9.]*\).*$/\1/p'`
   case $ac_prog_version in
     '') ac_prog_version="v. ?.??, bad"; ac_verc_fail=yes;;
-    2.1[3-9]*)
+    2.1[3-9]* | 2.2*)
        ac_prog_version="$ac_prog_version, ok"; ac_verc_fail=no;;
     *) ac_prog_version="$ac_prog_version, bad"; ac_verc_fail=yes;;

@@ -3978,7 +3978,7 @@
   ac_prog_version=`$LD --version 2>&1 | sed -n 's/^.*GNU ld.* \([0-9][0-9]*\.[0
-9.]*\).*$/\1/p'`
   case $ac_prog_version in
     '') ac_prog_version="v. ?.??, bad"; ac_verc_fail=yes;;
-    2.1[3-9]*)
+    2.1[3-9]* | 2.2*)
        ac_prog_version="$ac_prog_version, ok"; ac_verc_fail=no;;
     *) ac_prog_version="$ac_prog_version, bad"; ac_verc_fail=yes;;
~
```

クロスコンパイラとセルフコンパイラの生成（8）

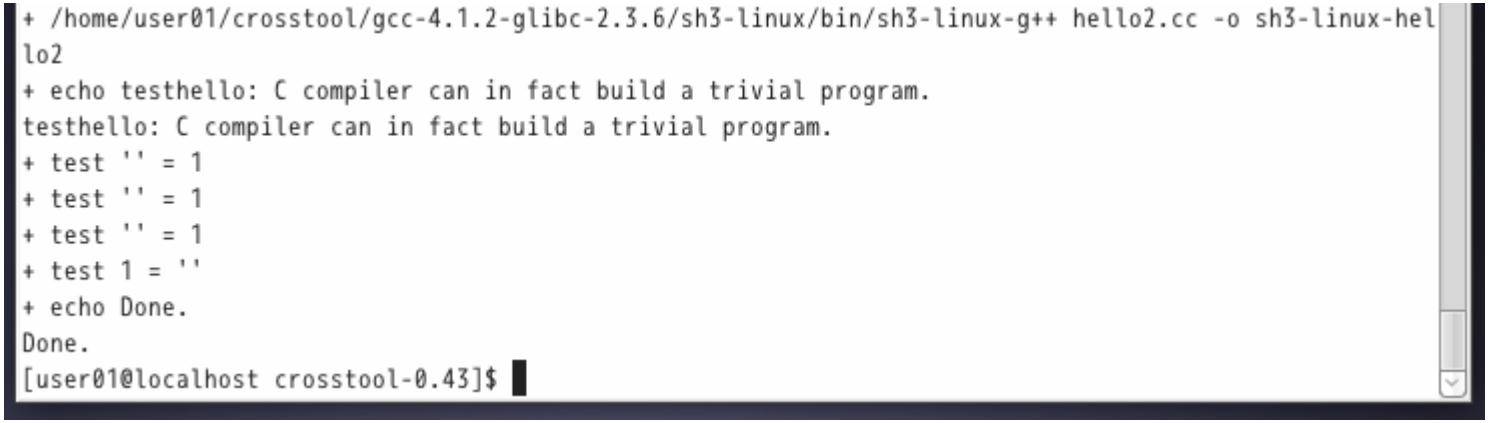
12．作成した”glibc-2.3.6-as-ld.patch”パッチファイルは、/crosstool-0.43/ patches /glibc-2.3.6/にコピーまたは移動する。



13．$ ./demo-sh.shにてクロスコンパイラの作成をスタート。



14．クロスコンパイラ作成の終了画面（約20分）



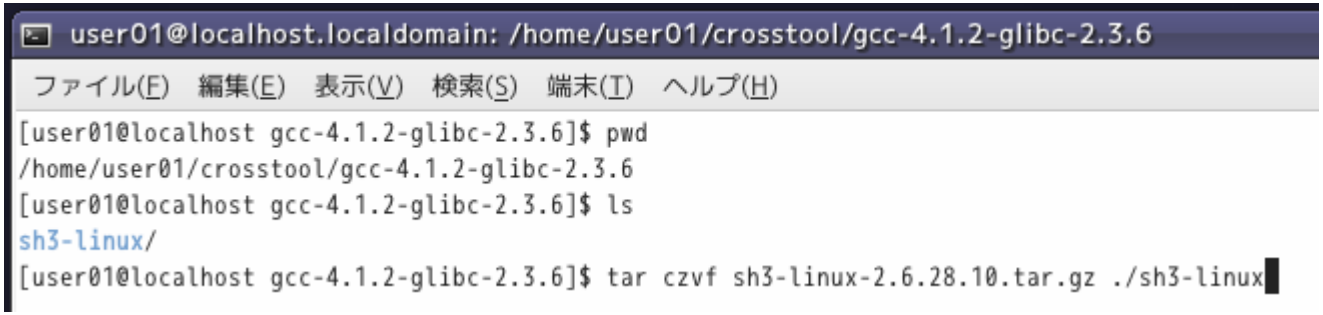
クロスコンパイラとセルフコンパイラの生成（9）

15. クロスコンパイラは　demo-sh3.shで設定したように　~/crosstool　フォルダ配下に構築されている。

クロスコンパイラとセルフコンパイラの生成（10）

16. 完成したクロスコンパイラの圧縮ファイルを作成する。

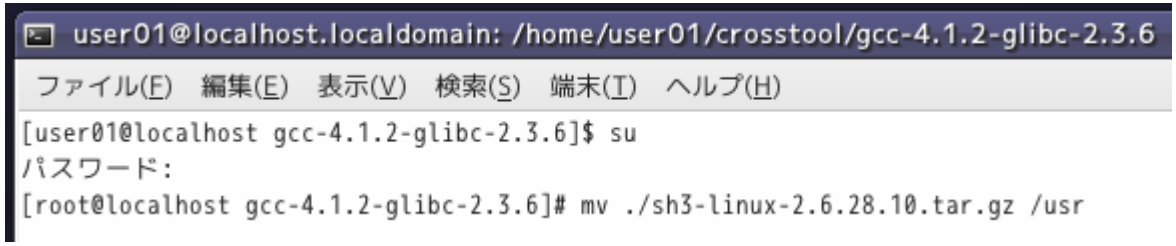
$ tar czvf （圧縮ファイル名）.gz ./sh-linux



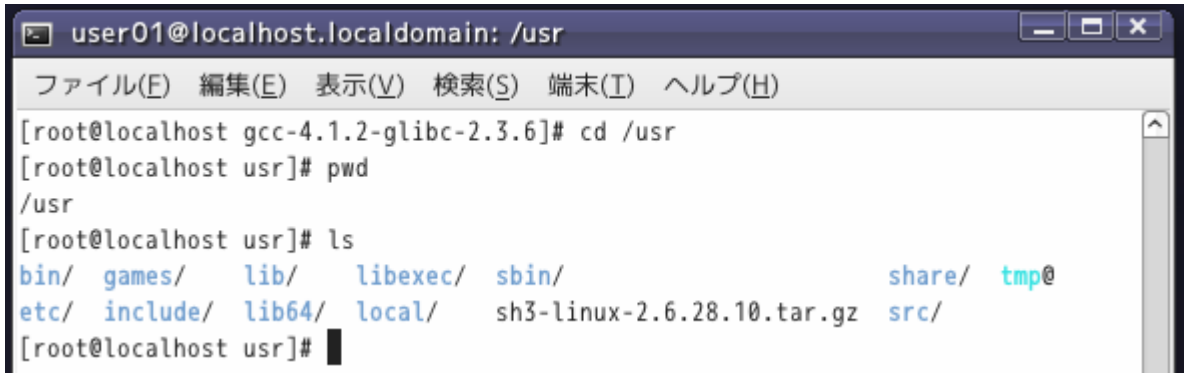
17. 作成したsh3-linux-2.6.28.10.tar.gzは別のLinuxマシンでクロスコンパイラ環境を作るのにも使用できるので配布資料の中にも納めている。

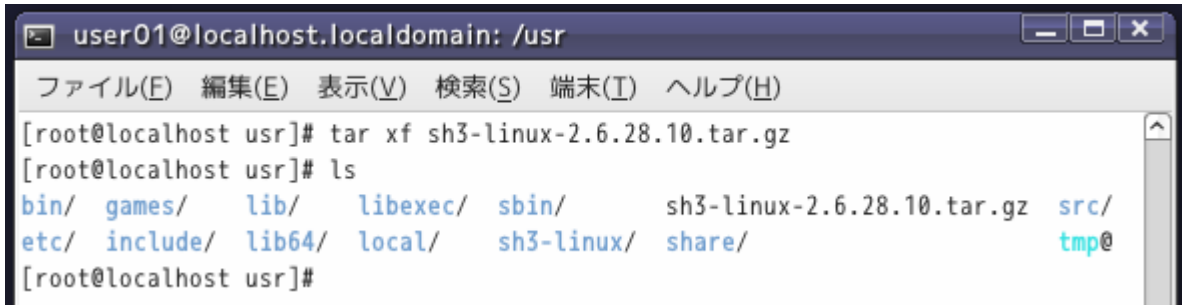
クロスコンパイラとセルフコンパイラの生成（11）

18．完成したクロスコンパイラをインストールする。ルートユーザとなって圧縮ファイルを/usrに移動する。



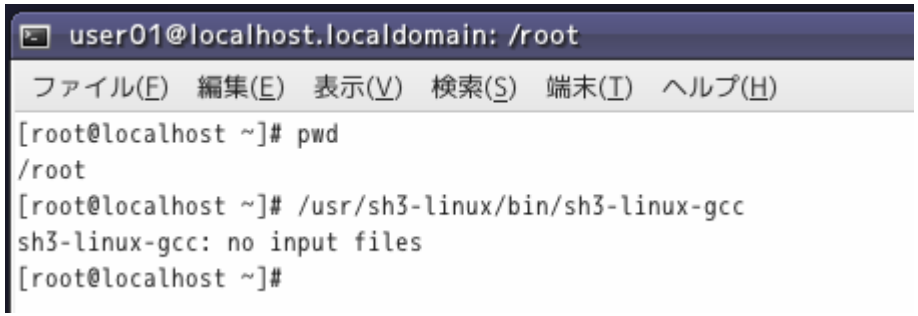
19．/usrに移り、圧縮ファイルを解凍する。（圧縮ファイルは削除してOK）





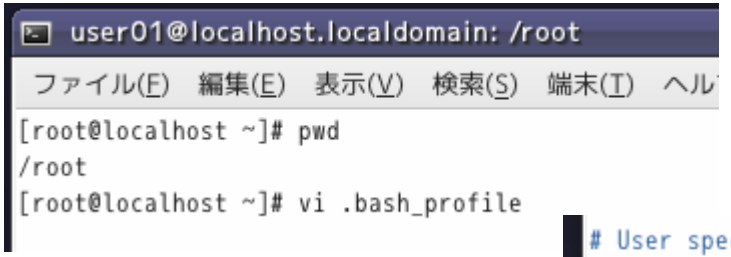
クロスコンパイラとセルフコンパイラの生成（12）

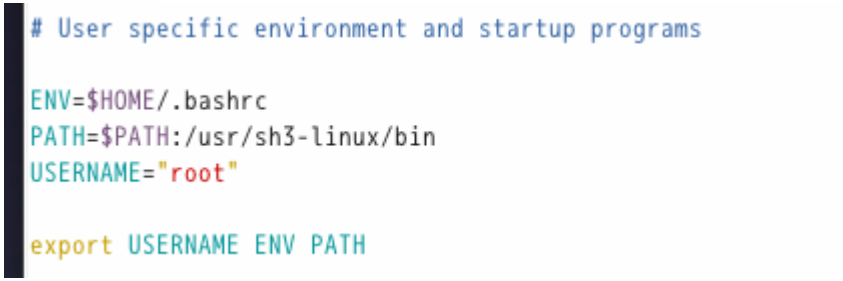
20. 以降rootで使用することを前提にホームに戻り、フルパスでgccを起動する。



“no input files”といったメッセージが返っているので反応している。

21. 相対パスが使用できるように　.bash_profileを編集する。再起動で設定される。





クロスコンパイラとセルフコンパイラの生成（13）

22．一般ユーザで環境を引き継ぐことも考え、user01のPATH環境も変更する。

$ vi　.bash_profile

23．Rootユーザ同様に相対パスが使用できるように　.bash_profileを編集する。再起動で設定される。

