

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>

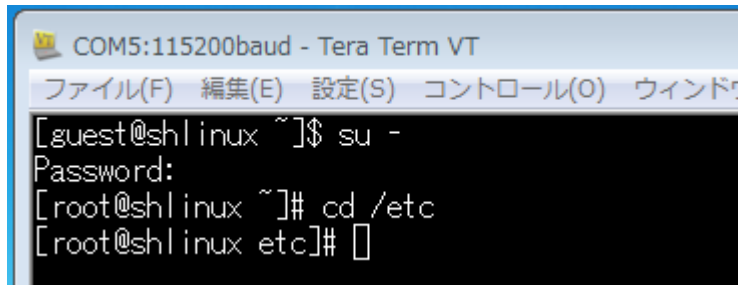
および、名もないページさんサイト(2015.4現在リンク切れです)

<http://wave2.iobb.net/doc/summary/sh3wiki/wifky.cgi>

の内容を参考に作成しています

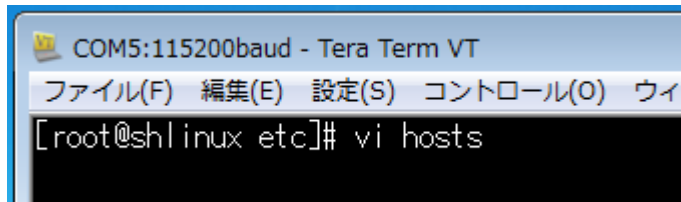
Xクライアントのssh接続(1)

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。



```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[guest@shlinux ~]$ su -
Password:
[root@shlinux ~]# cd /etc
[root@shlinux etc]#
```

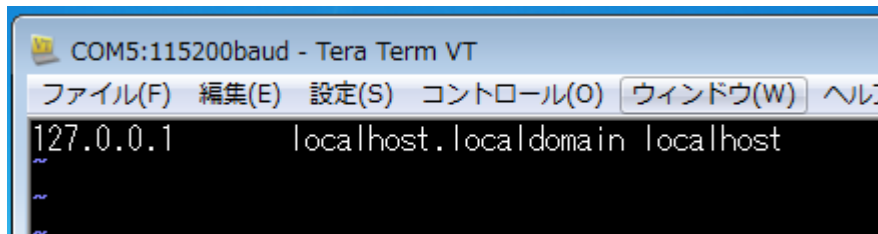
2. hosts ファイルを作成する。



```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[root@shlinux etc]# vi hosts
```

3. hostsの内容は

127.0.0.1 localhost.localdomain localhost とし、保存する。

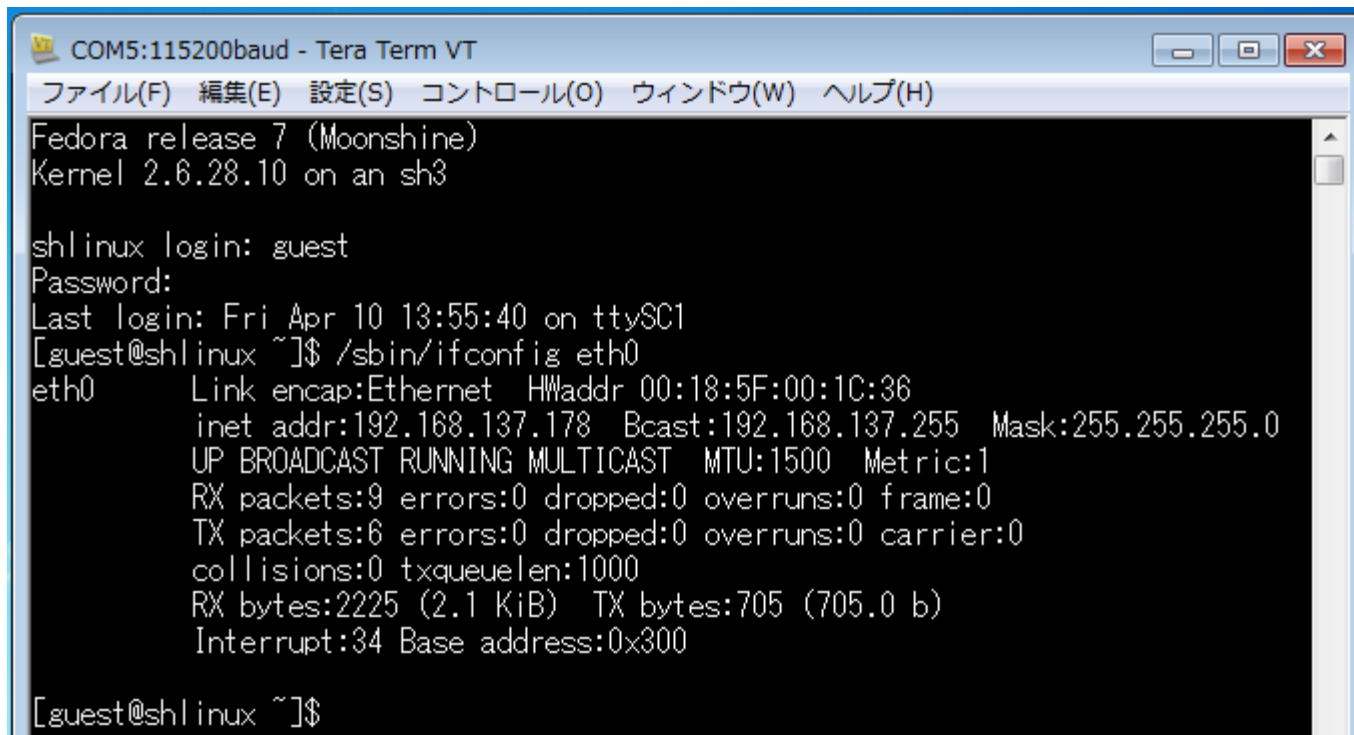


```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
127.0.0.1 localhost.localdomain localhost
~
~
~
```

Xクライアントのssh接続(2)

- 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
```



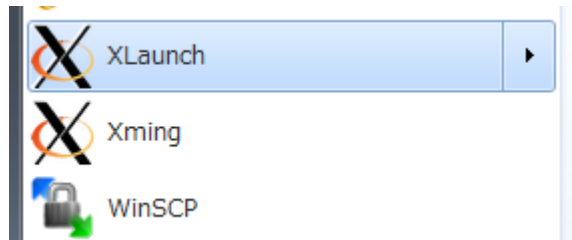
```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Fedora release 7 (Moonshine)
Kernel 2.6.28.10 on an sh3

shlinux login: guest
Password:
Last login: Fri Apr 10 13:55:40 on ttySC1
[guest@shlinux ~]$ /sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:18:5F:00:1C:36
          inet addr:192.168.137.178  Bcast:192.168.137.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9  errors:0  dropped:0  overruns:0  frame:0
          TX packets:6  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2225 (2.1 KiB)  TX bytes:705 (705.0 b)
          Interrupt:34 Base address:0x300

