Linux 操作系统及应用
附录 C — C 语言编程

唐晓晟　李亦农
txs@bupt.edu.cn　hoplee@bupt.edu.cn

Beijing University of Posts and Telecommunications (BUPT)
School of Information and Communication Engineering

大纲

大纲

大纲

1. 使用 gcc
2. 使用 GNU make 管理项目
3. 使用 Autotools 创建可跨平台编译的软件

大纲

大纲

大纲

gcc 特性

GCC (GNU Compiler Collection，是 GNU 计划提供的编译器家族。

- 支持多种语言，C/CPP/Object C/Fortran/Java/Ada 等
- 支持多种平台，x86/x86-64/IA-64/PowerPC/SPARC/Alpha 等
- 代码编译质量好，效率非常高
- 编译过程 - 预处理、编译、链接
- 支持风格 - ANSI.C、C++、Objective C
- 调试信息 - 能够在生成调试信息的同时进行优化
- 交叉编译 - 能够编译出另一个平台上的可执行代码
- 对 C 和 C++ 进行了大量的扩展（降低了可移植性）

一个简单示例

Hello, world!

```
#include <stdio.h>

int main(int argc, char* argv[])
{
  printf("Hello, world!\n");
  return 0;
}
```

- 编译和运行：
  - gcc hello.c -o hello
  - ./hello

编译过程

编译过程和相关参数

- 过程
  - cpp 预处理所有的宏、展开头文件
  - 编译生成目标代码
  - 使用 ld 命令，链接生成二进制文件
- 常用参数
  - -E 只进行预处理，例如：gcc -E hello.c -o hello.cpp
  - -x 让 gcc 从指定步骤开始编译，例如：gcc -x cpp-output -c hello.cpp hello.o
  - -c 只编译，不链接，例如：gcc -c hello.c -o hello.o
  - -o 指定输出文件名称，例如：
    - gcc hello.o -o hello
    - gcc test2.c test.c -o test

gcc 对扩展名的默认解释

扩展名和相应类型

扩展名	类型
.c	C 语言源代码
.C, .cc	C++ 语言源代码
.i	预处理后的 C 源代码
.ii	预处理后的 C++ 源代码
.S, .s	汇编语言代码
.o	编译后的目标代码
.a, .so	编译后的库代码

Table: 文件名后缀和相应类型

常用命令行选项 I

- -o，FILE 指定输出文件名，未制定时，输出文件为 a.out
- -c，只编译，不链接
- -DFOO=BAR，编译时定义预处理宏 FOO=BAR
- -IDIR，将 DIR 指定的目录添加到头文件搜索路径中
- -LDIR，将 DIR 指定的目录添加到库文件的搜索路径中，缺省情况下 gcc 只链接共享库
- -static，链接静态库（或者说静态链接方式）
- -lFOO，链接名为 FOO 的函数库，例如：-lmath，编译时连接器会在相应目录查找 libmath.so（动态链接）或 libmath.a（静态链接）文件进行链接
- -g，在可执行文件中包含调试信息，共三种级别（1-3），默认级别为 2

常用命令行选项 II

- -ggdb，在可执行程序中包含只有 GNU Debugger（也即，gdb）才能识别的大量调试信息，级别范围和默认级别同上
- -p，加入 Prof 程序能够读取的剖析符号信息
- -pg，在代码中加入 gprof 能够识别的符号信息
- -a，在代码中加入代码块（例如函数）累计使用的次数
- -save-temps，可以保存在编译过程中生成的中间文件，其中包括目标文件和汇编代码文件
- -Q，让 GCC 显示编译过程中遇到的每个函数，并提供编译器编译每个函数所花时间的剖析信息
- -ON，编译时进行优化 (N 为优化级别)，优化主要针对代码大小和运行速度两方面：
  - -O 或 -O1，普通优化（但是尽量不花费太多编译时间），编译时需更多内存和更多时间

常用命令行选项 III

- -O2，增强优化，激活几乎所有优化选项（除了那些空间和时间之间的折衷类选项），需花费更多时间，使用更大内存
- -O3，最大强度的优化，激活所有选项
- -O0，不进行优化
- -Os，只对代码大小进行优化（激活不增加代码体积的优化选项、激活减小代码体积的选项）