```
# User specific environment and startup programs

# addpath $HOME/bin
BASH_ENV=$HOME/.bashrc
PATH=$PATH:/usr/sh3-linux/bin
USERNAME=""

export USERNAME BASH_ENV PATH LESSOPEN
```

24．再起動させる。

クロスコンパイラとセルフコンパイラの生成（14）

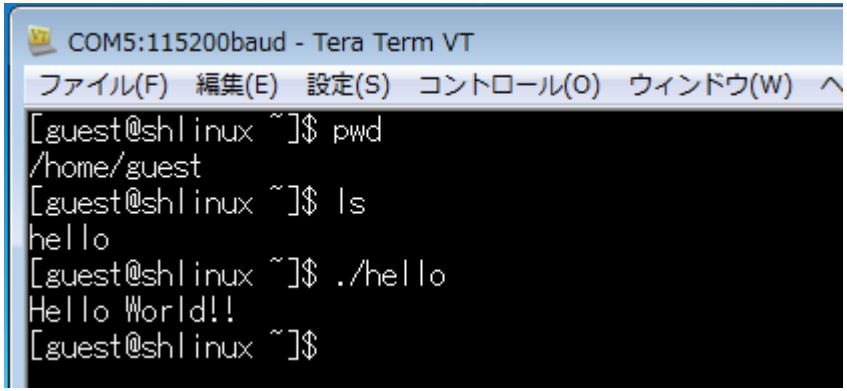
25. 再起動後、以下のソースファイルを作成する。どこで作業してもよいがここでは./sampleフォルダで作業する。

26. コンパイルする。（ライブラリが異なるので-staticオプションを付加する）

クロスコンパイラとセルフコンパイラの生成（15）

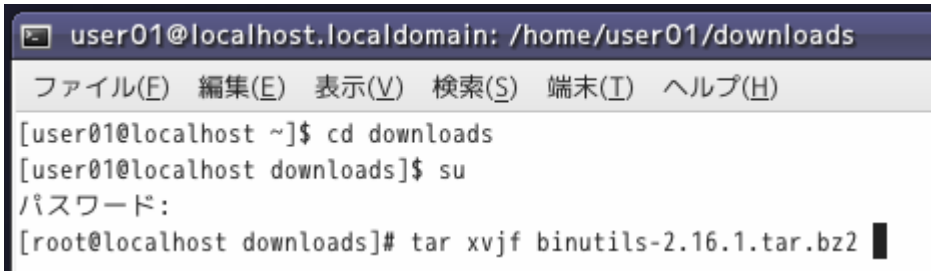
27．少し手間だが、SDカードのguestのルートフォルダにコピーする。

28．ターゲットにhelloファイルをコピーしたSDカードにて起動し、ターミナルを通してhelloを実行する。

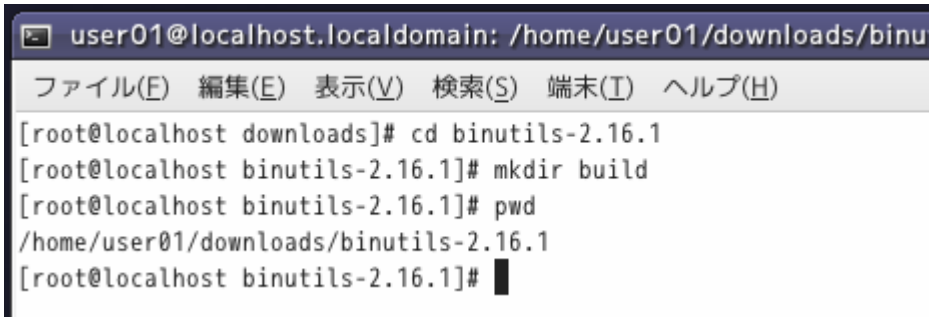


クロスコンパイラとセルフコンパイラの生成（16）

29．続いてセルフコンパイル環境を構築する。記事と異なるのは直接SDカードのルートフォルダの/usrフォルダにコンパイラを書き込むこととrootユーザにて処理をすすめることである。サイト記事にしたがい“downloads”フォルダ内で”binutils-2.16.1.tar.bz2”を展開する。



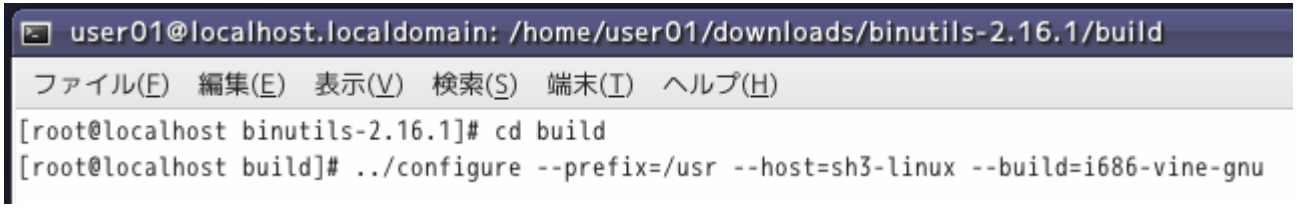
30．展開後、binutils-2.16.1に入り、buildフォルダを作成し、buildフォルダに移動する。



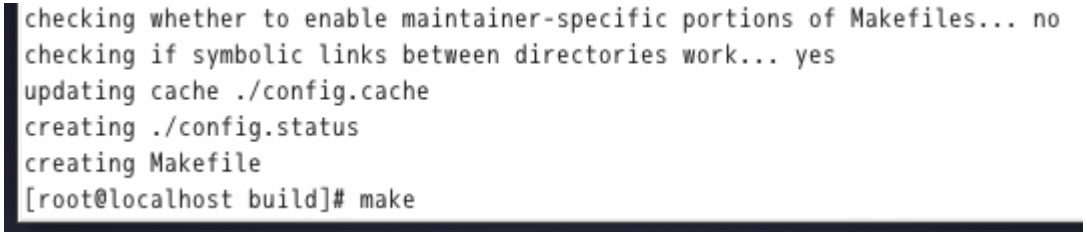
クロスコンパイラとセルフコンパイラの生成（17）

31. buildに移動後、configureにてMakefileを作成する。対象フォルダは/usrに設定。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu

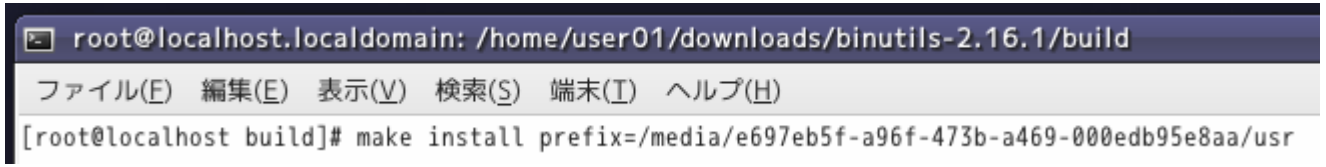


引き続き # make を実行



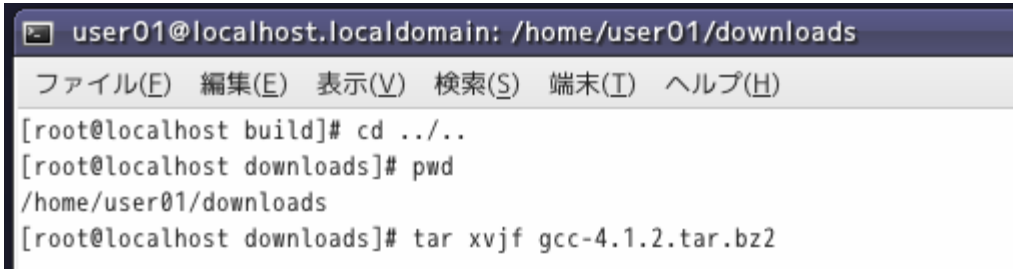
32. インストール先は、直接マイコンボードの/usr、すなわちSDカードの/usrとする。

make install prefix=/media/(SDカードのルート)/usr



クロスコンパイラとセルフコンパイラの生成（18）

33．引き続き同様に“downloads”フォルダ内で”gcc-4.1.2.tar.bz2”を展開する。

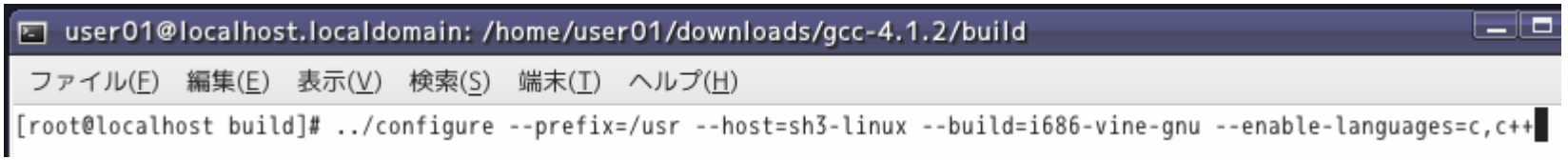


34．展開後、gcc-4.1.2に入り、buildフォルダを作成し、buildフォルダに移動する。

クロスコンパイラとセルフコンパイラの生成（19）

35．buildに入った後、configureにてMakefileを作成する。対象フォルダを/usrに設定。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu --enable-languages=c,c++

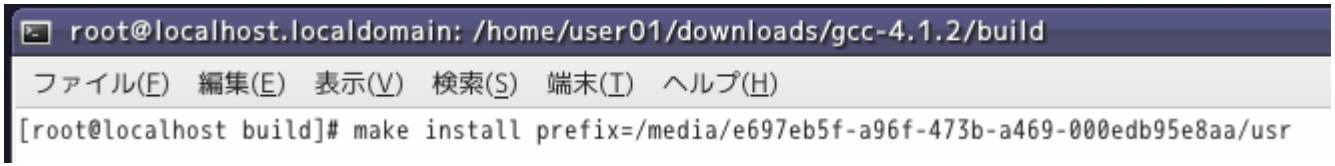


引き続き # make する。（約5分）