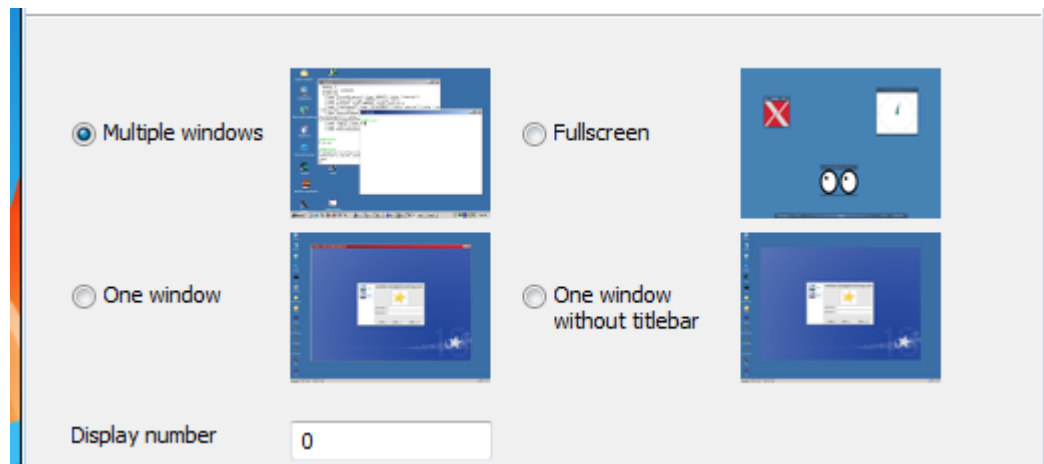
[guest@shlinux ~]$
```

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ウィ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)

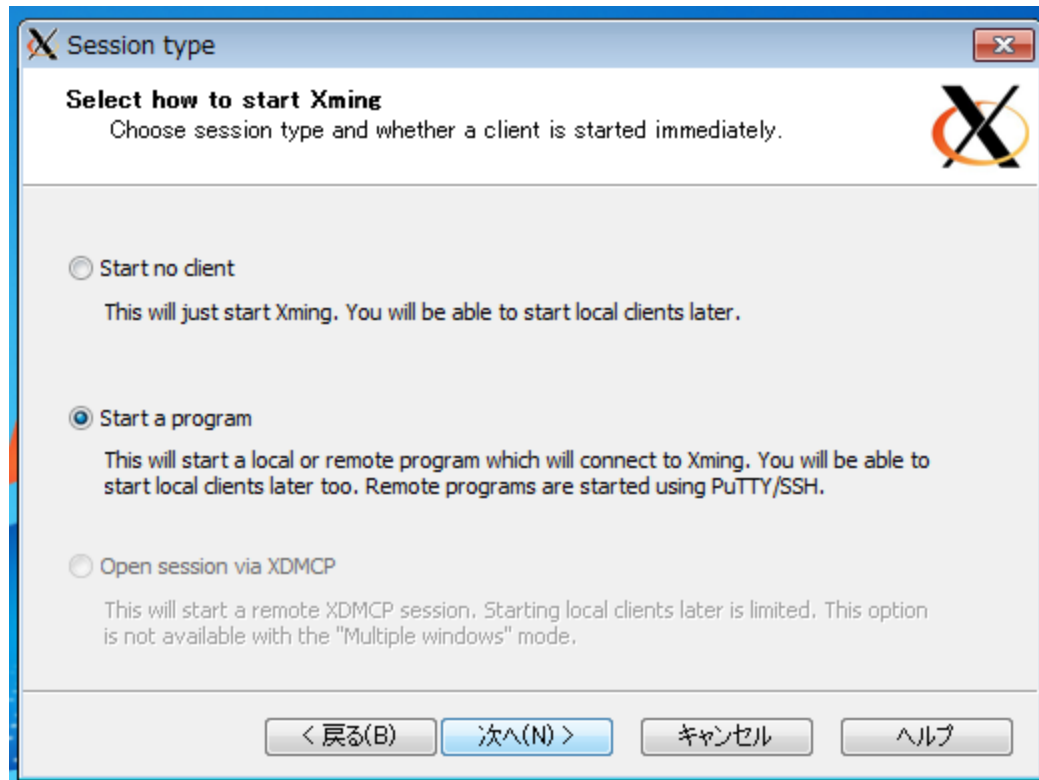


2. 標準的なMultiple windowsを選択する。この選択ではWindows画面を背景にxtermが使用できる。



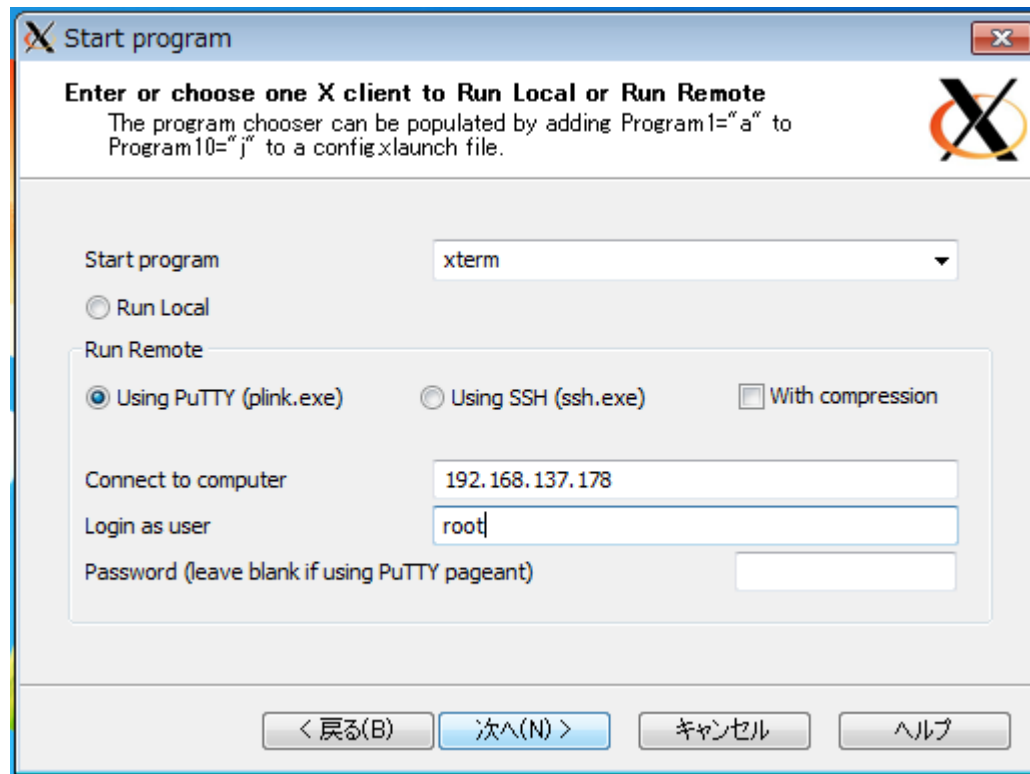
Windowsパソコンでのssh接続(2)

3. Start a program を選択して「次へ」



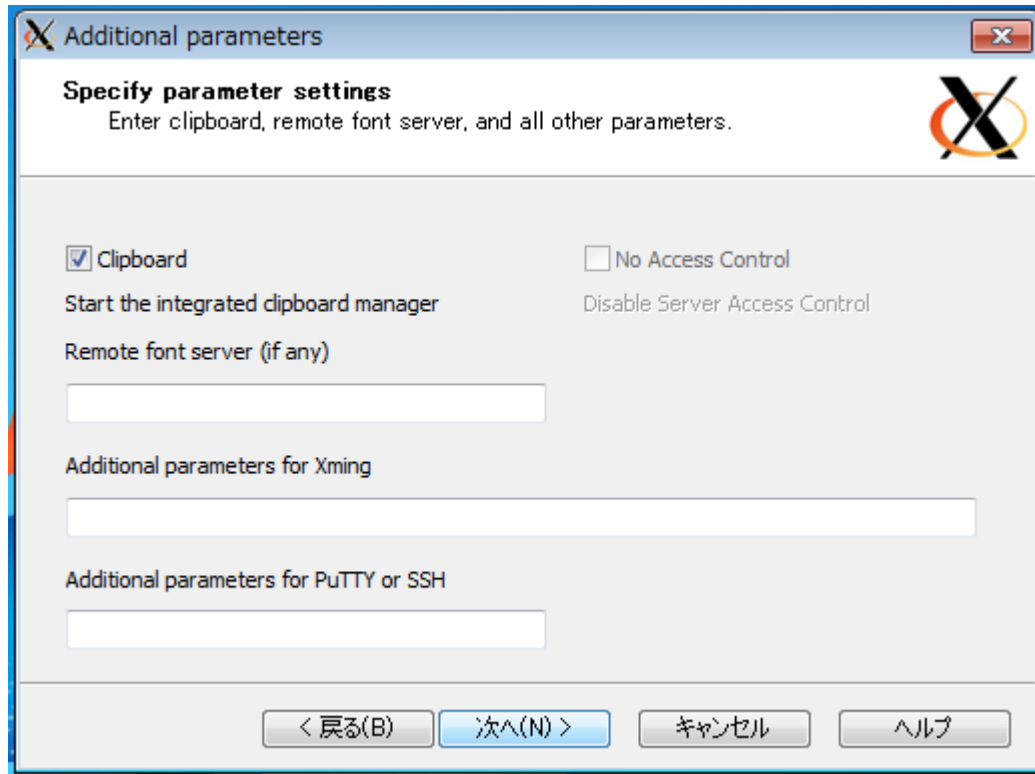
Windowsパソコンでのssh接続(3)

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「空白」にして「次へ」をクリックする。



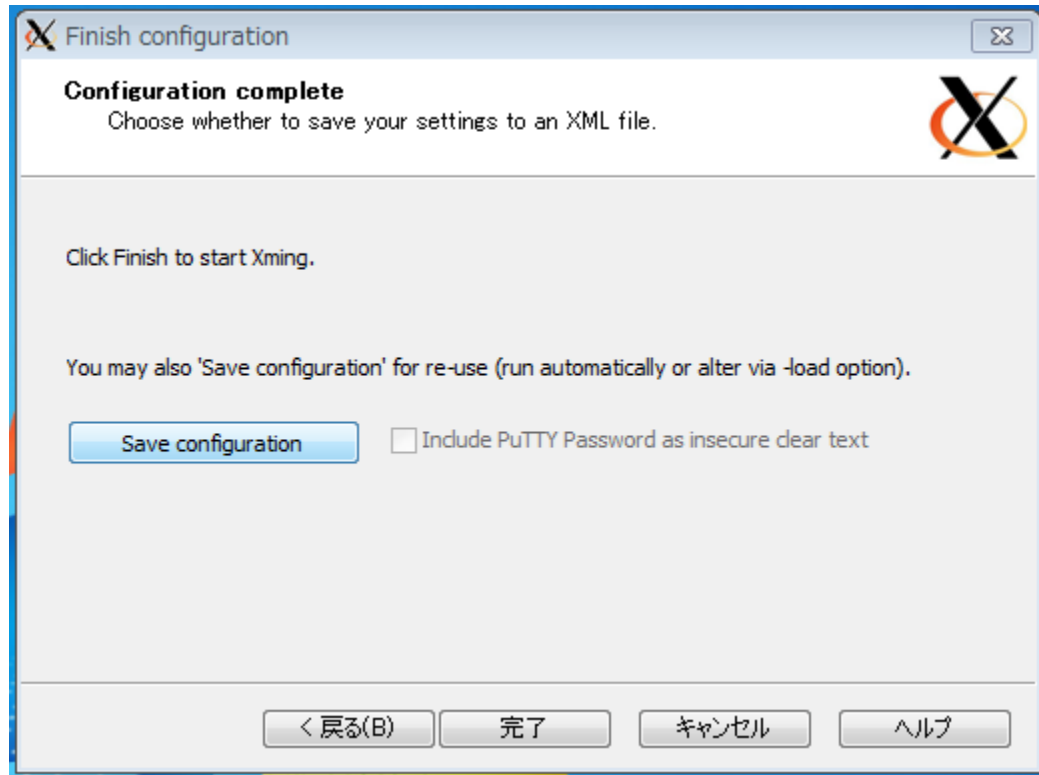
Windowsパソコンでのssh接続(4)

5. 次の画面はそのまま「次へ」



Windowsパソコンでのssh接続(5)

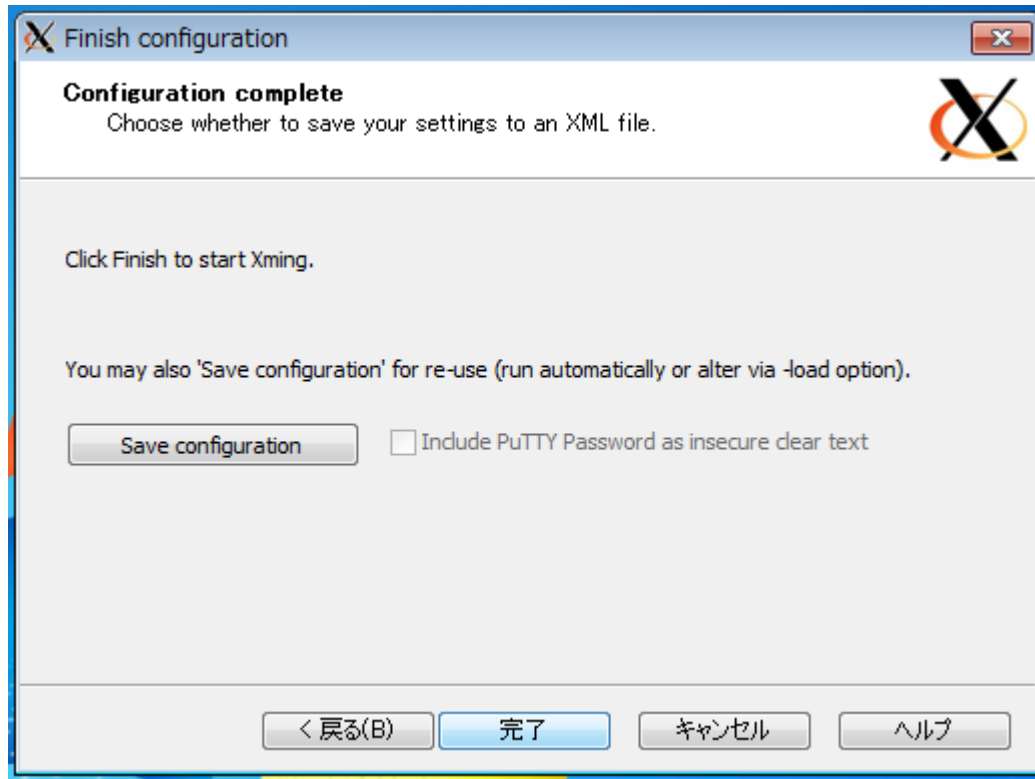
6. Configurationで「Save configuration」をクリック
2回目以降は「完了」クリックでOK



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)

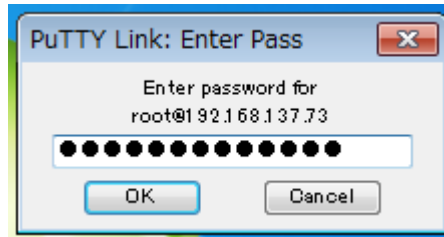
9. 「はい」をクリック



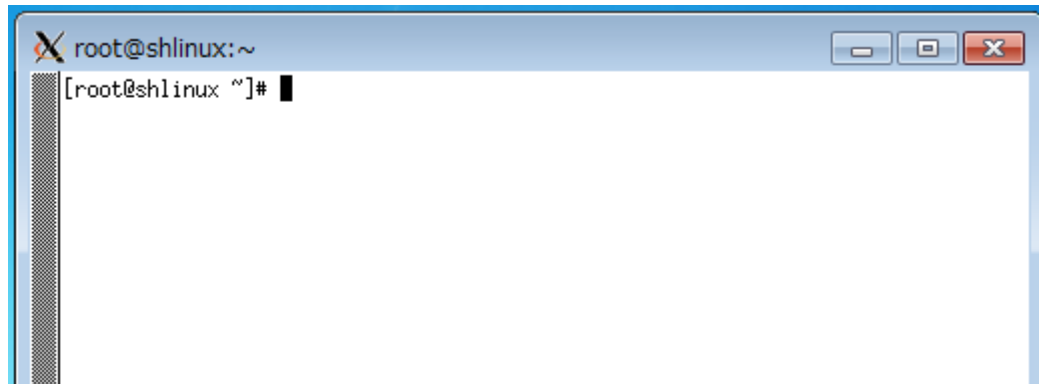
2回目以降は表示されない

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。

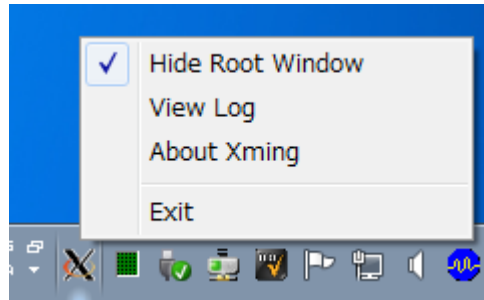
その判断方法としてタスクバーにある以下のアイコン「X」表示で、接続処理が不調に終わると何のメッセージも残さず消えてゆく。

(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ウィ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。)

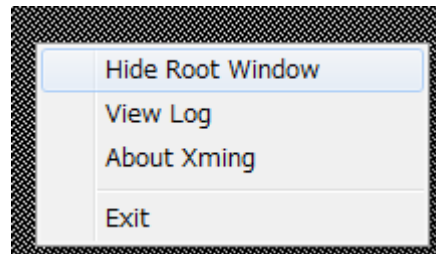


Unixライクな画面で作業したい人向き用でのxtermが表示できる。



Windowsパソコンでのssh接続(11)

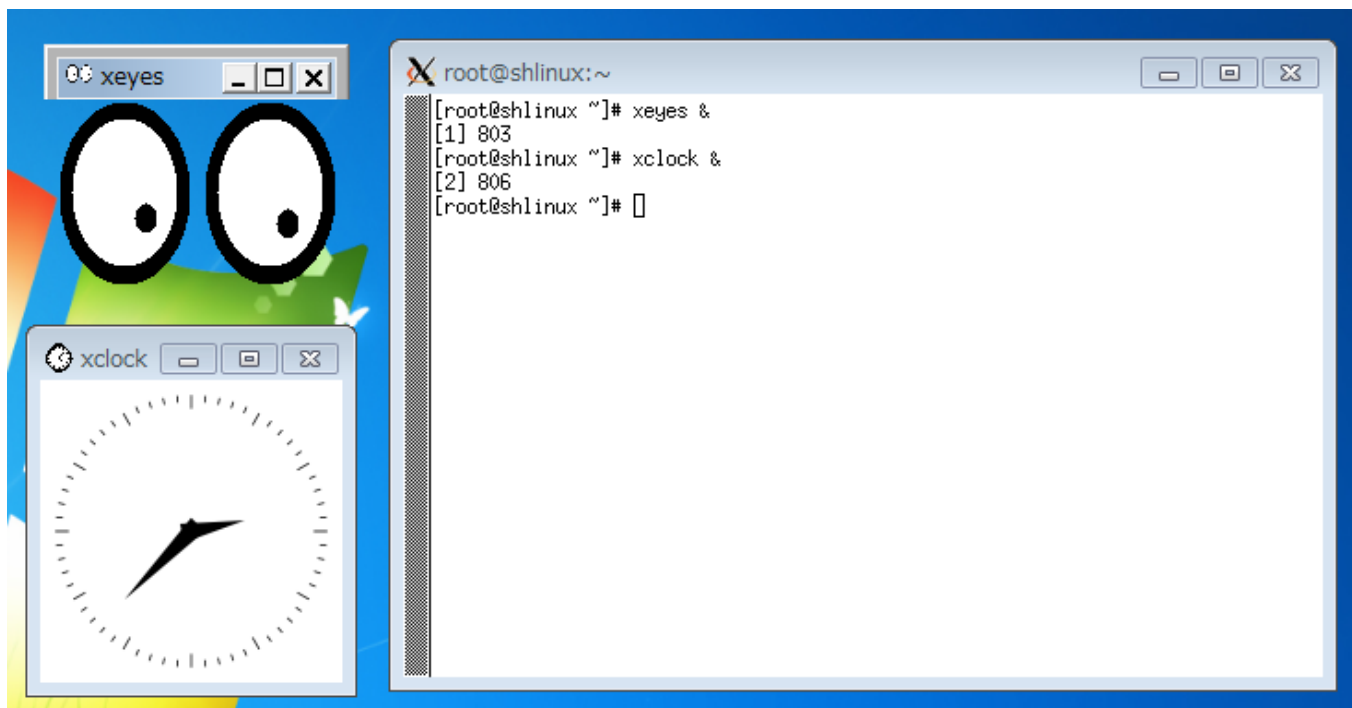
14. Unixになじみのない人には、突然格子模様になるとびっくりするが、元にもどすには[Ctrl]+[Alt]+[Del]をクリックし、「タスクマネージャーの起動」を選択するとタスクバーが現れるので改めて右クリックで下記ポップアップウインドウを開き「Hide Root Window」にチェックを入れればよい。



Windowsパソコンでのssh接続(12)

15. xeyesとxclockを起動する。

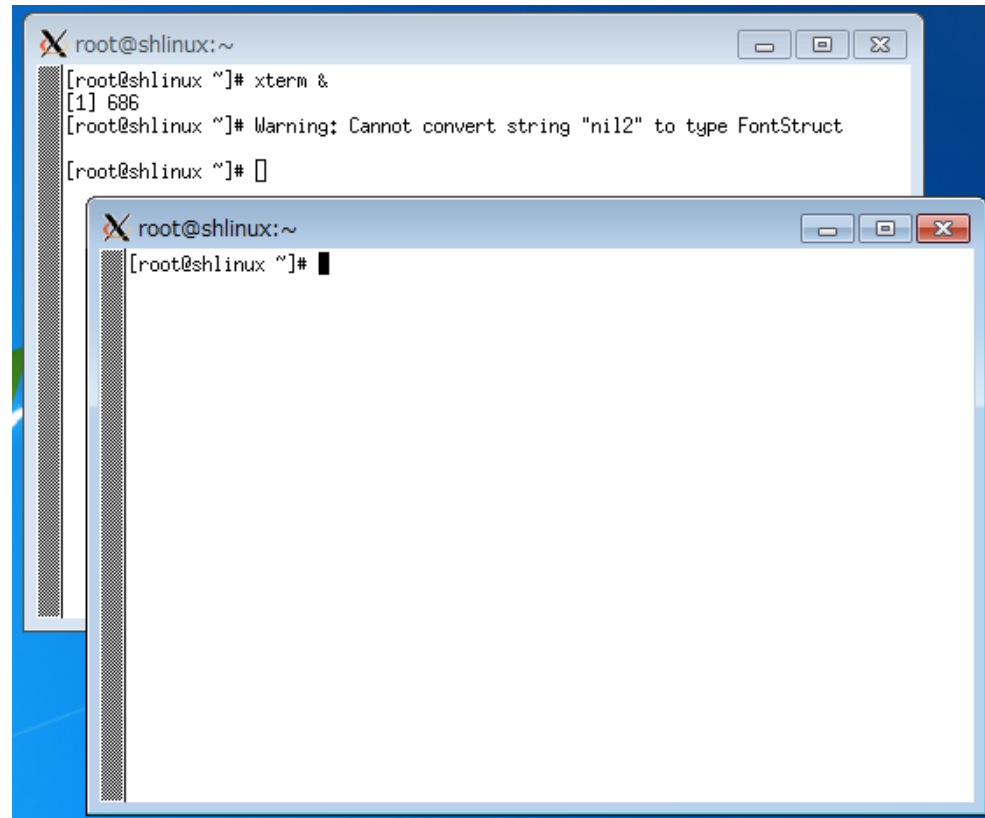
```
root@shlinux:~  
[root@shlinux ~]# xeyes &  
[1] 803  
[root@shlinux ~]# xclock &  
[2] 806  
[root@shlinux ~]# █
```



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。

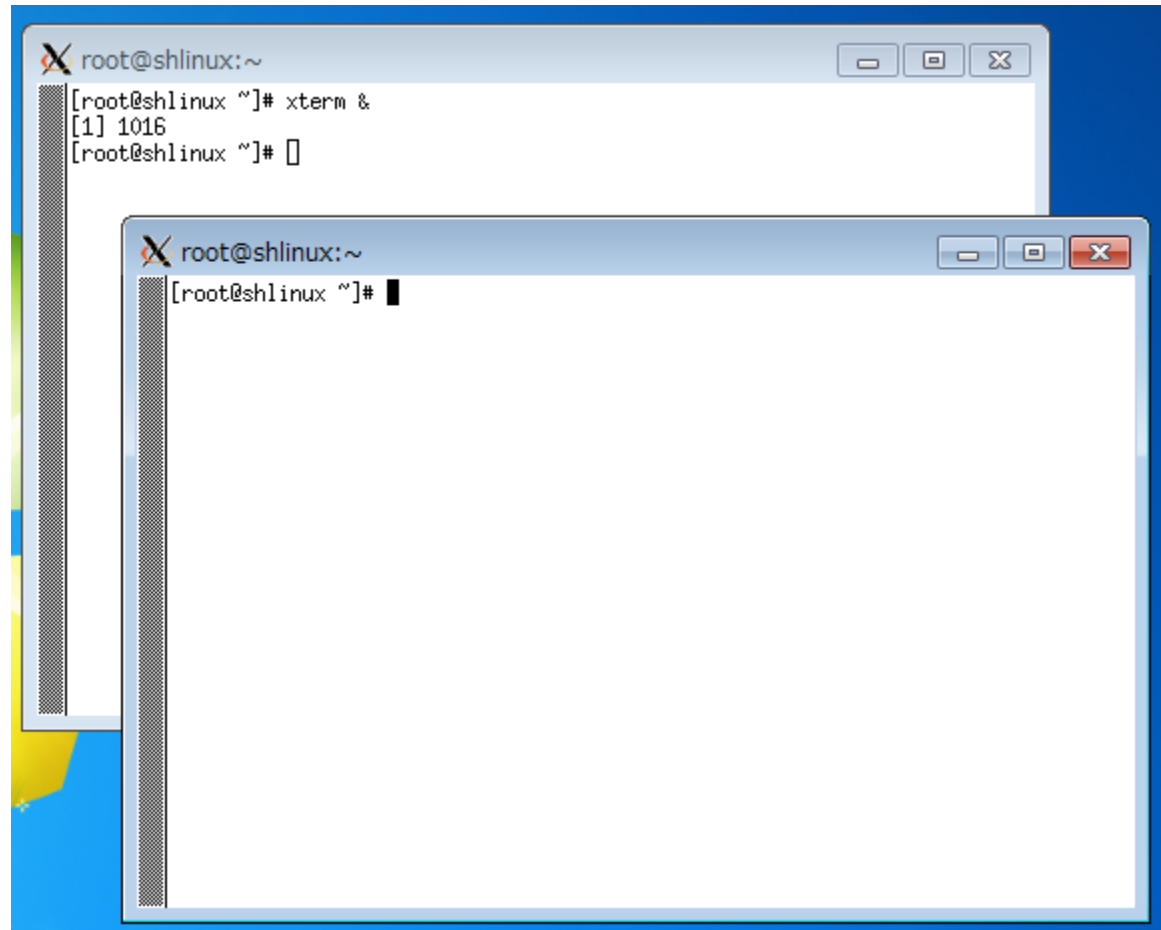
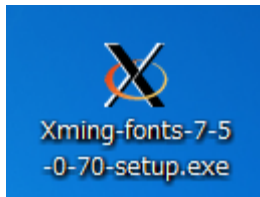
```
root@shlinux:~  
[root@shlinux ~]# xterm &
```