- -ansi，支持 ANSI/ISO C 的标准语法，取消 gnu 的语法扩展中与该标准有冲突部分（但这一选项并不能保证生成 ANSI 兼容的代码）
- -pedantic，允许发出 ANSI/IOS C 标准所列出的所有警告
- -pedantic-errors，允许发出 ANSI/IOS C 标准所列出的所有错误
- -traditional，支持 Kernighan & Ritchie C 语法（用旧式语法定义函数）
- -w，关闭所有警告
- -Wall，发出所有 gcc 能提供的警告

常用命令行选项 IV

- -werror，将警告转化为错误，中止编译
- -v，显示每一步详细信息，方便调试

GCC 常见用法总结 I

- 只编译，不链接
  - gcc -c hello.c
- 指定目标文件名
  - gcc hello.c -o hello
- 建立静态库
  - gcc -c liberr.c -o liberr.o
  - ar liberr.a liberr.o
- 建立动态库
  - gcc -fPIC -g -c liberr.c -o liberr.o
  - gcc -g -shared -Wl,-soname,liberr.so -o liberr.so.1.0.0 liberr.o -lc
  - ln -s liberr.so.1.0.0 liberr.so.1
  - ln -s liberr.so.1.0.0 liberr.so

相应解释：

GCC 常见用法总结 II

- 使用 gcc 的 -fPIC 选项，产生与位置无关的代码并能被加载到任何地址
- 需使用 gcc 的 -shared 和 -soname 选项；-Wl 选项把参数传递给连接器 ld；-lc 选项，连接 C 库，保证可以得到所需的启动（startup）代码，从而避免程序在使用不同的，可能不兼容的 C 库的系统上不能正确运行的情形
- 建立相应的符号链接

大纲

make 简介

- make 是一个工具，可以控制从程序源代码产生可执行文件或其他非源程序文件的过程。
- Make is a tool which controls the generation of executables and other non-source files of a program from the program's source files.
- make 通过一个名为 makefile 的文件获取各种信息，从而指导自己的工作，一般来说，只要编写了代码，就应该编写一个相应的 makefile 文件，来通过 make 工具来进行代码的编译、安装、卸载等各种工作

make 能够做什么

- make 可以使得最终用户在完全不了解软件包详细信息的情形下，对其进行编译和安装，这些详细步骤全部都由 makefile 记录
- 当软件包中某个文件修改时，make 能够基于依赖关系，指出哪些文件需要更新，同时能够以正确的步骤来完成更新，例如，如果用户修改了项目中的某个文件，make 不会重新编译整个项目，而只会编译那些修改影响到的文件
- make 工具不和任何特定语言相关，比方说，我们可以使用 makefile 来控制 C、C++ 项目的编译、安装，也可以用 makefile 来控制 Tex 文件的编译，库文件的生成等
- make 工具不仅仅限于制作软件包，也可以用来安装或者卸载软件包，不夸张的说，使用 make 工具配合创建的 makefile 文件，我们可以做与软件项目相关的所有事情

使用 make 的必要性

- 包含多个源文件的项目在编译时有长而复杂的命令行，可以通过 makefile 保存这些命令行来简化该工作
- make 可以减少重新编译所需要的时间，因为 make 可以识别出哪些文件是新修改的
- make 维护了当前项目中各文件的相关关系，从而可以在编译前检查是否可以找到所有的文件
- make 工具是免费的

什么是 makefile I

- makefile 是一个纯文本文件，其中包含一些规则告诉 make 工具在工作时编译哪些文件以及怎样编译这些文件，每条规则都包含以下内容：

  目标 target，一个，即最终创建的东西

  依赖关系列表 dependencies list，一个或多个，通常是编译目标文件所需的其他文件

  命令 commands，一系列命令，用于从指定的相关文件创建目标文件

- make 执行时，按照顺序查找名为 GNUmakefile、makefile 以及 Makefile 文件作为其配置文件，通常，大家都采用 Makefile 作为 make 工具的默认配置文件名
- Makefile 规则示例：

什么是 makefile II

```
target: dependency dependency [..]
  command
  command
  [..]
```

需要注意的是：command 前面必须是制表符

- 一个更复杂的示例：

```
editor: editor.o screen.o keyboard.o
  gcc -o editor editor.o screen.o keyboard.o
editor.o : editor.c editor.h keyboard.h screen.h
  gcc -c editor.c
screen.o: screen.c screen.h
  gcc -c screen.c
keyboard.o : keyboard.c keyboard.h
  gcc -c keyboard.c
clean:
  rm editor *.o
```

什么是 makefile III

深入了解 makefile I

- 伪目标：上例中的 clean
- 变量说明：

  声明 VARNAME=sometext

  使用 ANOTHER=$(VARNAME)