```
checking whether to enable maintainer-specific portions of Makefiles... no
checking if symbolic links between directories work... yes
updating cache ./config.cache
creating ./config.status
creating Makefile
[root@localhost build]# make
```

36．Web記事にあるようにコンパイラは失敗しているが、インストール先はマイコンボードの/usrなのでSDカードの/usrにインストールする。

make install prefix=/media/(SDカードのルート)/usr



クロスコンパイラとセルフコンパイラの生成（20）

37．コンパイルエラーのため、インストールで記事と同様のエラーメッセージが出る。

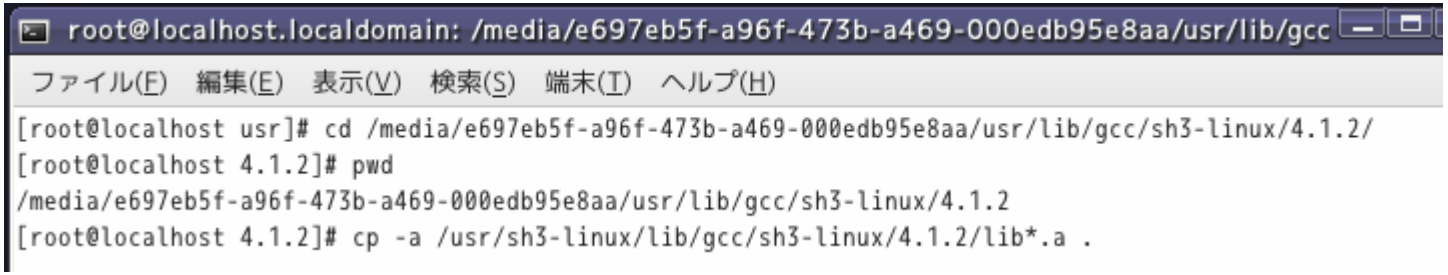
```
/usr/bin/install: 宛先の `/media/8afb8188-3c71-45a7-814c-319318cefe0f/usr/libexec/g
stalldirs' はディレクトリではありません
/usr/bin/install: `build/fix-header' を stat できません: そのようなファイルやディレ
make[2]: *** [install-mkheaders] Error 1
make[2]: Leaving directory `/home/user01/downloads/gcc-4.1.2/build/gcc'
make[1]: *** [install-gcc] Error 2
make[1]: Leaving directory `/home/user01/downloads/gcc-4.1.2/build'
make: *** [install] Error 2
[root@localhost build]#
```

38．Web記事にしたがい、SDカードの/usrに移動し、クロスコンパイラから　/include　フォルダを持ってくる。