```
root@shlinux:~  
[root@shlinux ~]# xterm &  
[1] 686  
[root@shlinux ~]# Warning: Cannot convert string "nil2" to type FontStruct  
[root@shlinux ~]#  
root@shlinux:~  
[root@shlinux ~]#
```


Windowsパソコンでのssh接続(14)

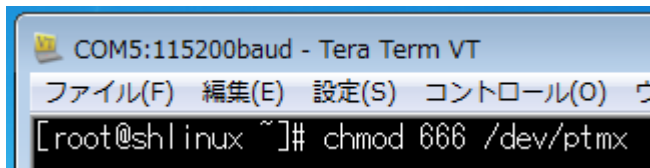
17. # xterm & で....”nil2“...のワーニングが煩わしいのでXming-fontsをインストールする。

The image shows two terminal windows from a Windows environment. The top window shows the command 'xterm &' being executed, resulting in a new terminal window opening. The bottom window shows the prompt '[root@shlinux ~]#', indicating that the user is now in a shell environment. The terminal windows have a blue title bar and standard window controls (minimize, maximize, close).

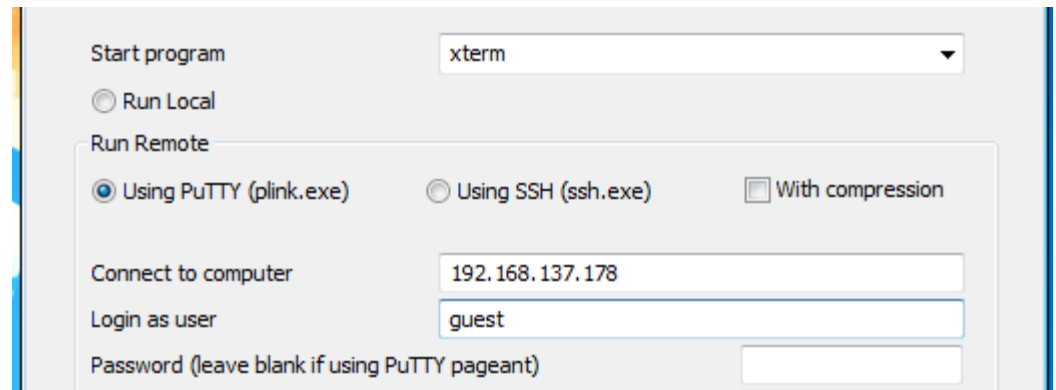
```
root@shlinux:~  
[root@shlinux ~]# xterm &  
[1] 1016  
[root@shlinux ~]#  
  