  递归展开 TOPDIR=/home/young, SRCDIR=$(TOPDIR)/src，则 SRCDIR=/home/young/src
- 避免错误递归展开
  - CC=gcc, CC=$(CC) -O2
  - CC=gcc, CC+= -O2
- 一个使用了变量的 makefile 示例

深入了解 makefile II

```
1  OBJS = editor.o screen.o keyboard.o
2  HDRS = editor.h screen.h keyboard.h
3
4  editor: $(OBJS)
5    gcc -o editor $(OBJS)
6  editor.o : editor.c $(HDRS)
7    gcc -c editor.c
8  screen.o: screen.c screen.h
9    gcc -c screen.c
10 keyboard.o : keyboard.c keyboard.h
11   gcc -c keyboard.c
12 clean:
13   rm editor $(OBJS)
```

- 环境变量：make 会自动读取环境变量并使用
- 自动变量：

  $@ 规则的目标对应的文件名

深入了解 makefile III

$<	规则中的第一个相关文件名
$^	规则中的所有相关文件的列表
$?	规则中日期新于目标的所有相关文件的列表
$(@D)	目标文件的目录部分（如果目标在子目录中）
$(@F)	目标文件的文件名部分

- 预定义变量：

AR	归档维护程序 ar
AS	汇编程序 as
CC	C 编译程序 cc
CPP	C 预处理程序 cpp
RM	文件删除程序 rm –f
其他	ARFLAGS ASFLAGS CPPFLAGS LDFLAGS

- 隐式规则及基于隐式规则的示例

深入了解 makefile IV

```
1 OBJS = editor.o screen.o keyboard.o
2
3 editor: $(OBJS)
4   gcc -o editor $(OBJS)
5
6 clean:
7   rm editor $(OBJS)
```

- 模式规则:

```
1 %.o: %.c
2   $(CC) -c $<
```

- # 开头的行为注释行

额外的 make 命令行选项

参数列表

-f file 指定 makefile 的文件名

-n 打印要执行的命令，但不实际执行

-r 禁止使用所有 make 的内置规则

-d 打印调试信息

-k 某个编译目标失败时，继续编译其他目标

调试 make

使用 -d 参数

输出信息包括：

- 重新编译时需要比较的文件
- 被比较文件以及比较结果
- 需要被重新生成的文件
- make 想要使用的隐含规则
- make 实际使用的隐含规则以及所执行的命令

常见错误信息

错误信息

- No rule to make target “target”. Stop
- “target” is up to date
- Target “target” not remake because of errors. (-k 参数)
- command: Command not found
- Illegal option - option

有用的 makefile 目标

常见目标

all	执行编译应用程序的所有工作
install	从编译出来的二进制文件进行应用程序的安装
uninstall	将安装的软件包卸载
clean	将编译产生的二进制文件（可执行文件和目标文件）删除
distclean	为软件分发做准备工作，删除所有编译出来的二进制文件以及通过./configure 命令产生的 Makefile 相关文件
dist	将程序的源代码和相关文档打包成一个压缩文件，便于分发
test|check	执行与应用程序相关的测试工作
installtest|installcheck	进行安装前的测试工作

大纲

Autotools 简介 I

- Autotools 是一套用于生成可以自动地配置软件工具包，以适应多种 Unix 类系统的 shell 脚本的工具，可以帮助开发者开发能够运行在多种不同平台上的软件
- 除了 Autotools，Larry Wall 的 metaconfig（被 rn, perl, patch...使用）以及 X Window Consortimu 的 imake，都可以实现自动配置功能
- Autotools 工具包中包括：autoconf, automake, libtool 三个工具包
- 没有 Autotools 时，开发人员一般写一个包罗万象的 makefile，并告诉用户如何编辑这个 makefile 文件以适应本机环境
- 由 Autotools 生成的配置脚本在运行的时候与 Autotools 是无关的，就是说配置脚本的用户并不需要拥有 Autotools
- 由 Autotools 生成的配置脚本在运行的时候不需要用户的手工干预；通常它们甚至不需要通过给出参数以确定系统的类型对软件包可能需要的各种特征进行独立的测试

工具包及应用程序说明

Autoconf 用来创建 configure 脚本程序，其主要功能是在编译工作前分析用户系统，比方说：系统中安装了 cc 编译器还是 gcc 编译器？...