```
root@localhost.localdomain: /media/e697eb5f-a96f-473b-a469-000edb95e8
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# cd /media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr
[root@localhost usr]# pwd
/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr
[root@localhost usr]# cp -a /usr/sh3-linux/sh3-linux/include .
```

クロスコンパイラとセルフコンパイラの生成（21）

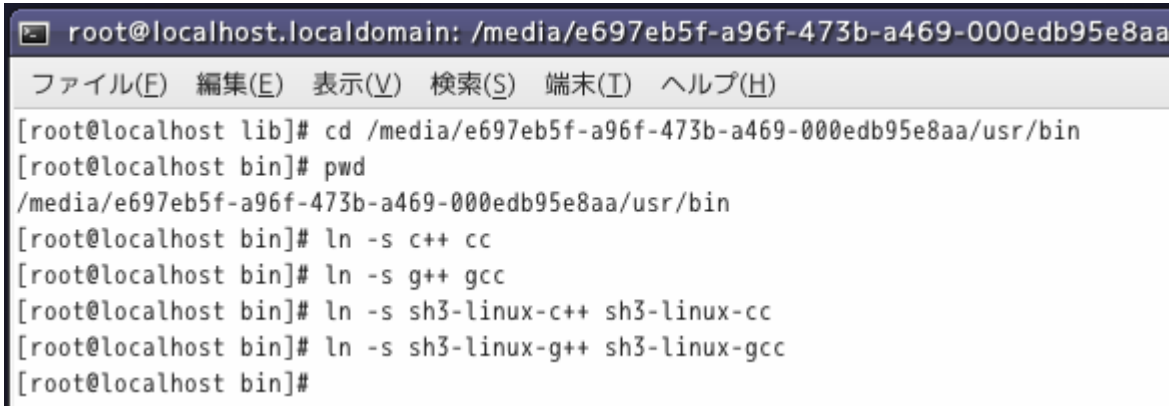
39. SDカードの/usr/lib/gcc/sh3-linux/4.1.2に移動して、lib*.aライブラリファイルをクロスコンパイラから持ってくる。



40. glibcの構築にてCコンパイラ構築の失敗しているので、クロスコンパイラから/usr/libへもってくる。

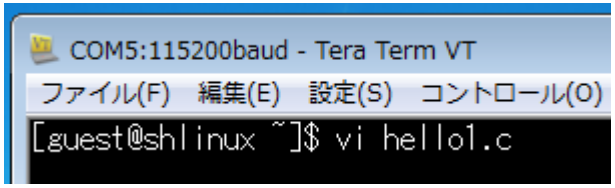
クロスコンパイラとセルフコンパイラの生成（22）

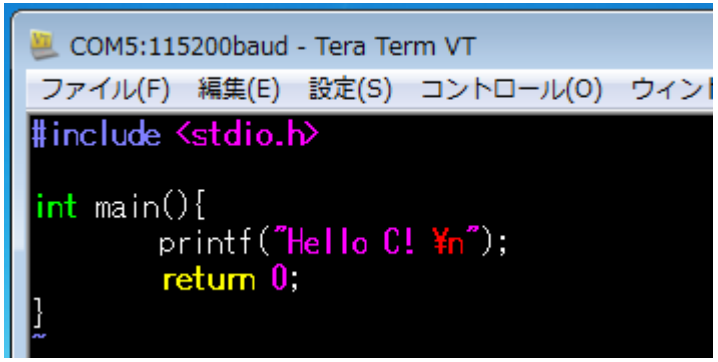
41. コンパイラのコマンドを整理する。



42. SDカードをターゲットボードに差し、起動する。

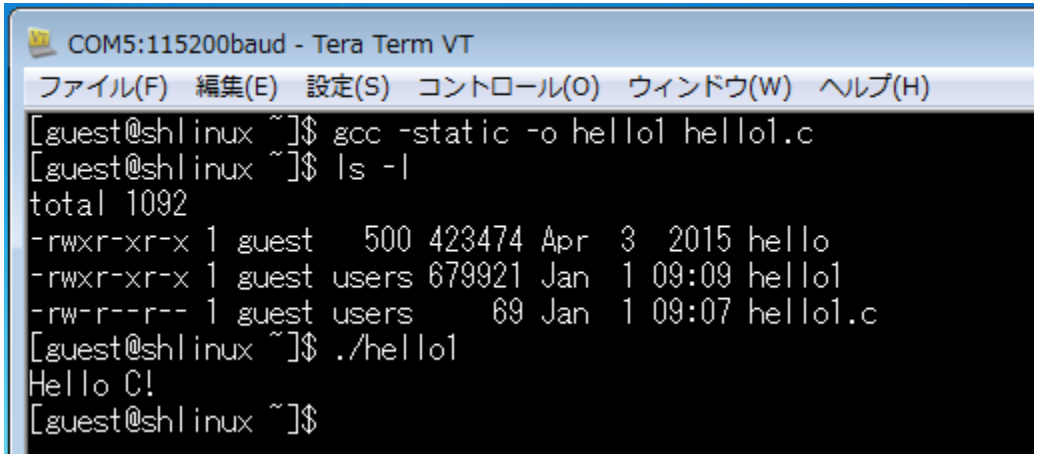
guestでログインしたあと、以下の　hello1.c のソースコードを作成する。



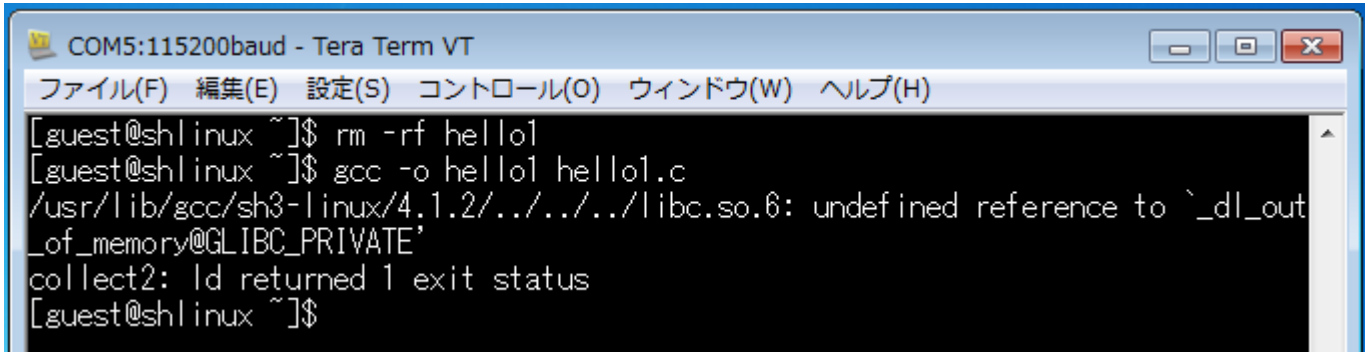


クロスコンパイラとセルフコンパイラの生成（23）

43. まずは、-static　オプション付にてコンパイルする。実行ファイルはhello1
コンパイルに時間がかかるが、実行ファイルが完成している。
実行ファイルの大きさはおよそ680Kバイトもある。

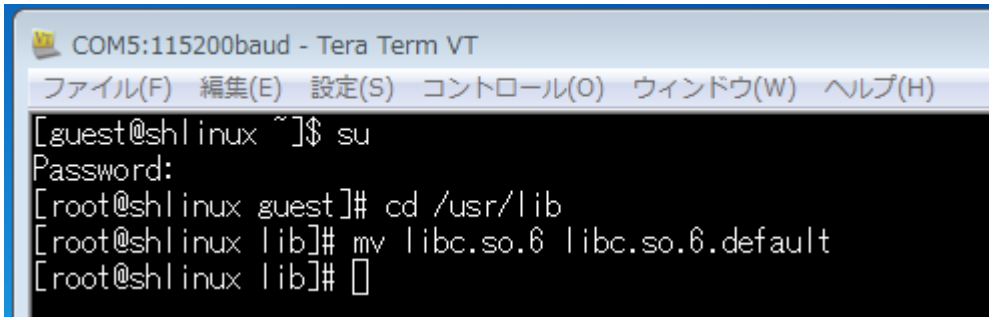


44. 続いて –static オプションをはずしてライブラリを使用するコンパイルをおこなうと
定義されていない変数があるようで、コンパイルは通らない。

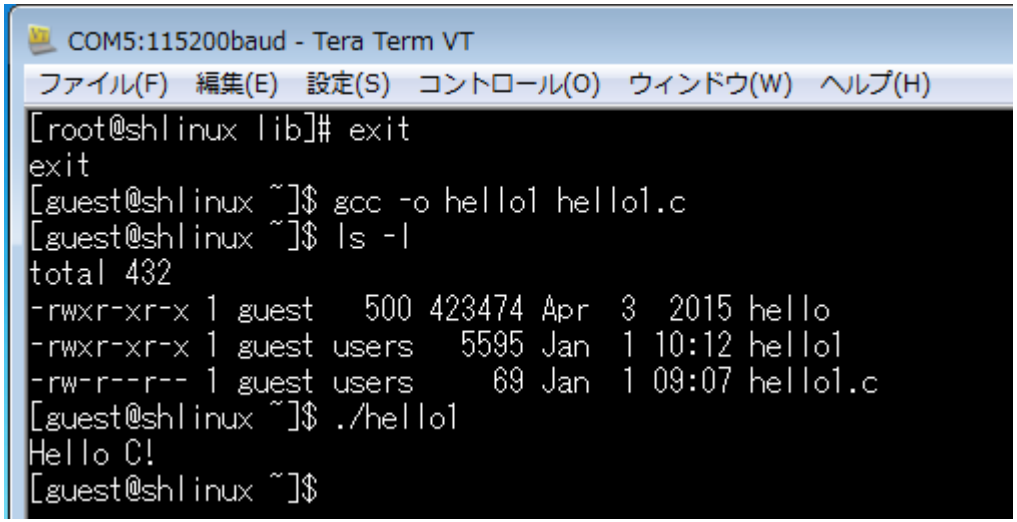


クロスコンパイラとセルフコンパイラの生成（24）

45．クロスコンパイラが使用するCライブラリとRPMデータから構築したCライブラリとに違いがあるようで、クロスコンパイラのlibc.so.6を参照しないように別の名前に差し替える。

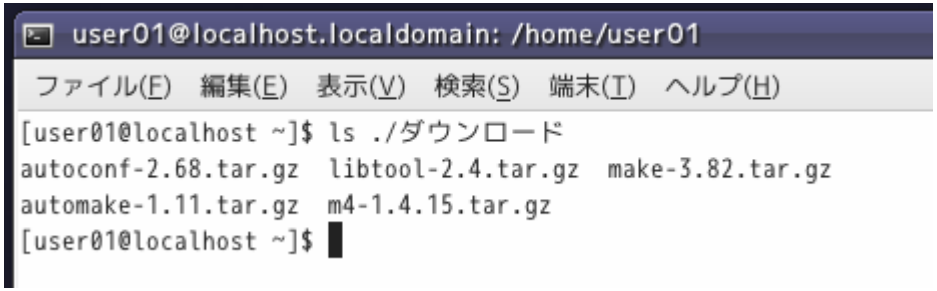


46．guestユーザに戻って、再度コンパイルする。/lib/libc.so.6のライブラリが使用され今度は実行ファイルが生成される。ファイルサイズは約5．6Kバイトとなる。



GNU標準パッケージの移植（1）

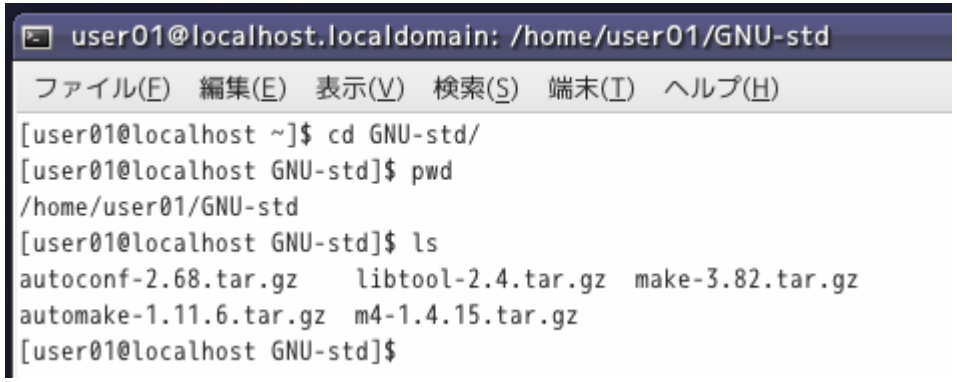
1. サイト記事に沿ってGNU標準パッケージを移植する。現行状態ではターゲットボード上処理で./configureが使えない。サイト記事に従いインターネットなどから以下のファイルをダウンロードする。セミナーでは配布資料に「GNU標準パッケージ」として保存しているのでこれを使用する。



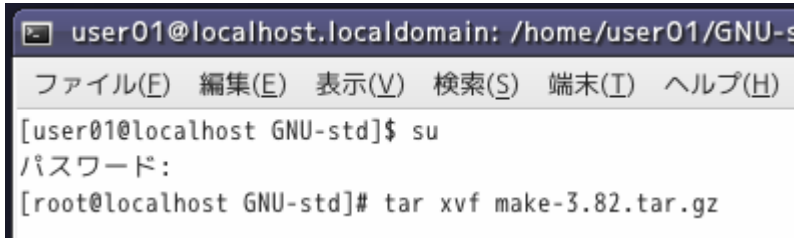
2. 作業フォルダを作り、この中で作業をおこなう。ここでは作業フォルダ名を“GNU-std”とする。

GNU標準パッケージの移植（2）

3. Web記事のままではなく、ここではインストール先を直接SDカードにする。もう一度フォルダ内の様子を示す。



4. SDカードへの書込みはrootユーザでしかできないのでrootユーザーになる。
最初にmakeのクロスコンパイルを行う。tarにてmake-*****.gzファイルを展開する。



GNU標準パッケージの移植（3）

5．展開したフォルダに入り、buildフォルダを作成し、さらにbuildに移動する。