root@shlinux:~  
[root@shlinux ~]#
```

Windowsパソコンでのssh接続(15)

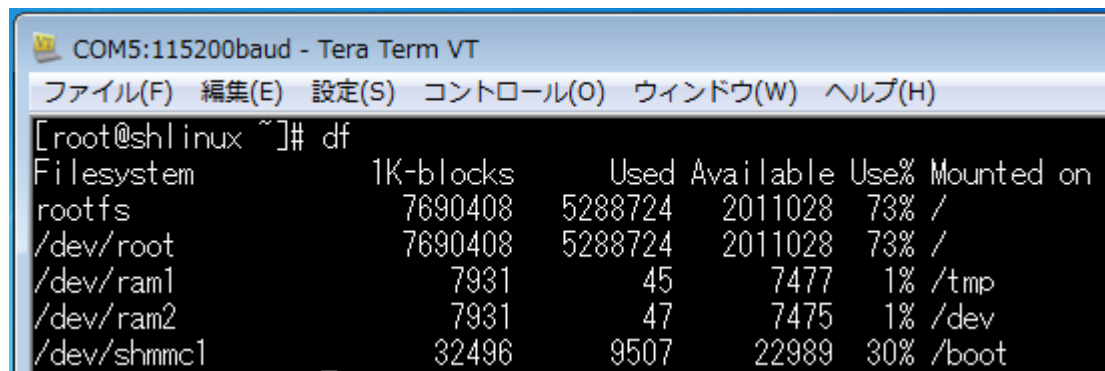
18. 一般ユーザのguestとして接続し、xtermを起動させるには、ターゲットの/dev/ptmxのパーミッションを666にする。



```
COM5:115200baud - Tera Term VT  
ファイル(F) 編集(E) 設定(S) コントロール(O) ウ  
[root@shlinux ~]# chmod 666 /dev/ptmx
```



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。



```
COM5:115200baud - Tera Term VT  
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)  
[root@shlinux ~]# df  
Filesystem      1K-blocks    Used Available Use% Mounted on  
rootfs          7690408  5288724  2011028  73% /  
/dev/root       7690408  5288724  2011028  73% /  
/dev/ram1        7931         45      7477    1% /tmp  
/dev/ram2        7931         47      7475    1% /dev  
/dev/shmmc1     32496        9507    22989   30% /boot
```

Windowsパソコンでのssh接続(16)

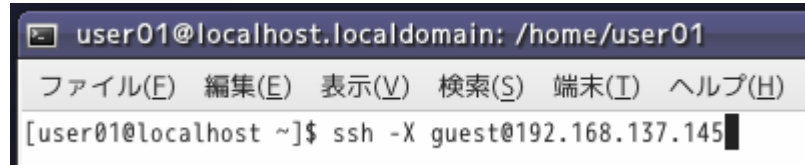
19. SDカードの(SDカード)/dev/ptmsのパーミションを666にすれば再起動時にもパーミションは元に戻らない。

```
user01@localhost.localdomain: /media/e697eb5f-a96f-473b-a469-000edb95
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost user01]# cd /media/e697eb5f-a96f-473b-a469-000edb95e8aa/dev
[root@localhost dev]# pwd
/media/e697eb5f-a96f-473b-a469-000edb95e8aa/dev
[root@localhost dev]# ls -l ptmx
crw-r--r-- 1 root root 5, 2  9月 21  2007 ptmx
[root@localhost dev]# chmod 666 ptmx
[root@localhost dev]# ls -l ptmx
crw-rw-rw- 1 root root 5, 2  9月 21  2007 ptmx
[root@localhost dev]#
```

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
```



```
user01@localhost.localdomain: /home/user01  
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)  
[user01@localhost ~]$ 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。



```
guest@shlinux:~  
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)  
[user01@localhost ~]$ ssh -X guest@192.168.137.145  
guest@192.168.137.145's password:  
Last login: Fri Sep 12 12:57:51 2014 from 192.168.137.4  
[guest@shlinux ~]$ xeyes &  
[1] 1847  
[guest@shlinux ~]$ xclock &  
[2] 1850  
[guest@shlinux ~]$
```

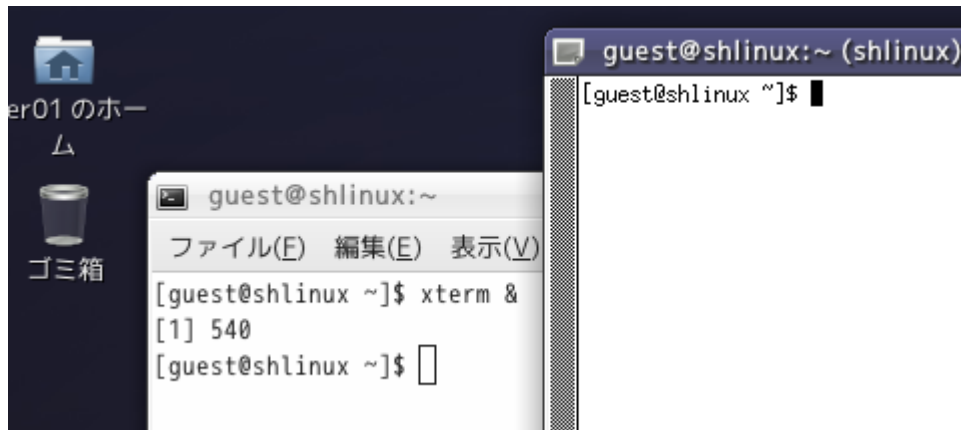
The screenshot shows two graphical windows on the remote host. The top window, titled 'xeyes', displays two large, white, cartoonish eyes with black pupils. The bottom window, titled 'xclock', displays a simple analog clock face with a black dial and a single black hand pointing to the 12 o'clock position.

Linuxマシンでのssh接続(2)

- 複数のターミナルがほしい場合、xtermを追加表示する。



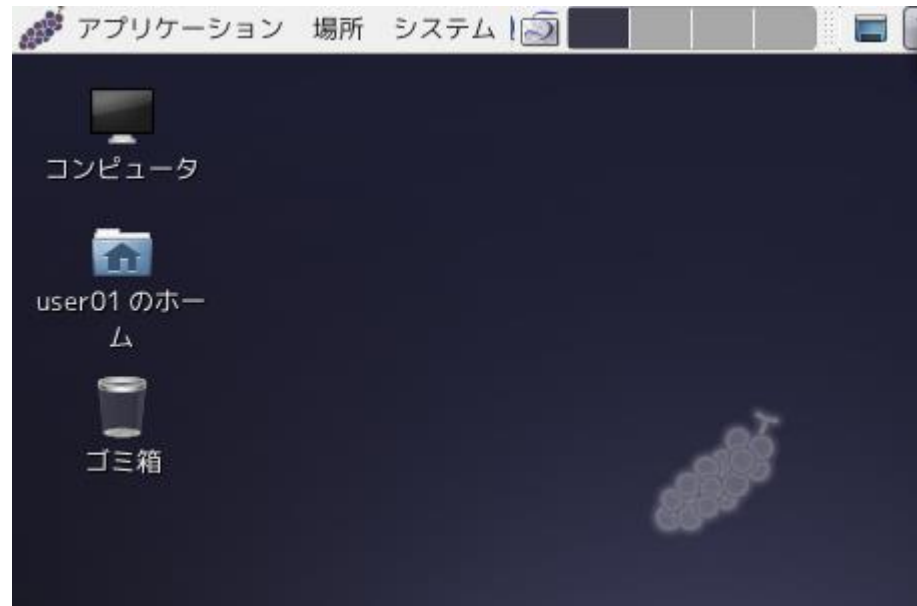
```
guest@shlinux:~  
ファイル(E) 編集(E) 表示(V) 検索(S) 設定(O) ヘルプ(H) 終了(X)  
[guest@shlinux ~]$ xterm &  
[1] 540
```



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)

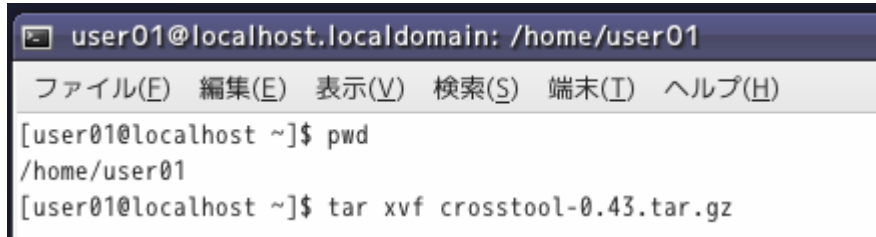


クロスコンパイラとセルフコンパイラの生成(2)

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。



```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ tar xvf crosstool-0.43.tar.gz
```

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。



```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ mkdir downloads
```

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。

```
user01@localhost.localdomain: /home/user01/downloads
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[user01@localhost ~]$ cd downloads/
[user01@localhost downloads]$ ls
binutils-2.16.1.tar.bz2  glibc-linuxthreads-2.3.6.tar.bz2
gcc-3.3.6.tar.bz2      linux-2.6.15.4.tar.gz
gcc-4.1.2.tar.bz2     linux-2.6.28.10.tar.bz2
gdb-6.5.tar.bz2       linux-2.6.39.4.tar.bz2
glibc-2.3.6.tar.bz2   linux-libc-headers-2.6.12.0.tar.bz2
[user01@localhost downloads]$
```


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。

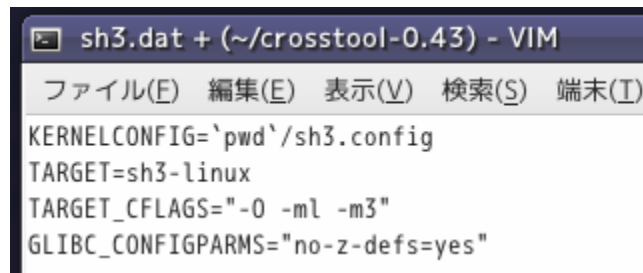


```
user01@localhost.localdomain: /home/user01/cro
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ cd crosstool-0.43
[user01@localhost crosstool-0.43]$ vi sh3.dat
```

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フォルトのままではなく、

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

に変更する。



