

# GNU開発ツール

プログラミング環境特論

2007年12月13日

建部修見

# GNUの標準的なbuild手順

- % ./configure
  - % make
  - (% make check)
  - % make install
- 
- 様々な環境に対し, 上記でbuild可能なポータブルなパッケージングのための開発ツール群

# Autoconf

- 多くのUNIXライクなシステムに適応するためにソースコードを自動的にconfigureするスクリプトを生成
- 生成されたスクリプトはそれぞれのfeatureの存在をテストする
- パッケージに必要なシステムのfeatureをリストしたテンプレートファイルからスクリプトを生成
- Autoconfは単体で完結するものではなく, automake やlibtool等他のGNU buildツールと共に利用される
- GNU M4を利用してスクリプトを生成する

# GNU buildシステム

- ポータブルなソフトウェアを開発するためのシステム
  - Autoconf
  - Automake
  - Libtool
  - ...

# Automake: Introduction

- Makefileによるmakeには様々な限界がある
  - 自動的な依存関係の追跡のサポートなし
  - サブディレクトリの再帰的なbuildのサポートなし
  - (NFSなどのための)信頼できるタイムスタンプのサポートなし, . . .
- 典型的に必要なことなのに, 実現するための苦労が大, エラーが生じやすい
- Makeはシステムにより様々であり, ポータビリティは自明ではない
- make install, make distcleanなど想定されるターゲットを準備する必要がある
- AutoconfのためにMakefile.inに@CC@, @CFLAGS@などconfigureにより置換されるコードを挿入する必要がある

# Automake: Introduction (2)

- Buildに必要なことをMakefile.amに記述
  - Makefileに比べ途方もなく簡単で、より強力な文法
- Autoconfで利用される、ポータブルなMakefile.inを生成する
- Makefile.amの例

```
bin_PROGRAMS = hello
```
- 全ての標準target, autoconfによる置換, 自動依存追跡, VPATHなどをサポートした500行余りのMakefile.inが生成される

# Libtool

- 共有ライブラリの作成は各システムでさまざま
  - 共通ではないツール
  - 共通ではないコンパイルフラグ
  - Magic numberの呪文
  - 様々なsuffix
- Libtoolはポータブルに共有ライブラリをbuildするためのツール
- Libtoolは単独でも利用可能だが、automakeとともに利用するととても簡単に利用できる

# References

- Autoconf web page  
<http://www.gnu.org/software/autoconf/>
- Automake web page  
<http://www.gnu.org/software/automake/>
- Libtool web page  
<http://www.gnu.org/software/libtool/>
- Autoconf macro archive  
<http://www.gnu.org/software/ac-archive/>
- Info autoconf, ...



# Configureスクリプト

- Autoconfが生成するconfigureスクリプトは通常以下のファイルを生成する
  - それぞれのサブディレクトリのMakefile
  - Configureの結果を反映したCのヘッダファイル
  - 上記を生成するconfig.statusスクリプト
  - Configureの結果を保存しているconfig.cacheスクリプト
  - ログファイルのconfig.log

# Configure.ac

- Configureを生成するため、パッケージが必要な、また利用可能なfeatureをテストするautoconfマクロの呼出を含んだファイル
- 多くの既存マクロ
  - プラットフォーム, アーキテクチャ
  - コンパイラの引数
  - ライブラリの存在, ヘッダの存在, 引数の型
- カスタムマクロ
- Autoscanによりconfigure.acの雛形が生成される

# Autoconf言語(1)

- プレインテキストと実際のコードが同等に扱われる
- マクロの呼出し
  - マクロ名と(の間に空白はあってはならない
  - 引数はM4のクオート[と]で囲まれ, 区切りはコンマ
  - クオートされない限り引数内の空白は無視される
  - 単純な単語の場合, クォートは外せる
  - 以下は二つとも正しい例

```
AC_CHECK_HEADER([stdio.h],  
                [AC_DEFINE([HAVE_STDIO_H]),  
                [AC_MSG_ERROR([Sorry, can't do anything for you]])])
```

```
AC_CHECK_HEADER(stdio.h,  
                [AC_DEFINE(HAVE_STDIO_H)],  
                [AC_MSG_ERROR([Sorry, can't do anything for you]])])
```

# Autoconf言語(2)

- プレインテキストもマクロのように扱われる
  - echo “Hard rock was here! –[AC\_DC]”はecho “Hard rock was here! –AC\_DC”となる
  - マクロの引数ではなくても[]でクオートする必要がある
- マクロの引数の場合は、マクロの展開でクオートが一つはずれるため、二重のクオートが必須な場合がある