```
user01@localhost.localdomain: /home/user01/GNU-st
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost GNU-std]# cd make-3.82
[root@localhost make-3.82]# mkdir build
[root@localhost make-3.82]# cd build
[root@localhost build]# pwd
/home/user01/GNU-std/make-3.82/build
[root@localhost build]#
```

6．Web記事に従いConfigureコマンドでMaikefileを作成する。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu --disable-nls

```
user01@localhost.localdomain: /home/user01/GNU-std/make-3.82/build
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# pwd
/home/user01/GNU-std/make-3.82/build
[root@localhost build]# ../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu --disable-nls
```

```
config.status: executing default-1 commands
config.status: creating po/POTFILES
config.status: creating po/Makefile
config.status: creating build.sh
[root@localhost build]#
```

GNU標準パッケージの移植（4）

7. make する。

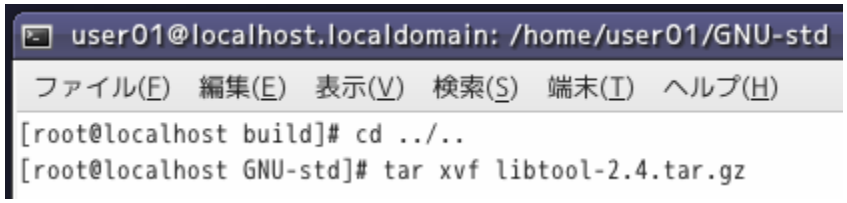
　# make

8. SDカードのルートフォルダの/usrにインストールする。

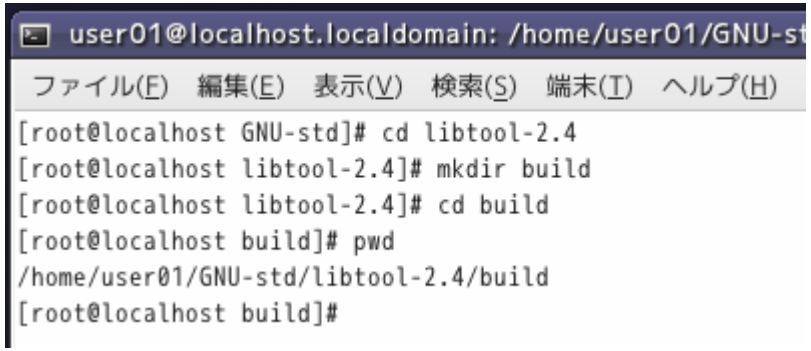
　#make install prefix=/media/(SDカードのルートフォルダ)/usr

GNU標準パッケージの移植（５）

９．続けてlibtoolのクロスコンパイルを行う。tarにてlibtool-*****.gzを展開する。



１０．展開したフォルダにbuildフォルダを作成し、buildに移動する。



GNU標準パッケージの移植（6）

11．ConfigureコマンドでMaikefileを作成する。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu

```
user01@localhost.localdomain: /home/user01/GNU-std/libtool-2.4/build
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# pwd
/home/user01/GNU-std/libtool-2.4/build
[root@localhost build]# ../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu
```

```
config.status: creating config.h
config.status: executing tests/atconfig commands
config.status: executing depfiles commands
config.status: executing libtool commands
[root@localhost build]#
```

12．make する。

make

```
user01@localhost.localdomain: /home
ファイル(F)  編集(E)  表示(V)  検索(S)  端末
[root@localhost build]# make
```

```
libtool: link: ( cd "libltdl/.libs" && rm -f "libltdl.la" && ln -s "../libltdl.la" "libltdl.la" )
make[2]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
make[1]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
[root@localhost build]#
```

GNU標準パッケージの移植（7）

13．SDカードのルートフォルダの/usrにインストールする。

#make install prefix=/media/(SDカードのルートフォルダ)/usr

```
root@localhost.localdomain: /home/user01/GNU-std/libtool-2.4/build
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# make install prefix=/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr
 /usr/bin/install -c -m 644 ../doc/libtool.1 ../doc/libtoolize.1 '/media/0ca3397f-3ac3-4
0/usr/share/man/man1'
make[3]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
make[2]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
make[1]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
[root@localhost build]#
```

14．libtoolはクロスコンパイル環境にもインストールする。buildフォルダで先の生成したものを消去しconfigureし直す。prefixはWeb記事に従い--prefix=/とする。

../configure --prefix=/ --host=sh3-linux --build=i686-vine-gnu

```
user01@localhost.localdomain: /home/user01/GNU-std/libtool-2.4/build
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# pwd
/home/user01/GNU-std/libtool-2.4/build
[root@localhost build]# rm -rf *
[root@localhost build]# ../configure --prefix=/ --host=sh3-linux --build=i686-vine-gnu
config.status: executing tests/atconfig commands
config.status: executing depfiles commands
config.status: executing libtool commands
[root@localhost build]#
```

GNU標準パッケージの移植（8）

15. makeする。

#make


```
libtool: link: ( cd "libltdl/.libs" && rm -f "libltdl.la" && ln -s "../libltdl.la" "libltdl.la" )
make[2]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
make[1]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
[root@localhost build]#
```

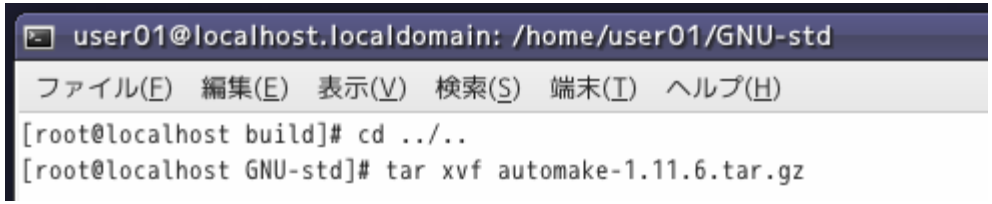
16. 記事に従い、PCの/usr/sh3-linux/sh3-linuxにインストールする。

make　install　prefix=/usr/sh3-linux/sh3-linux

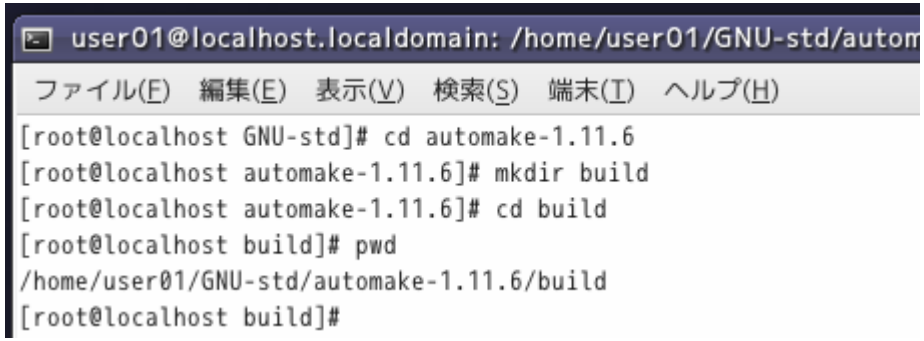