```
sh3.dat + (~/.crosstool-0.43) - VIM
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T)
KERNELCONFIG=`pwd`/sh3.config
TARGET=sh3-linux
TARGET_CFLAGS="-O -m1 -m3"
GLIBC_CONFIGPARMS="no-z-defs=yes"
```

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/crosstool
ファイル(E) 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ールートでの“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での“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(E) 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(E) 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(E) 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も "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(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[user01@localhost ~]$ vi glibc-2.3.6-ls-as.patch
```

```
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) 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 $ac_prog_version in
-       '2.1[3-9]*') ac_prog_version="v. ???, bad"; ac_verc_fail=yes;;
+       '2.1[3-9]* | 2.2*')
         ac_prog_version="$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 $ac_prog_version in
-       '2.1[3-9]*')
+       '2.1[3-9]* | 2.2*')
         ac_prog_version="$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。

```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ cp glibc-2.3.6-ls-as.patch ~/crosstool-0.43/patches/glibc-2.3.6/
```

13. \$./demo-sh.shにてクロスコンパイラの作成をスタート。

```
user01@localhost.localdomain: /home/user01/crosstool
ファイル(E) 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]$ ./demo-sh3.sh
```

14. クロスコンパイラ作成の終了画面(約20分)

```
+ /home/user01/crosstool/gcc-4.1.2-glibc-2.3.6/sh3-linux/bin/sh3-linux-g++ hello2.cc -o sh3-linux-hello2
+ echo testhello: C compiler can in fact build a trivial program.
testhello: C compiler can in fact build a trivial program.
+ test '' = 1
+ test '' = 1
+ test '' = 1
+ test 1 = ''
+ echo Done.
Done.
[user01@localhost crosstool-0.43]$
```

クロスコンパイラとセルフコンパイラの生成(9)

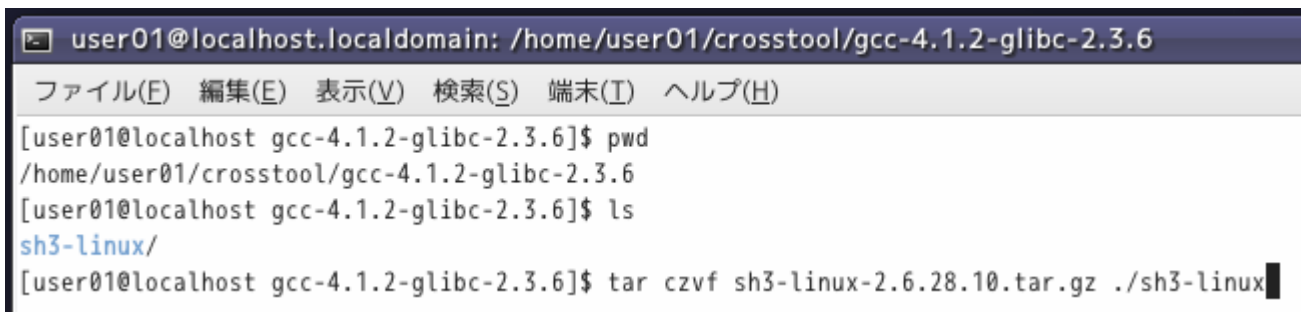
15. クロスコンパイラは demo-sh3.shで設定したように ~/crosstool フォルダ配下に構築されている。

```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ ls ~/crosstool
gcc-4.1.2-glibc-2.3.6/
[user01@localhost ~]$ ls ~/crosstool/gcc-4.1.2-glibc-2.3.6/
sh3-linux/
[user01@localhost ~]$ ls ~/crosstool/gcc-4.1.2-glibc-2.3.6/sh3-linux/
bin/          include/  lib/      man/      sh3-linux.crosstoolconfig.txt  tmp/
distributed/ info/     libexec/  sh3-linux/ share/
[user01@localhost ~]$
```

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
```



```
user01@localhost.localdomain: /home/user01/crosstool/gcc-4.1.2-glibc-2.3.6
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ pwd
/home/user01/crosstool/gcc-4.1.2-glibc-2.3.6
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ ls
sh3-linux/
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ tar czvf sh3-linux-2.6.28.10.tar.gz ./sh3-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。



```
user01@localhost.localdomain: /home/user01/cross
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ pwd
/home/user01/crosstool/gcc-4.1.2-glibc-2.3.6
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ ls
sh3-linux/  sh3-linux-2.6.28.10.tar.gz
[user01@localhost gcc-4.1.2-glibc-2.3.6]$
```

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。

```
user01@localhost.localdomain: /home/user01/crosstool/gcc-4.1.2-glibc-2.3.6
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ su
パスワード:
[root@localhost gcc-4.1.2-glibc-2.3.6]# mv ./sh3-linux-2.6.28.10.tar.gz /usr
```

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)

```
user01@localhost.localdomain: /usr
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost gcc-4.1.2-glibc-2.3.6]# cd /usr
[root@localhost usr]# pwd
/usr
[root@localhost usr]# ls
bin/  games/  lib/  libexec/  sbin/  share/  tmp@
etc/  include/  lib64/  local/  sh3-linux-2.6.28.10.tar.gz  src/
[root@localhost usr]#
```

```
user01@localhost.localdomain: /usr
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost usr]# tar xf sh3-linux-2.6.28.10.tar.gz
[root@localhost usr]# ls
bin/  games/  lib/  libexec/  sbin/  sh3-linux-2.6.28.10.tar.gz  src/
etc/  include/  lib64/  local/  sh3-linux/  share/  tmp@
[root@localhost usr]#
```


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でを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。

```
user01@localhost.localdomain: /root
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost ~]# pwd
/root
[root@localhost ~]# /usr/sh3-linux/bin/sh3-linux-gcc
sh3-linux-gcc: no input files
[root@localhost ~]#
```

“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。

```
user01@localhost.localdomain: /root
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost ~]# pwd
/root
[root@localhost ~]# vi .bash_profile

# User specific environment and startup programs


ENV=$HOME/.bashrc
PATH=$PATH:/usr/sh3-linux/bin
USERNAME="root"

export USERNAME ENV PATH
```

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
```



```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost ~]# exit
exit
[user01@localhost gcc-4.1.2-glibc-2.3.6]$ cd ~
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ 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。

```
user01@localhost.localdomain: /home/user01/sample
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ pwd
/home/user01
[user01@localhost ~]$ mkdir sample
[user01@localhost ~]$ cd sample
[user01@localhost sample]$ vi hello.c
```

```
hello.c + (~/sample) - VIM
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T)
#include <stdio.h>

int main(){
    printf("Hello World!!\n");
    return 0;
}
```

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)

```
user01@localhost.localdomain: /home/user01/sample
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost sample]$ sh3-linux-gcc -static -o hello hello.c
```

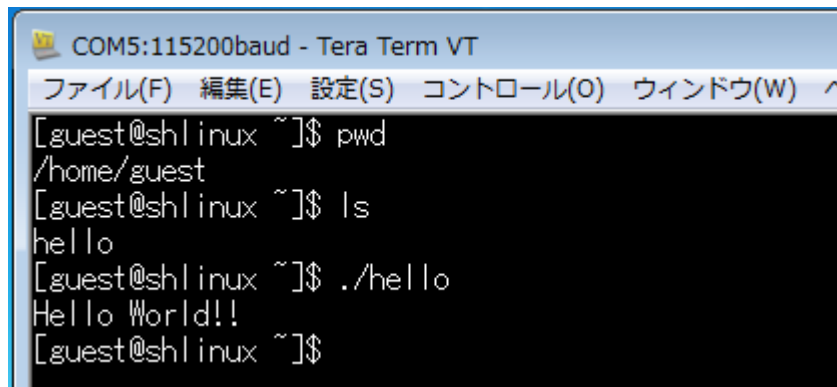
クロスコンパイラとセルフコンパイラの生成(15)

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。



```
user01@localhost.localdomain: /home/user01/sample
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost sample]$ ls -l hell*
-rwxr-xr-x 1 user01 user01 423474  4月  3 10:07 hello*
-rw-r--r-- 1 user01 user01    73  4月  3 10:06 hello.c
[user01@localhost sample]$ cp hello /media/0ca3397f-3ac3-4430-9cb5-6f9dcc47f830/home/guest
[user01@localhost sample]$
```

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。



```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[guest@shlinux ~]$ pwd
/home/guest
[guest@shlinux ~]$ ls
hello
[guest@shlinux ~]$ ./hello
Hello World!!
[guest@shlinux ~]$
```

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。

```
user01@localhost.localdomain: /home/user01/downloads
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[user01@localhost ~]$ cd downloads
[user01@localhost downloads]$ su
パスワード:
[root@localhost downloads]# tar xvjf binutils-2.16.1.tar.bz2
```

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。

```
user01@localhost.localdomain: /home/user01/downloads/binu
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost downloads]# cd binutils-2.16.1
[root@localhost binutils-2.16.1]# mkdir build
[root@localhost binutils-2.16.1]# pwd
/home/user01/downloads/binutils-2.16.1
[root@localhost binutils-2.16.1]#
```

クロスコンパイラとセルフコンパイラの生成(17)

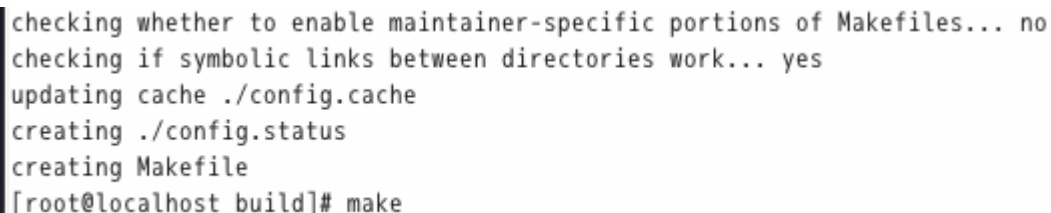
31. buildに移動後、configureにてMakefileを作成する。対象フォルダは/usrに設定。

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



```
user01@localhost.localdomain: /home/user01/downloads/binutils-2.16.1/build
ファイル(F) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost binutils-2.16.1]# cd build
[root@localhost build]# ../configure --prefix=/usr --host=sh3-linux --build=i686-vine-gnu
```

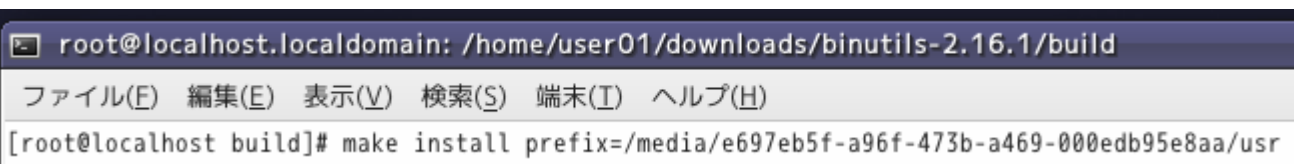
引き続き # make を実行



```
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
```

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
```



```
root@localhost.localdomain: /home/user01/downloads/binutils-2.16.1/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
```

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。

```
user01@localhost.localdomain: /home/user01/downloads
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost build]# cd ../../
[root@localhost downloads]# pwd
/home/user01/downloads
[root@localhost downloads]# tar xvjf gcc-4.1.2.tar.bz2
```

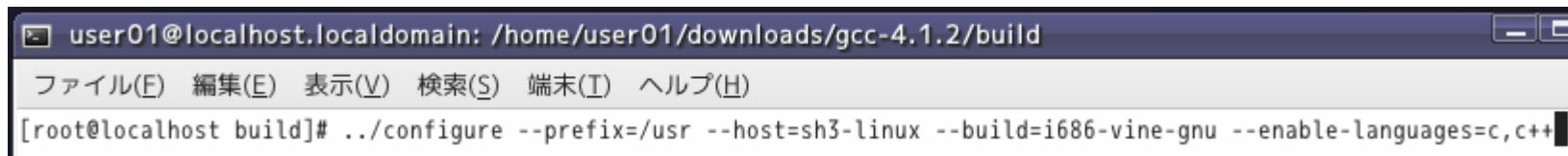
34. 展開後、gcc-4.1.2に入り、buildフォルダを作成し、buildフォルダに移動する。

```
user01@localhost.localdomain: /home/user01/downloads/gcc-4
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost downloads]# cd gcc-4.1.2
[root@localhost gcc-4.1.2]# mkdir build
[root@localhost gcc-4.1.2]# cd build
[root@localhost build]# pwd
/home/user01/downloads/gcc-4.1.2/build
[root@localhost build]# █
```