# Autoconf言語(3)

- 以下は誤り.

```
AC_COMPILE_IFELSE([char b[10];],, [AC_MSG_ERROR([you lose])])
```

- AC\_COMPILE\_IFELSEの第一引数が char b[10];となるため, 展開されてchar b10;となる
- 正しくは, 第一引数を二重にクオートする

```
AC_COMPILE_IFELSE([[char b[10];]],, [AC_MSG_ERROR([you lose])])
```

- 間違いやすいため, 常に二重にクオートしておくが良い

# Autoconf言語(4)

- マクロはオプションな引数をとることがある
  - [ARG]と記述される
- []をexplicitに与えても良いし, 空でも良いし, 省略しても良い

```
AC_CHECK_HEADERS(stdio.h, [], [], [])  
AC_CHECK_HEADERS(stdio.h,,,)   
AC_CHECK_HEADERS(stdio.h)
```

- コメントは#ではじめる. 行頭でなくても良い

```
# Process this file with autoconf to produce a configure script.
```

# Configure.acの標準レイアウト

- AC\_INITで始まり, AC\_OUTPUTで終わる
- マクロの呼出しには依存関係があることがある
- その他については順番は重要ではないが, 右のような順番が推奨されている

```
Autoconf requirements
`AC_INIT(PACKAGE, VERSION, BUG-
REPORT-ADDRESS)'  
information on the package  
checks for programs  
checks for libraries  
checks for header files  
checks for types  
checks for structures  
checks for compiler characteristics  
checks for library functions  
checks for system services  
`AC_CONFIG_FILES([FILE...])'  
`AC_OUTPUT'
```

# Autoscan

- ディレクトリツリーのソースを調べ、ポータビリティの問題がありそうな点をまとめて、configure.scanファイルを生成する
- Configure.scanを元にconfigure.acを作成
  - マクロの呼出順序が正しくない場合がある
  - Configurationヘッダファイル使いたい場合、AC\_CONFIG\_HEADERSを追加する
  - ソースコードに#ifを追加するなどしてconfigurationヘッダファイルを利用する
- Configure.scanによりconfigure.acを維持
  - autoscan.logになぜ必要かなどの情報がある



# Ifnames

- Cプリプロセッサで使われているidentifierを表示する
- Configureで何を調べればいいのか調べる
- コマンドラインで指定されたCのソースを全て調べ、`#if`, `#elif`, `#ifdef`, `#ifndef`で利用されているidentifierとファイル名のリストを表示する

# Autoconf

- `configure.ac`から`configure`を作成する
  - Autoconfマクロを利用し, M4マクロプロセッサにより生成
- Autoconfマクロはいくつかのファイルで定義される
  - Autoconfと共に配布されるもの(まずこれを読む)
  - 上記ディレクトリの`acsite.m4`
  - カレントディレクトリの`aclocal.m4`
- 複数の定義がある場合, 後で読む方が有効

# Autoreconf

- 生成されたconfigurationファイルを更新する
    - autoconf, autoheader, aclocal, automake, autopoint, libtoolizeを繰り返し呼び、GNU Buildシステムのファイルを生成する
  - --install, -i
    - Buildシステムに必要なファイルをコピーする
  - --force, -f
    - 新しいバージョンのBuildシステムを導入する
- ```
% autoreconf -vfi
```

# Configure.acにおける初期化

- AC\_INITをまず呼び、初期化を行う
- Macro: AC\_INIT (PACKAGE, VERSION, [BUG-REPORT], [TARNAME])
  - PACKAGEとVERSIONはconfigure --versionなどでも利用される
- 以下のM4マクロが定義される
  - AC\_PACKAGE\_{NAME,TARNAME,VERSION,STRING,BUGREPORT}
- 以下の変数, プリプロセッサシンボルが定義される
  - PACKAGE\_{NAME,TARNAME,VERSION,STRING,BUGREPORT}

# ソースディレクトリの確認

- Macro: AC\_CONFIG\_SRCDIR (UNIQUE-FILE-IN-SOURCE-DIR)
- UNIQUE-FILE-IN-SOURCE-DIRはパッケージのソースディレクトリのファイル
- 正しいディレクトリでconfigureが実行されているかの確認のため

# ファイルの出力

- `configure.ac`は`AC_OUTPUT`の呼出で終了する必要がある
- Macro: `AC_OUTPUT`
  - `config.status`を生成し, `Makefile`などを生成するために`config.status`を実行する

# Configurationアクション

- Configureはシステムを調べ, config.statusを生成する
- Config.statusはファイル生成など様々なアクションを起こす
- AC\_CONFIG\_FILES, AC\_CONFIG\_HEADERS,  
AC\_CONFIG\_COMMANDS, AC\_CONFIG\_LINKS
- Macro: AC\_CONFIG\_FOOS(TAG..., [COMMANDS], [INIT-CMDS])
- TAG...は空白で区切られたタグのリスト. 普通は生成されるファイル名のリスト
- TAGとしてはliteralを利用することが望ましい
- AC\_CONFIG\_FILES, AC\_CONFIG\_HEADERSではOUTPUT:INPUTというTAGが利用可能. 省略時はOUTPUT.inが仮定される
  - AC\_CONFIG\_FILES(a:b:c)ではbとcを結合がINPUTとなる