```
 /usr/bin/install -c -m 644 ../doc/libtool.1 ../doc/libtoolize.1 '/usr/sh3-l
make[3]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
make[2]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
make[1]: Leaving directory `/home/user01/GNU-std/libtool-2.4/build'
[root@localhost build]#
```

GNU標準パッケージの移植（9）

17. 続けてautomakeのクロスコンパイルを行う。同様にtarでautomake-*****.gzを展開する。



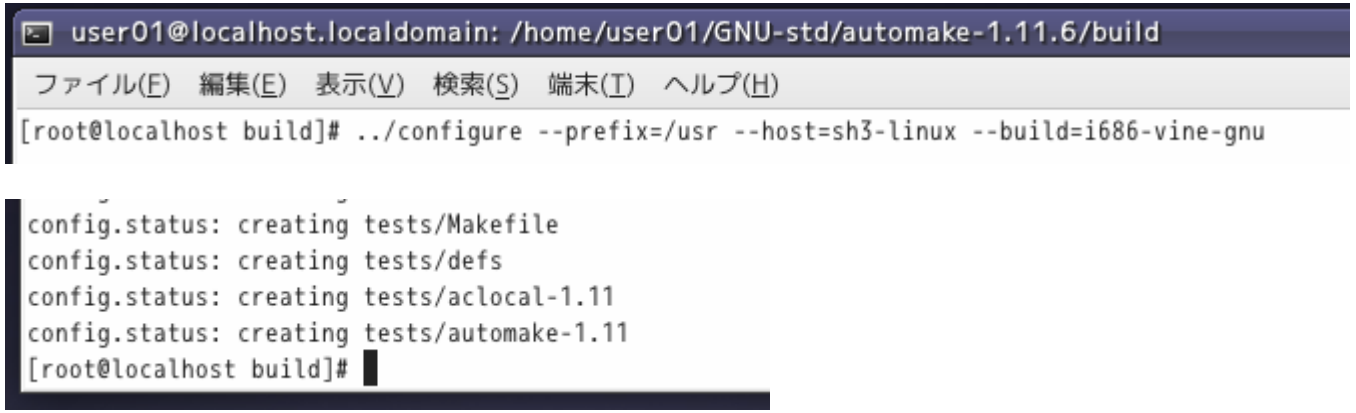
18. 展開したフォルダにbuildフォルダを作成し、buildに移動する。



GNU標準パッケージの移植（10）

19. ConfigureコマンドでMaikefileを作成する。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu



20. make する。

#make

GNU標準パッケージの移植（１１）

２１．SDカードのルートフォルダの/usrにインストールする。

#make install prefix=/media/(SDカードのルートフォルダ)/usr

２２．automakeもクロスコンパイル環境にインストールする。buildフォルダの再生データを削除しconfigureし直す。prefixはWeb記事に従い--prefix=/とする。

../configure --prefix=/ --host=sh3-linux --build=i686-vine-gnu

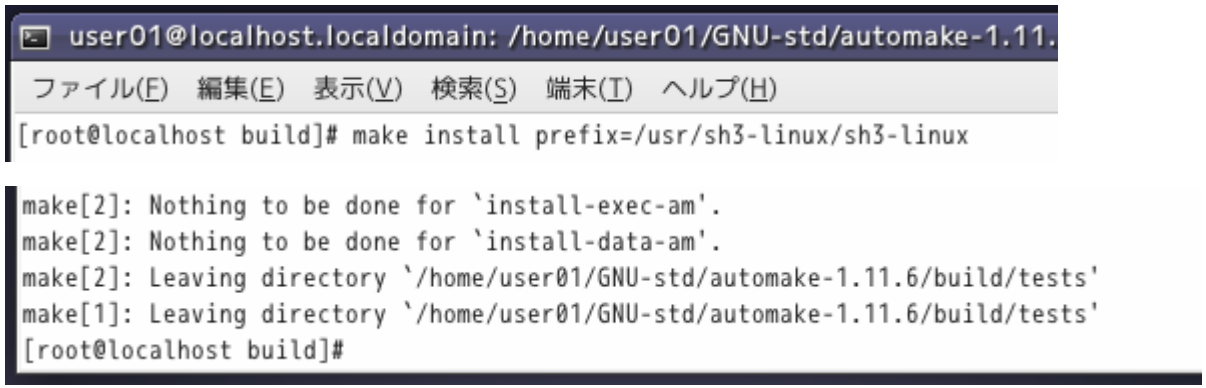
GNU標準パッケージの移植（12）

23．make する。

#make

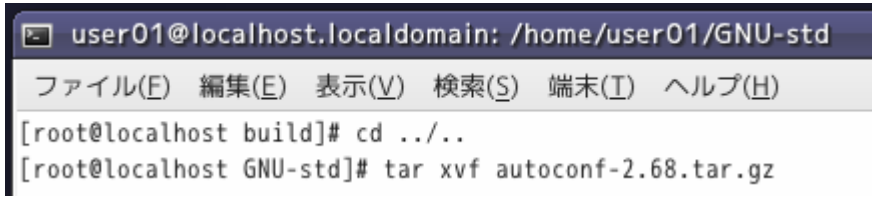
24．記事に従い、PCの/usr/sh3-linux/sh3-linuxにインストールする。

make　install　prefix=/usr/sh3-linux/sh3-linux

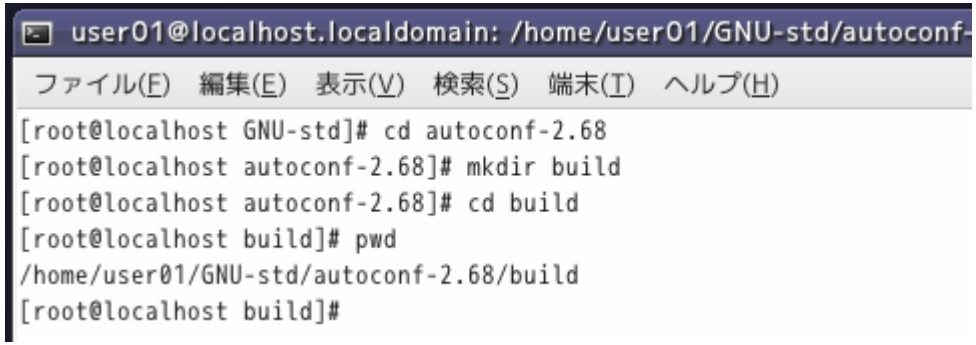


GNU標準パッケージの移植（13）

25. 続けてautoconfのクロスコンパイルを行う。同様にtarにてautoconf-*****.gzを展開する。



26. 展開したフォルダにbuildフォルダを作成し、buildに移動する。



GNU標準パッケージの移植（14）

27．ConfigureコマンドでMakefileを作成する。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu

```
user01@localhost.localdomain: /home/user01/GNU-std/autoconf-2.68/build
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# ../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu
```

```
config.status: creating lib/autotest/Makefile
config.status: creating bin/Makefile
config.status: linking ../GNUmakefile to GNUmakefile
config.status: executing tests/atconfig commands
[root@localhost build]#
```

28．makeする。

```
user01@localhost.localdomain: /h
ファイル(F)  編集(E)  表示(V)  検索(S)
[root@localhost build]# make
```

```
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/home/user01/GNU-std/autoconf-2.68/build/man'
make[1]: Leaving directory `/home/user01/GNU-std/autoconf-2.68/build'
[root@localhost build]#
```

GNU標準パッケージの移植（15）

29. SDカードのルートフォルダの/usrにインストールする。

make install prefix=/media/(SDカードのルートフォルダ)/usr

```
root@localhost.localdomain: /home/user01/GNU-std/autoconf-2.68/build
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# make install prefix=/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr
```

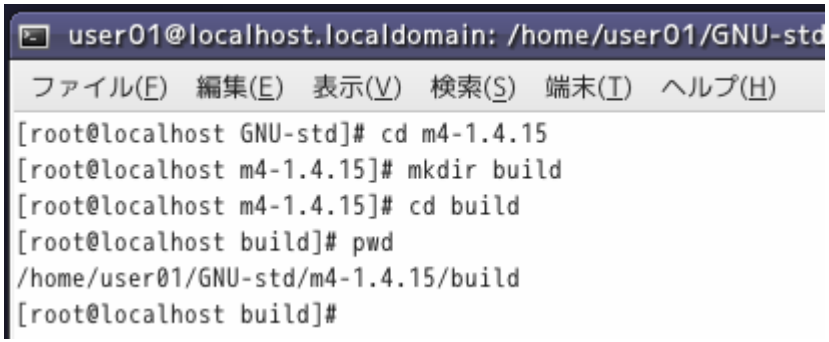
```
make[3]: Leaving directory `/home/user01/GNU-std/autoconf-2.68/build/man'
make[2]: Leaving directory `/home/user01/GNU-std/autoconf-2.68/build/man'
make[1]: Leaving directory `/home/user01/GNU-std/autoconf-2.68/build'
[root@localhost build]#
```

30. 続けてm4のクロスコンパイルを行う。同様にtarにてm4-*****.gzを展開する。