クロスコンパイラとセルフコンパイラの生成(19)

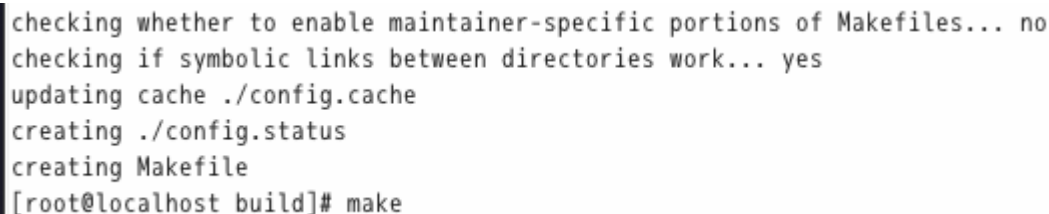
35. buildに入った後、configureにてMakefileを作成する。対象フォルダを/usrに設定。

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



```
user01@localhost.localdomain: /home/user01/downloads/gcc-4.1.2/build
ファイル(E) 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 --enable-languages=c,c++
```

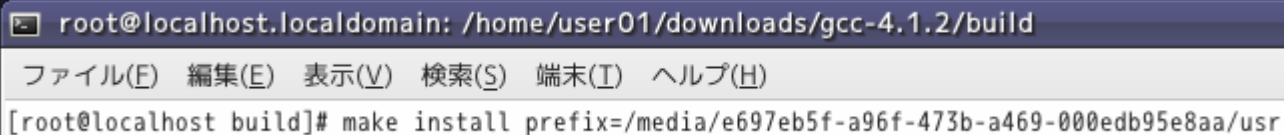
引き続き # make する。(約5分)



```
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
```



```
root@localhost.localdomain: /home/user01/downloads/gcc-4.1.2/build
ファイル(E) 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
```


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。

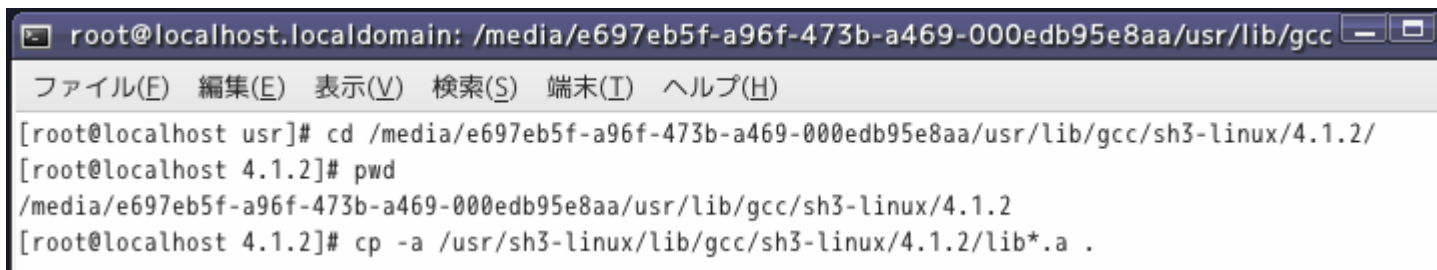
```
/usr/bin/install: 宛先の '/media/8afb8188-3c71-45a7-814c-319318cefe0f/usr/libexec/ghostalldirs' はディレクトリではありません
/usr/bin/install: 'build/fix-header' を stat できません: そのようなファイルやディレ
make[2]: *** [install-mkheaders] Error 1
make[2]: Leaving directory '/home/user01/downloads/gcc-4.1.2/build/gcc'
make[1]: ***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

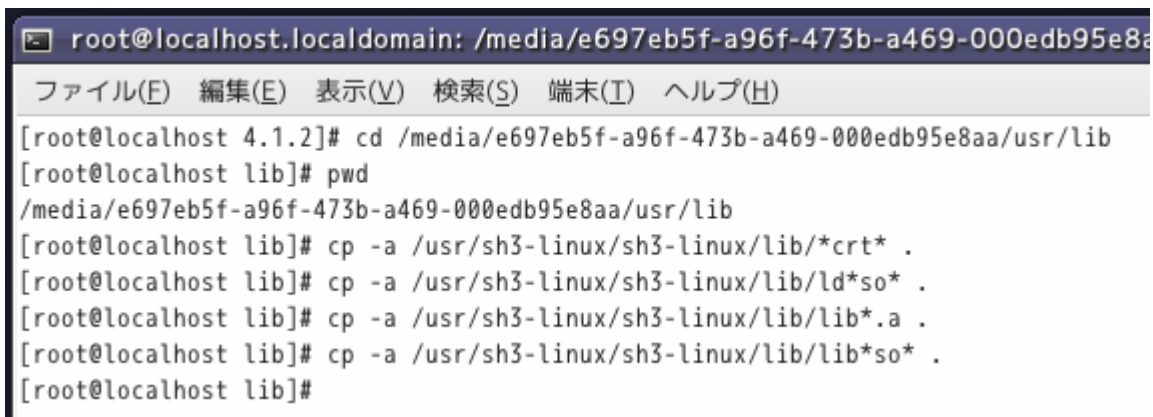
クロスコンパイラとセルフコンパイラの生成(21)

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。



```
root@localhost.localdomain: /media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/lib/gcc
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost usr]# cd /media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/lib/gcc/sh3-linux/4.1.2/
[root@localhost 4.1.2]# pwd
/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/lib/gcc/sh3-linux/4.1.2
[root@localhost 4.1.2]# cp -a /usr/sh3-linux/lib/gcc/sh3-linux/4.1.2/lib*.a .
```

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。



```
root@localhost.localdomain: /media/e697eb5f-a96f-473b-a469-000edb95e8aa
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost 4.1.2]# cd /media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/lib
[root@localhost lib]# pwd
/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/lib
[root@localhost lib]# cp -a /usr/sh3-linux/sh3-linux/lib/*crt* .
[root@localhost lib]# cp -a /usr/sh3-linux/sh3-linux/lib/ld*so* .
[root@localhost lib]# cp -a /usr/sh3-linux/sh3-linux/lib/lib*.a .
[root@localhost lib]# cp -a /usr/sh3-linux/sh3-linux/lib/lib*so* .
[root@localhost lib]#
```

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。

```
root@localhost.localdomain: /media/e697eb5f-a96f-473b-a469-000edb95e8aa
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost lib]# cd /media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/bin
[root@localhost bin]# pwd
/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr/bin
[root@localhost bin]# ln -s c++ cc
[root@localhost bin]# ln -s g++ gcc
[root@localhost bin]# ln -s sh3-linux-c++ sh3-linux-cc
[root@localhost bin]# ln -s sh3-linux-g++ sh3-linux-gcc
[root@localhost bin]#
```

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。

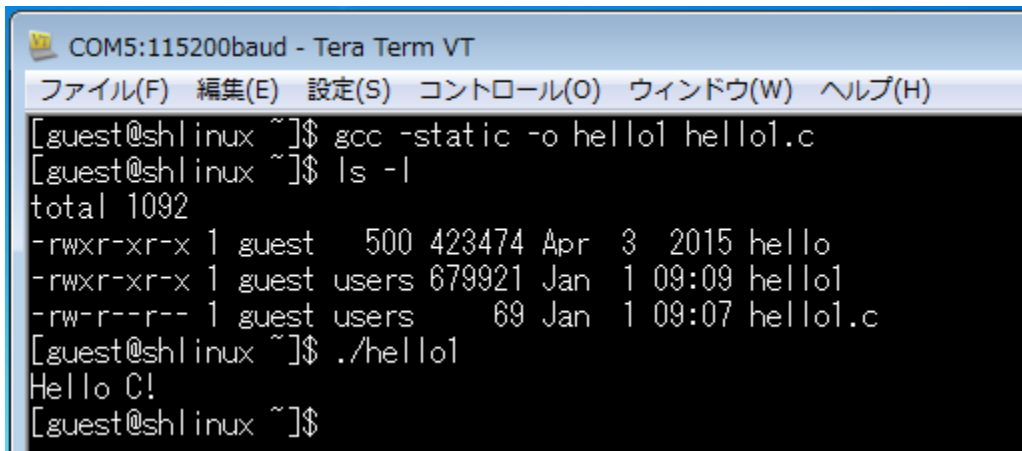
```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O)
[guest@shlinux ~]$ vi hello1.c
```

```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィン
#include <stdio.h>

int main(){
    printf("Hello C! \n");
    return 0;
}
```

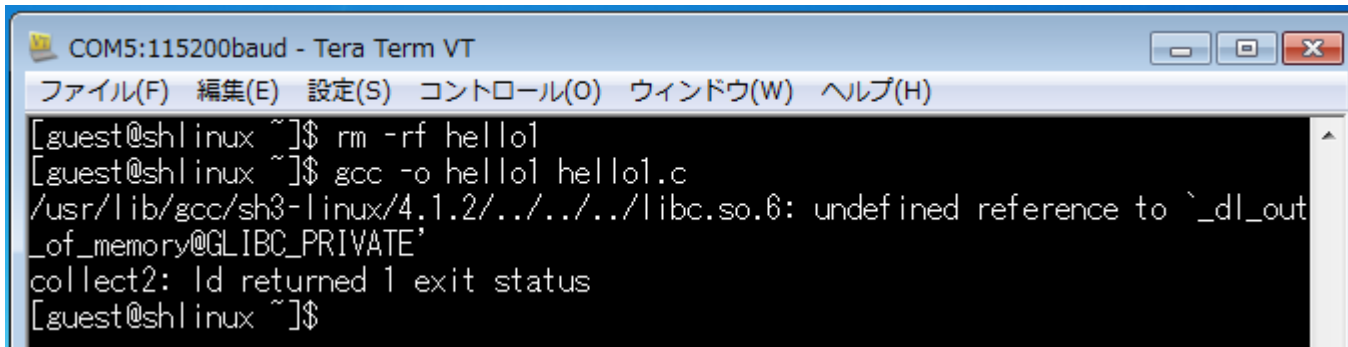
クロスコンパイラとセルフコンパイラの生成(23)

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。



```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[guest@shlinux ~]$ gcc -static -o hello1 hello1.c
[guest@shlinux ~]$ ls -l
total 1092
-rwxr-xr-x 1 guest 500 423474 Apr  3 2015 hello
-rwxr-xr-x 1 guest users 679921 Jan  1 09:09 hello1
-rw-r--r-- 1 guest users 69 Jan  1 09:07 hello1.c
[guest@shlinux ~]$ ./hello1
Hello C!
[guest@shlinux ~]$
```

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。



```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[guest@shlinux ~]$ rm -rf hello1
[guest@shlinux ~]$ gcc -o hello1 hello1.c
/usr/lib/gcc/sh3-linux/4.1.2/../../../../lib/libc.so.6: undefined reference to `_dl_out_of_memory@GLIBC_PRIVATE'
collect2: ld returned 1 exit status
[guest@shlinux ~]$
```

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。

```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[guest@shlinux ~]$ su
Password:
[root@shlinux guest]# cd /usr/lib
[root@shlinux lib]# mv libc.so.6 libc.so.6.default
[root@shlinux lib]#
```

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。

```
COM5:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
[root@shlinux lib]# exit
exit
[guest@shlinux ~]$ gcc -o hello1 hello1.c
[guest@shlinux ~]$ ls -l
total 432
-rwxr-xr-x 1 guest 500 423474 Apr 3 2015 hello
-rwxr-xr-x 1 guest users 5595 Jan 1 10:12 hello1
-rw-r--r-- 1 guest users 69 Jan 1 09:07 hello1.c
[guest@shlinux ~]$ ./hello1
Hello C!
[guest@shlinux ~]$
```

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。