# Configurationファイルの生成

- Macro: AC\_CONFIG\_FILES (FILE..., [CMDSD], [INIT-CMDSD])
- AC\_OUTPUT時に, FILEを(デフォルトでは FILE.inから)変数を置換して生成する
- 例

```
AC_CONFIG_FILES([Makefile src/Makefile man/Makefile X/Imakefile])
AC_CONFIG_FILES([autoconf], [chmod +x autoconf])
```

```
AC_CONFIG_FILES([Makefile:boiler/top.mk:boiler/bot.mk]
                 [lib/Makefile:boiler/lib.mk])
```



# Makefileにおける置換

- ConfigureはMakefile.inの@VARIABLE@をconfigureが決定した値に置換する
- 置換される変数をoutput variablesという
- Output variablesはconfigureによりshellの変数としても定義される
- 特定の変数をconfigureにより置換するためにはAC\_SUBSTマクロを利用する

# プリセットoutput variables

- 典型的な変数はあらかじめ置換される
- Variable: CFLAGS, CPPFLAGS, CXXFLAGS, DEFS, FCFLAGS, FFLAGS, LDFLAGS, LIBS
- Variable: configure\_input
  - Configureで自動的に生成されたよというコメント
  - # @configure\_input@
- Variable: ECHO\_C, ECHO\_N, ECHO\_T

# インストールディレクトリ

- インストールされるディレクトリの変数
- Variable: prefix, exec\_prefix, bindir, datadir, includedir, infodir, libdir, libexecdir, localstatedir, mandir, oldincludedir, sbindir, sharedstatedir, sysconfdir
- ほとんどの変数はprefixあるいはexec\_prefixを参照している
- Configureで指定されたprefixと違うprefixをmake install時に指定できる

# Buildディレクトリ

- 複数のアーキテクチャでのコンパイルをサポートするためにソースディレクトリではないディレクトリでbuildする仕組み
- GNU makeはVPATH変数を利用してこの仕組みを実現
- 以下を全てのMakefile.inに追加し

```
srcdir = @srcdir@  
VPATH = @srcdir@
```

- ソースファイルの参照には\$(srcdir)/をつける

# Configurationヘッダファイル

- 多くのCプリプロセッサのシンボル定義がある場合、コンパイラのコマンドラインの-Dオプションでは渡せない
  - コマンドラインが長くなりエラーの発見が難しくなる
  - OSによってはコマンドラインの長さの限界を超えてしまう
- #defineディレクティブでシンボル定義をしたヘッダファイルをAC\_CONFIG\_HEADERSで生成する
- ソースでは(生成したヘッダをconfig.hとして)どのヘッダファイルより前に#include <config.h>でインクルードする

# AC\_CONFIG\_HEADERS

- Macro: AC\_CONFIG\_HEADER (HEADER..., [CMDS], [INIT-CMDS])
- HEADERを生成し, @DEF@を-DHAVE\_CONFIG\_Hで置換する
- HEADERがすでに存在し, 変更がない場合は何もしない
  - 無駄な再コンパイルを防ぐ

# Configurationヘッダのテンプレート

- unistd.hがあるかどうか
  - ソースコード
- チェックする例

- configure.ac

```
AC_CONFIG_HEADERS([conf.h])
AC_CHECK_HEADERS([unistd.h])
```

- conf.h.in

```
/* Define as 1 if you have unistd.h. */
#undef HAVE_UNISTD_H
```

```
#include <conf.h>
```

```
#if HAVE_UNISTD_H
# include <unistd.h>
#else
```

```
/* We are in trouble. */
#endif
```

# Autoheader

- Autoheaderはconfigurationヘッダファイルのテンプレートを生成する
- AC\_CONFIG\_HEADERS(FILE)と指定されている場合FILE.inが生成される
- デフォルトはconfig.h.in
- AC\_CHECK\_HEADERS, AC\_CHECK\_LIBSなどのBuiltinのテスト以外は第3引数付きのAC\_DEFINEかAC\_DEFINE\_UNQUOTEDで指定されたシンボルが用いられる
  - Macro: AC\_DEFINE (VARIABLE, VALUE, [DESCRIPTION])
  - Macro: AC\_DEFINE (VARIABLE)



# 利用可能なテスト

- 標準的なシンボル
  - AC\_DEFINEされるシンボルはテストの引数による
  - 大文字にして, \*はPに, それ以外は\_に変換
  - AC\_CHECK\_TYPES(struct \$Expensive\*)では, 成功した場合, HAVE\_STRUCT\_\_EXPENSIVEPが定義される