```
user01@localhost.localdomain: /home/user01/GNU-std
ファイル(F)  編集(E)  表示(V)  検索(S)  端末(T)  ヘルプ(H)
[root@localhost build]# cd ../..
[root@localhost GNU-std]# tar xvf m4-1.4.15.tar.gz
```

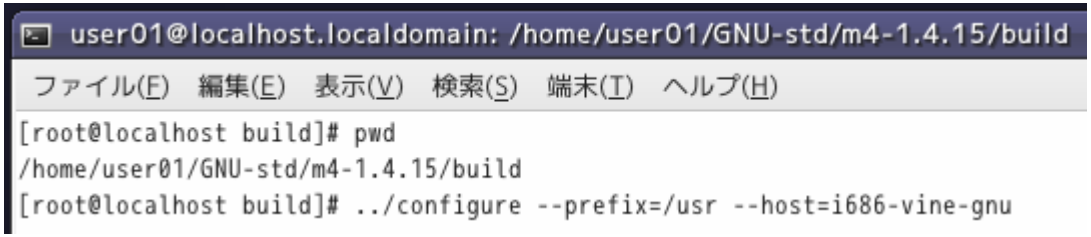
GNU標準パッケージの移植（16）

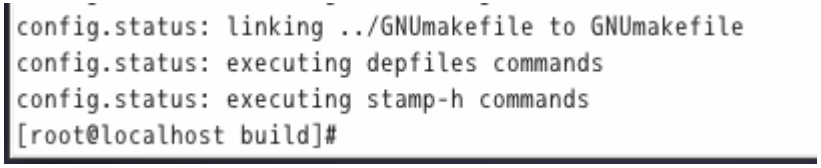
31．展開したフォルダにbuildを作成し、buildに移動する。



32．configureコマンドでmakefileを作成する。

../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu



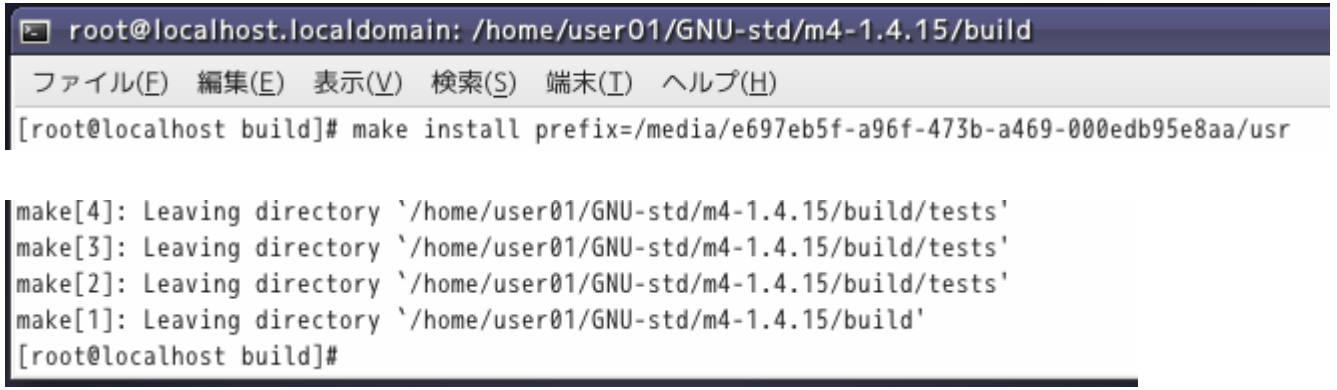


GNU標準パッケージの移植（17）

33．makeする。

34．SDカードのルートフォルダの/usrにインストールする。

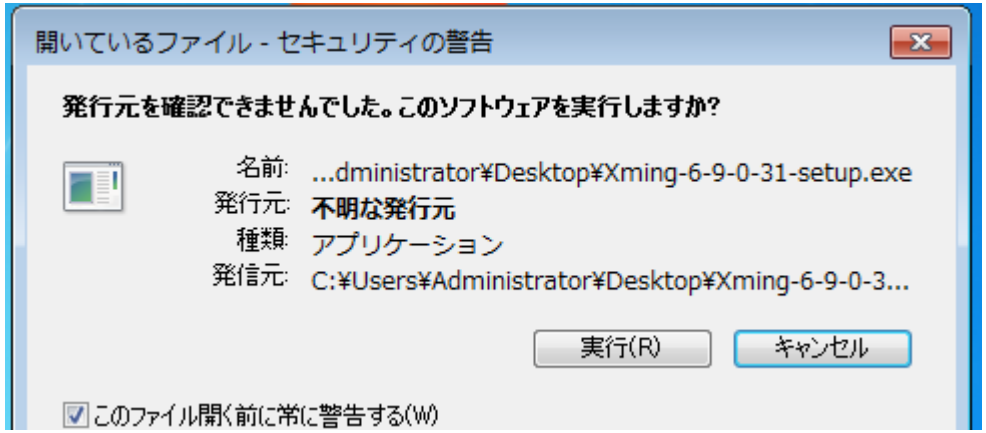
#make install prefix=/media/(SDカードのルートフォルダ)/usr



(附1)
Xmingのインストール(1)

1. インストール手順はhttp://sourceforge.jp/magazine/09/10/14/0753240に詳しく記述されているがリンク切れなど考慮して同様の内容を掲示しておく。