```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[user01@localhost ~]$ ls ./ダウンロード
autoconf-2.68.tar.gz  libtool-2.4.tar.gz  make-3.82.tar.gz
automake-1.11.tar.gz  m4-1.4.15.tar.gz
[user01@localhost ~]$
```

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。

```
user01@localhost.localdomain: /home/user01
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[user01@localhost ~]$ mkdir GNU-std
[user01@localhost ~]$ mv ./ダウンロード/* ./GNU-std
[user01@localhost ~]$ ls ./GNU-std/
autoconf-2.68.tar.gz  libtool-2.4.tar.gz  make-3.82.tar.gz
automake-1.11.tar.gz  m4-1.4.15.tar.gz
[user01@localhost ~]$
```

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。

```
user01@localhost.localdomain: /home/user01/GNU-std
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[user01@localhost ~]$ cd GNU-std/
[user01@localhost GNU-std]$ pwd
/home/user01/GNU-std
[user01@localhost GNU-std]$ ls
autoconf-2.68.tar.gz  libtool-2.4.tar.gz  make-3.82.tar.gz
automake-1.11.6.tar.gz  m4-1.4.15.tar.gz
[user01@localhost GNU-std]$
```

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。

```
user01@localhost.localdomain: /home/user01/GNU-s
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[user01@localhost GNU-std]$ su
パスワード:
[root@localhost GNU-std]# tar xvf make-3.82.tar.gz
```

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(E) 編集(E) 表示(V) 検索(S) 端末(I) 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Makefile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(E) 編集(E) 表示(V) 検索(S) 端末(I) 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

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

getopt1.o implicit.o job.o main.o misc.o read.o remake.o remote-stub.o r
.o version.o vpath.o hash.o -lrt
make[2]: Leaving directory `/home/user01/GNU-std/make-3.82/build'
make[1]: Leaving directory `/home/user01/GNU-std/make-3.82/build'
[root@localhost build]#
```

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

```
root@localhost.localdomain: /home/user01/GNU-std/make-3.82/build
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[root@localhost build]# make install prefix=/media/e697eb5f-a96f-473b-a469-000edb95e8aa/usr

.....
test -z "/media/0ca3397f-3ac3-4430-9cb5-6f9dcc47f830/usr/share/man/man1" || /bin/m
ac3-4430-9cb5-6f9dcc47f830/usr/share/man/man1"
/usr/bin/install -c -m 644 ../make.1 '/media/0ca3397f-3ac3-4430-9cb5-6f9dcc47f830
make[2]: Leaving directory `/home/user01/GNU-std/make-3.82/build'
make[1]: Leaving directory `/home/user01/GNU-std/make-3.82/build'
[root@localhost build]#
```

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。

```
user01@localhost.localdomain: /home/user01/GNU-std
ファイル(E) 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 libtool-2.4.tar.gz
```


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。

```
user01@localhost.localdomain: /home/user01/GNU-std
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost GNU-std]# cd libtool-2.4
[root@localhost libtool-2.4]# mkdir build
[root@localhost libtool-2.4]# cd build
[root@localhost build]# pwd
/home/user01/GNU-std/libtool-2.4/build
[root@localhost build]#
```

GNU標準パッケージの移植(6)

11. 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/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(T) ヘルプ(H)
[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(E) 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(E) 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
```



```
user01@localhost.localdomain: /home/user01/GNU-std/libtool-2.4/build
ファイル(E) 編集(E) 表示(V) 検索(S)
[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]#
```

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
```



```
user01@localhost.localdomain: /home/user01/GNU-std/libtool-2.4/build
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost build]# 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。

```
user01@localhost.localdomain: /home/user01/GNU-std
ファイル(E) 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 automake-1.11.6.tar.gz
```

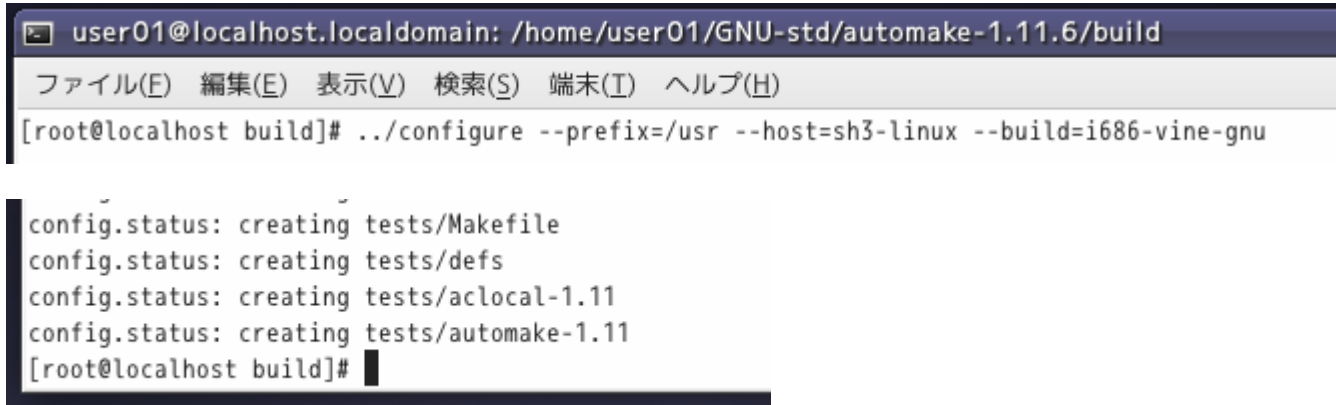
18. 展開したフォルダにbuildフォルダを作成し、buildに移動する。

```
user01@localhost.localdomain: /home/user01/GNU-std/autom
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost GNU-std]# cd automake-1.11.6
[root@localhost automake-1.11.6]# mkdir build
[root@localhost automake-1.11.6]# cd build
[root@localhost build]# pwd
/home/user01/GNU-std/automake-1.11.6/build
[root@localhost build]#
```

GNU標準パッケージの移植(10)

19. 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/automake-1.11.6/build
ファイル(E) 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 tests/Makefile
config.status: creating tests/defs
config.status: creating tests/aclocal-1.11
config.status: creating tests/automake-1.11
[root@localhost build]#
```

20. make する。

```
#make
```



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

Making all in tests
make[1]: Entering directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
[root@localhost build]#
```

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
```

```
root@localhost.localdomain: /home/user01/GNU-std/automake-1.11.6/build
ファイル(E) 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[2]: Nothing to be done for `install-exec-am'.
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
make[1]: Leaving directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
[root@localhost build]#
```

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
```

```
user01@localhost.localdomain: /home/user01/GNU-std/automake-1.11.6/build
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost build]# pwd
/home/user01/GNU-std/automake-1.11.6/build
[root@localhost build]# rm -rf *
[root@localhost build]# ../configure --prefix=/ --host=sh3-linux --build=i686-vine-gnu

config.status: creating tests/defs
config.status: creating tests/aclocal-1.11
config.status: creating tests/automake-1.11
[root@localhost build]#
```


GNU標準パッケージの移植(12)

23. make する。

#make

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

make[1]: Entering directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
[root@localhost build]#
```

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

```
user01@localhost.localdomain: /home/user01/GNU-std/automake-1.11.
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost build]# make install prefix=/usr/sh3-linux/sh3-linux

make[2]: Nothing to be done for `install-exec-am'.
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
make[1]: Leaving directory `/home/user01/GNU-std/automake-1.11.6/build/tests'
[root@localhost build]#
```

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。

```
user01@localhost.localdomain: /home/user01/GNU-std
ファイル(E) 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 autoconf-2.68.tar.gz
```

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。

```
user01@localhost.localdomain: /home/user01/GNU-std/autoconf-
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(T) ヘルプ(H)
[root@localhost GNU-std]# cd autoconf-2.68
[root@localhost autoconf-2.68]# mkdir build
[root@localhost autoconf-2.68]# cd build
[root@localhost build]# pwd
/home/user01/GNU-std/autoconf-2.68/build
[root@localhost build]#
```

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(E) 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(E) 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(E) 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(E) 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。

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

32. 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/m4-1.4.15/build
ファイル(E) 編集(E) 表示(V) 検索(S) 端末(I) ヘルプ(H)
[root@localhost build]# pwd
/home/user01/GNU-std/m4-1.4.15/build
[root@localhost build]# ../configure --prefix=/usr --host=i686-vine-gnu

config.status: linking ../GNUmakefile to GNUmakefile
config.status: executing depfiles commands
config.status: executing stamp-h commands
[root@localhost build]#
```

GNU標準パッケージの移植(17)

33. makeする。

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

make[4]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build/tests'
make[3]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build/tests'
make[2]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build/tests'
make[1]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build'
[root@localhost build]#
```

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

```
root@localhost.localdomain: /home/user01/GNU-std/m4-1.4.15/build
ファイル(E) 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[4]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build/tests'
make[3]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build/tests'
make[2]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build/tests'
make[1]: Leaving directory `/home/user01/GNU-std/m4-1.4.15/build'
[root@localhost build]#
```

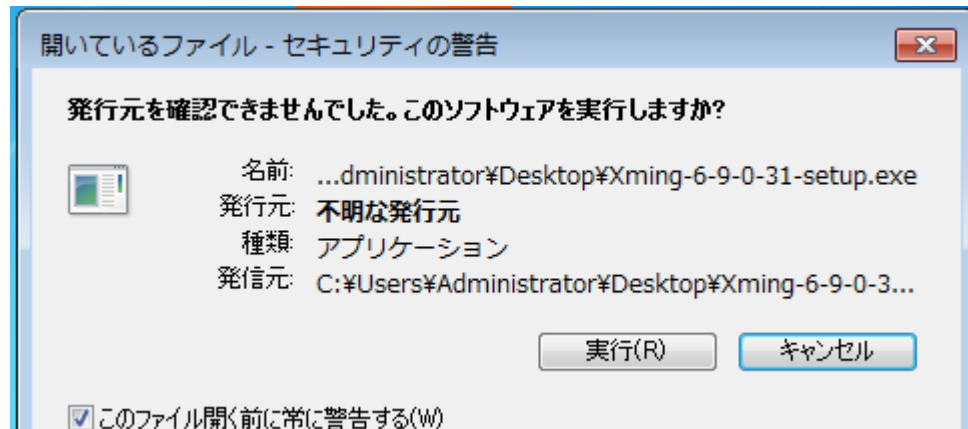
(附1)

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)

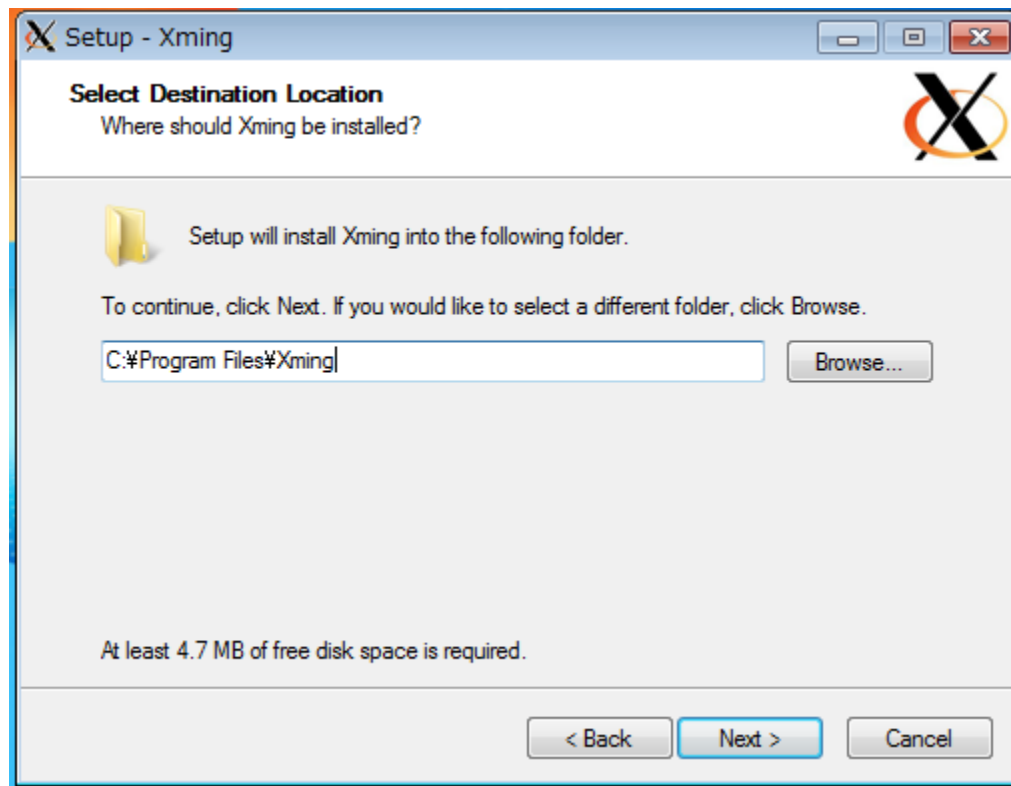
5. 「Next」をクリック



(附1)