# 利用可能なテスト(2)

- プログラム
  - Macro: AC\_PROG\_INSTALL, AC\_PROG\_RANLIB, ...
- ファイル
  - Macro: AC\_CHECK\_FILE (FILE, [ACTION-IF-FOUND], [ACTION-IF-NOT-FOUND])
- ライブラリ
  - Macro: AC\_CHECK\_LIB (LIBRARY, FUNCTION, [ACTION-IF-FOUND], [ACTION-IF-NOT-FOUND], [OTHER-LIBRARIES])
    - [ACTION-IF-FOUND]が指定されなければ, -llibraryをLIBSに追加し, HAVE\_LIBLIBRARYを定義する
- 関数
  - Macro: AC\_CHECK\_FUNC (FUNCTION, [ACTION-IF-FOUND], [ACTION-IF-NOT-FOUND])

# Autoconfの例(1)

- Hello.cを作成
- Makefile.inの作成
  - Cコンパイラのプログラム, フラグの設定
  - TARGETとOBJSの設定

```
# @configure_input@  
  
CC = @CC@  
CFLAGS = @CFLAGS@  
CPPFLAGS = @CPPFLAGS@  
DEFS = @DEFS@  
LDFLAGS = @LDFLAGS@  
LIBS = @LIBS@
```

```
TARGET = hello  
OBJS = hello.o  
  
all: $(TARGET)  
  
$(TARGET): $(OBJS)  
  
clean:  
    rm -f $(OBJS)
```

# Autoconfの例(2)

- Autoscanでconfigure.scanを作成
- Configure.scanをconfigure.acにコピーして編集
  - AC\_INITの編集
- Autoheaderによりconfig.h.inを作成
- Autoconfでconfigureを作成

# Automake

- Makefile.amからMakefile.inを生成
- Makefileの仕様は複雑かつ変更されやすいため、Makefileの維持、作成を簡単にする
- autoconfの利用が前提、configure.acに若干の制限
- automakeはperlが必要であるが、automakeを利用したdistributionにはperlは必要ない

# strictness

- GNUのconventionにどれくらい従うか
  - foreign
    - Buildに最低限必要なものだけチェックする。例えばGNU標準パッケージで必要とされるNEWSファイルなどはなくてもよい
  - gnu
    - 可能な限りGNUの標準パッケージ構成に従う
  - gnits
    - まだドキュメントされていないGnitsの標準に従う(Gnits標準のコントリビュータ以外には推奨されない)

# Uniform naming scheme

- Buildされるものを示す変数はprimaryと呼ばれる
  - PROGRAMS = cpio pax
  - コンパイル, リンクされるプログラムのリスト
- インストール先はprefixで指定できる
  - sbin\_PROGRAMS = fsck
  - \$(prefix)/sbinディレクトリにインストールされるプログラムのリスト
- EXTRA\_により選択的に必要なものを指定する
- Primary names
  - PROGRAMS, LIBRARIES, LISP, PYTHON, JAVA, SCRIPTS, DATA, HEADERS, MANS, TEXINFOS

# Derived variables

- 利用者により指定された名前から派生する変数名
  - PROGRAMSで指定されたプログラム名  
+\_SOURCES
  - PROGRAMS = foo bar
  - foo\_SOURCES = foo.c baz.c
- 名前が英数字と@以外のときは全て\_とする
  - PROGRAMS = sniff-glue
  - sniff\_glue\_SOURCES = sniff.c glue.c



# ユーザのために予約された変数名

- CFLAGS, LDFLAGS, CC, ... はユーザのために予約されている
- 例えば, 必要なインクルードパス, ライブラリパスを指定してconfigureを実行
  - `% CFLAGS=-I/usr/local/include %`
  - `LDFLAGS=-L/usr/local/lib ./configure`
- 指定するためにはAM\_のついたshadow variableを利用する
  - `AM_CFLAGS=-I/usr/local/include`

# Automakeの例

- Main.c, hello.c, world.cを作成
- Makefile.amの作成

```
bin_PROGRAMS = main
main_SOURCES = main.c hello.c world.c
```

- Autoscanでconfigure.scanを作成
- Configure.scanをconfigure.acにコピーして編集
  - AC\_INITの編集
  - AM\_INIT\_AUTOMAKE([foreign])を追加

- Autoheaderを実行してconfig.h.inを生成
- Aclocalを実行してaclocal.m4を生成
  - Automakeに必要なM4マクロ
- Autoconfでconfigureを生成
- AutomakeでMakefile.inとそのほか必要なファイルの作成
  - % automake -a
- 上記のaclocal, autoheader, autoconf, automakeの呼出はautoreconfで自動的に行われる
  - % autoreconf -i