2. http://sourceforge.jp/projects/sfnet_xming/releases/などのサイトから Xming-6-9-31-setup.exeをダウンロードする。
3. Xming-6-9-31-setup.exeをダブルクリック

4. 実行をクリック

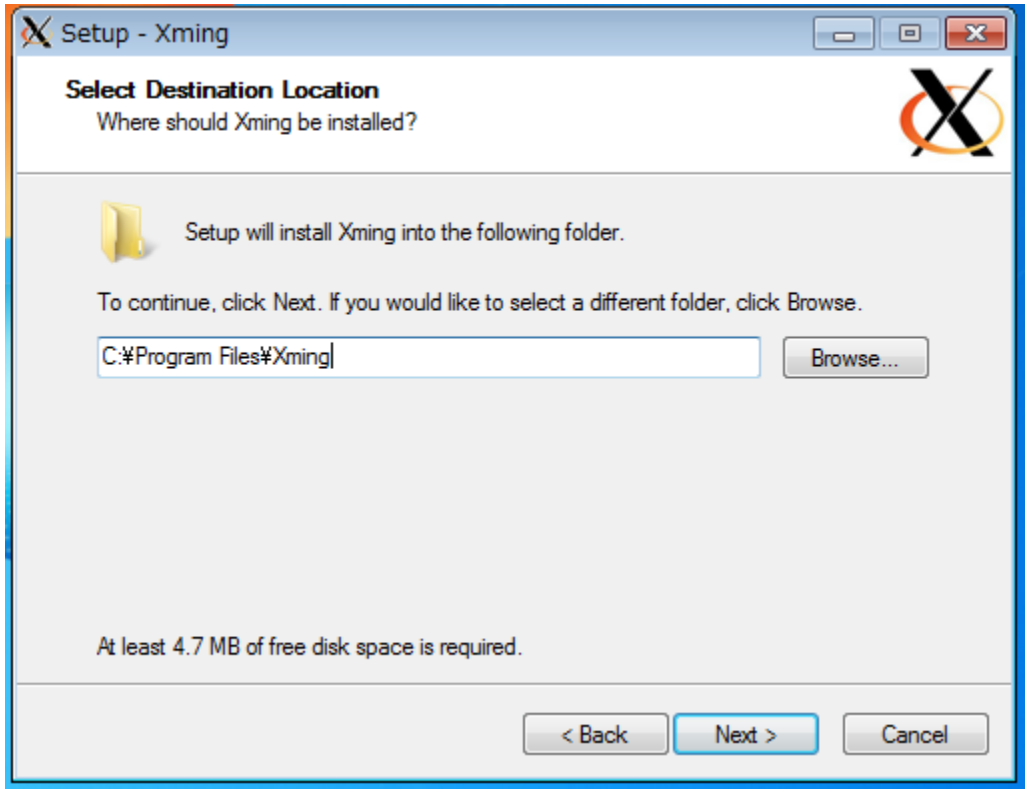


(附1)
Xmingのインストール(2)

5.「Next」をクリック

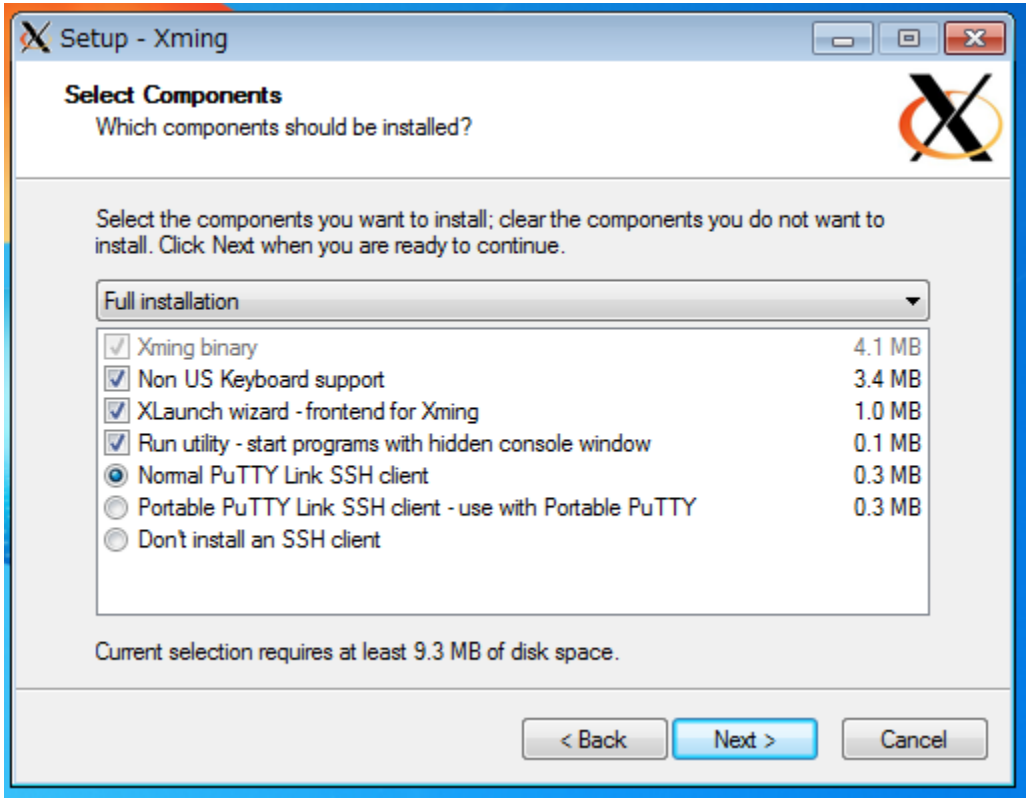
(附1)
Xmingのインストール(3)

6. 適当なフォルダを指定して「Next」をクリック。ここではディフォルトフォルダ。



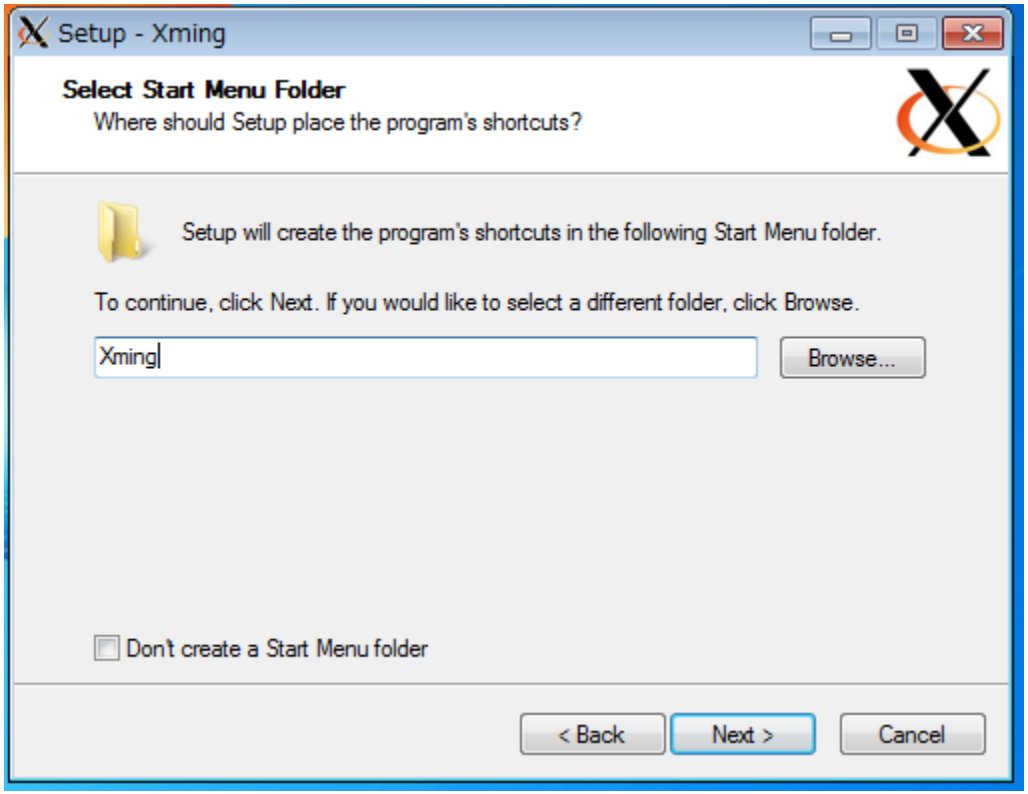
(附1)
Xmingのインストール(4)

7. Puttyはssh接続をサポートするツール。すでにインストール済みならDon't Install an SSH client を選択する。通常はディフォルトにて「Next」をクリック



(附１)
Xmingのインストール（５）

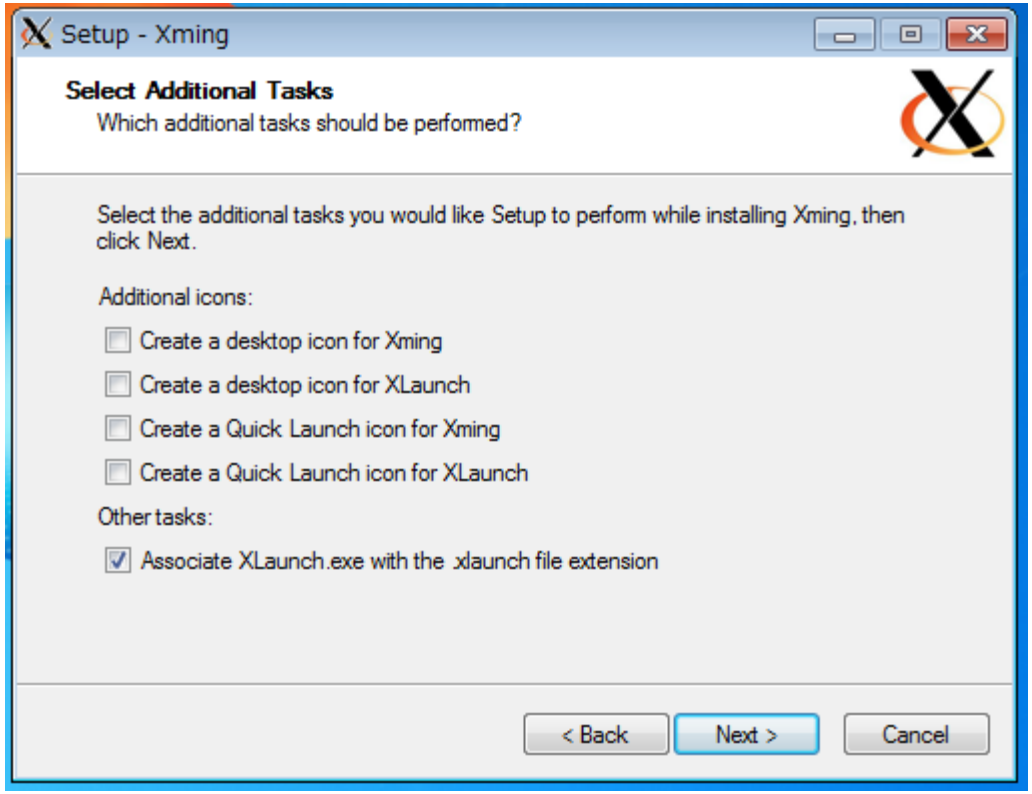
８．問題なければ、「Next」をクリック



(附1)
Xmingのインストール(6)

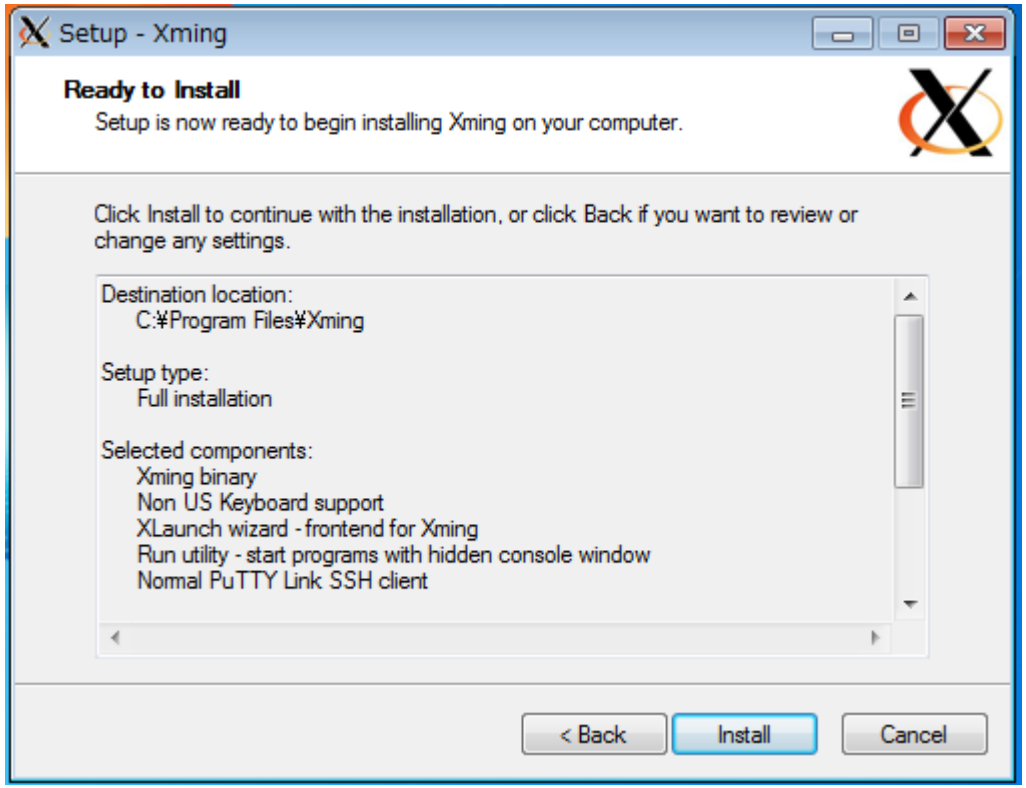
9.「Xlaunch」はXmingを起動したうえ、SSH経由で最初のターミナルを立ち上げる作業をおこなう。このまま「Next」をクリック。

ディスクトップにこれらのアイコンがほしいときは他にチェックを入れる。



(附1)
Xmingのインストール(7)

10.「Install」する。



(附1)
Xmingのインストール(8)

1.「Finish」する。