Xmingのインストール(3)

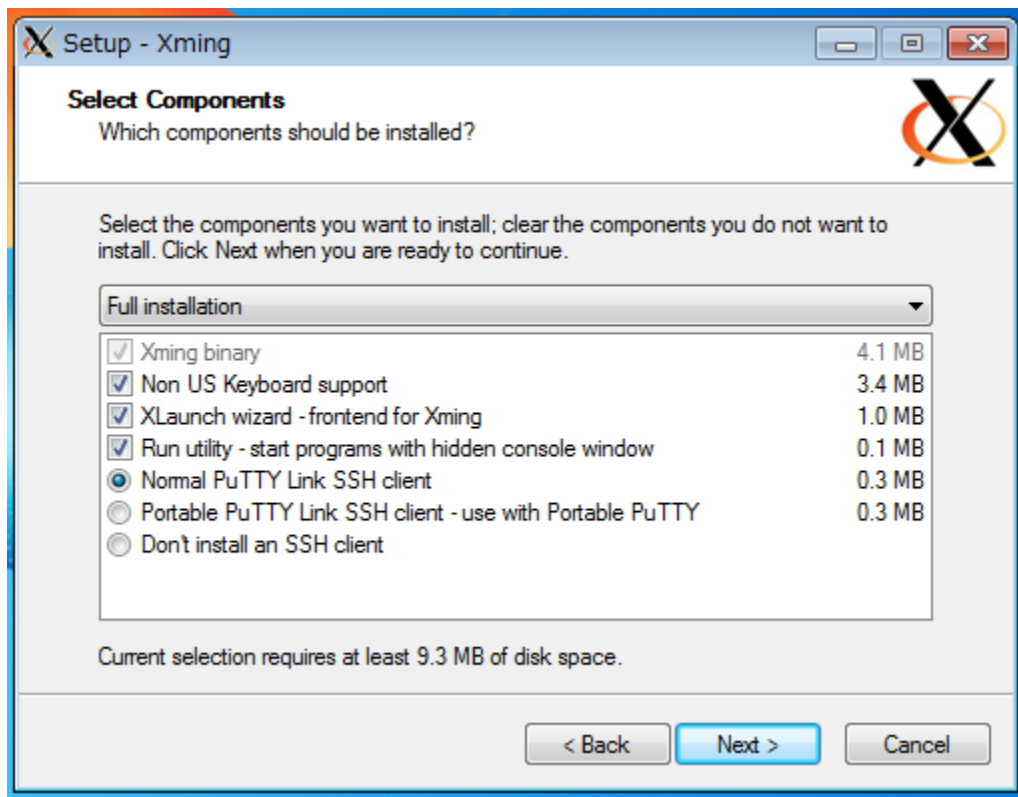
6. 適当なフォルダを指定して「Next」をクリック。ここではデフォルトフォルダ。



(附1)

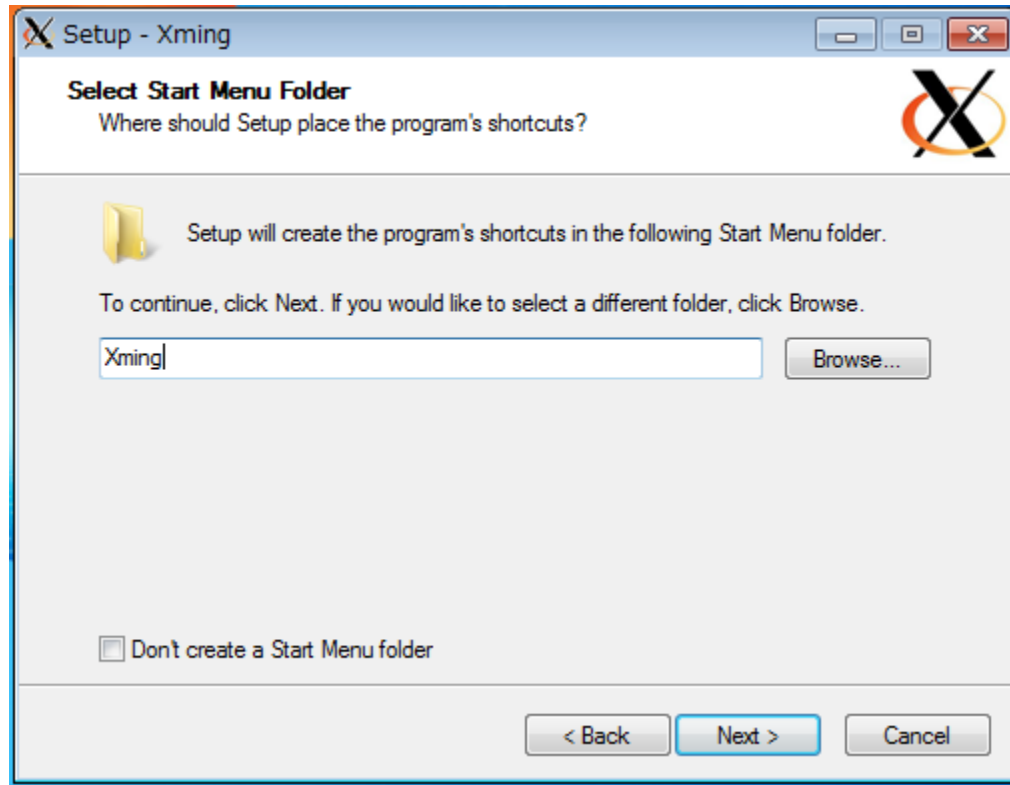
Xmingのインストール(4)

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)

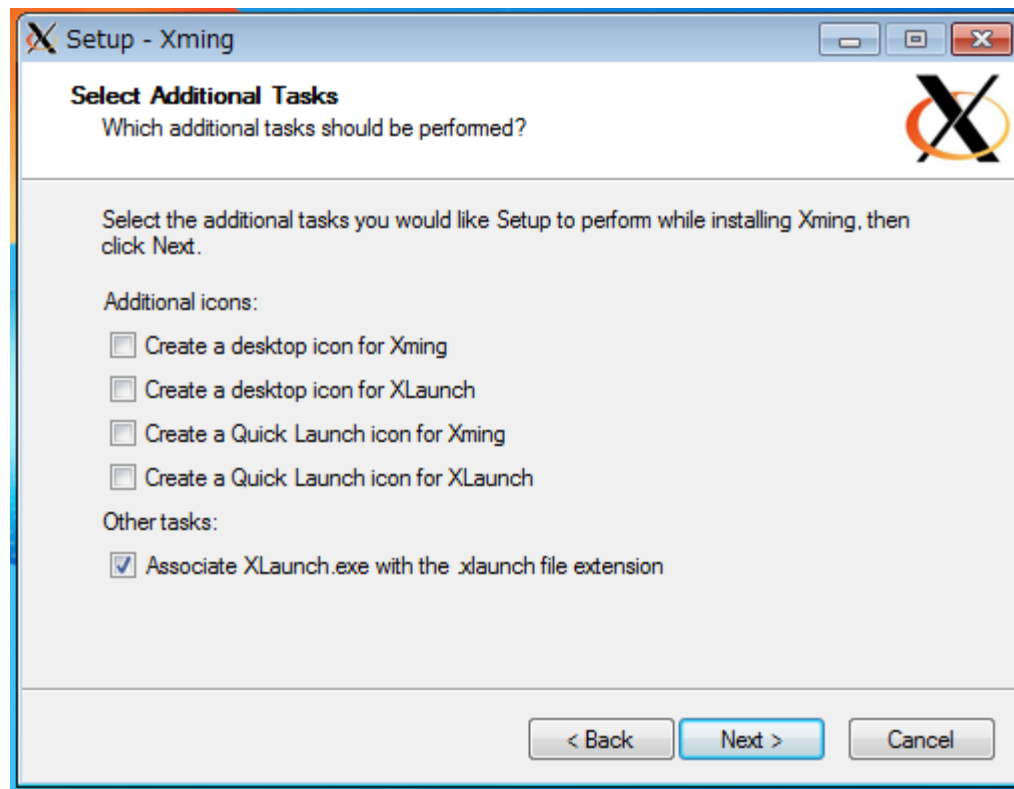
8. 問題なければ、「Next」をクリック



(附1)

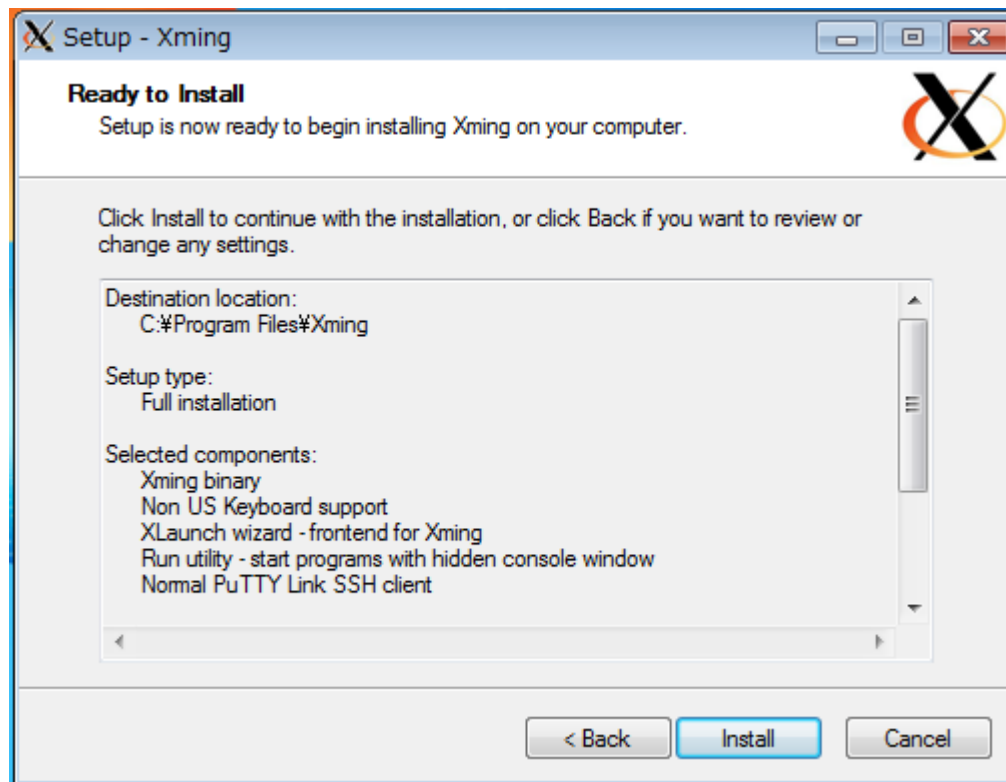
Xmingのインストール(6)

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。