- **autoscan** 创建一个初始 configure.ac
- **autoheader** 基于 configure.ac，创建一个模板头文件，以便为 configure 使用
- **autoconf** 基于输入的文件 (configure.ac(in))，产生 configure 文件
- **autoupdate** 更新 configure.ac 文件

Automake 基于 Autoconf 产生的信息，创建 Makefile 文件

- **aclocal** 为 configure.ac 中使用的宏创建 aclocal.m4 文件，以便为 autoconf 使用

libtool 用来创建平台无关的共享库文件，本讲义中不涉及此部分

Autotools 工

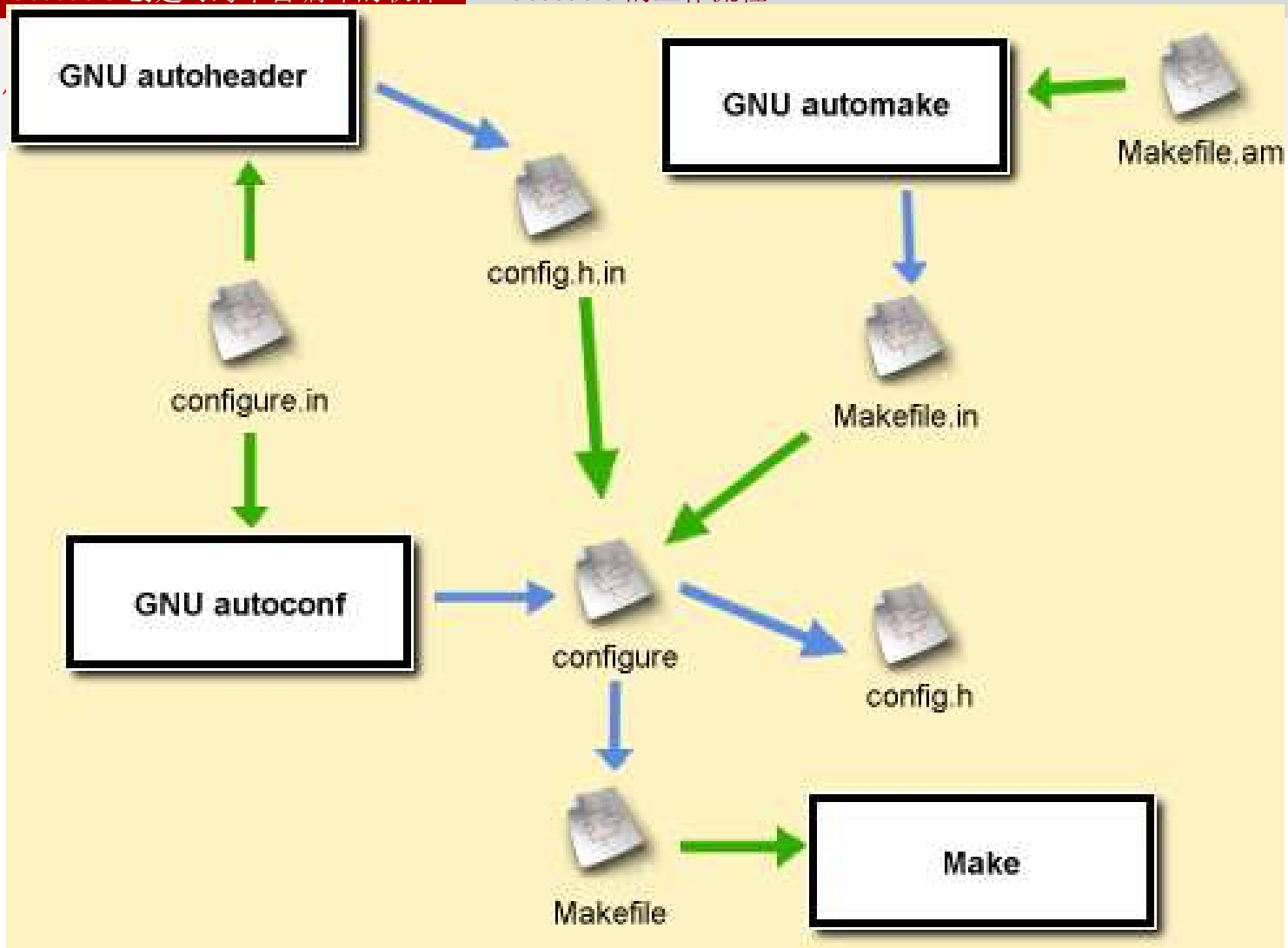

Figure: Autotools 工作流程

Autotools 工

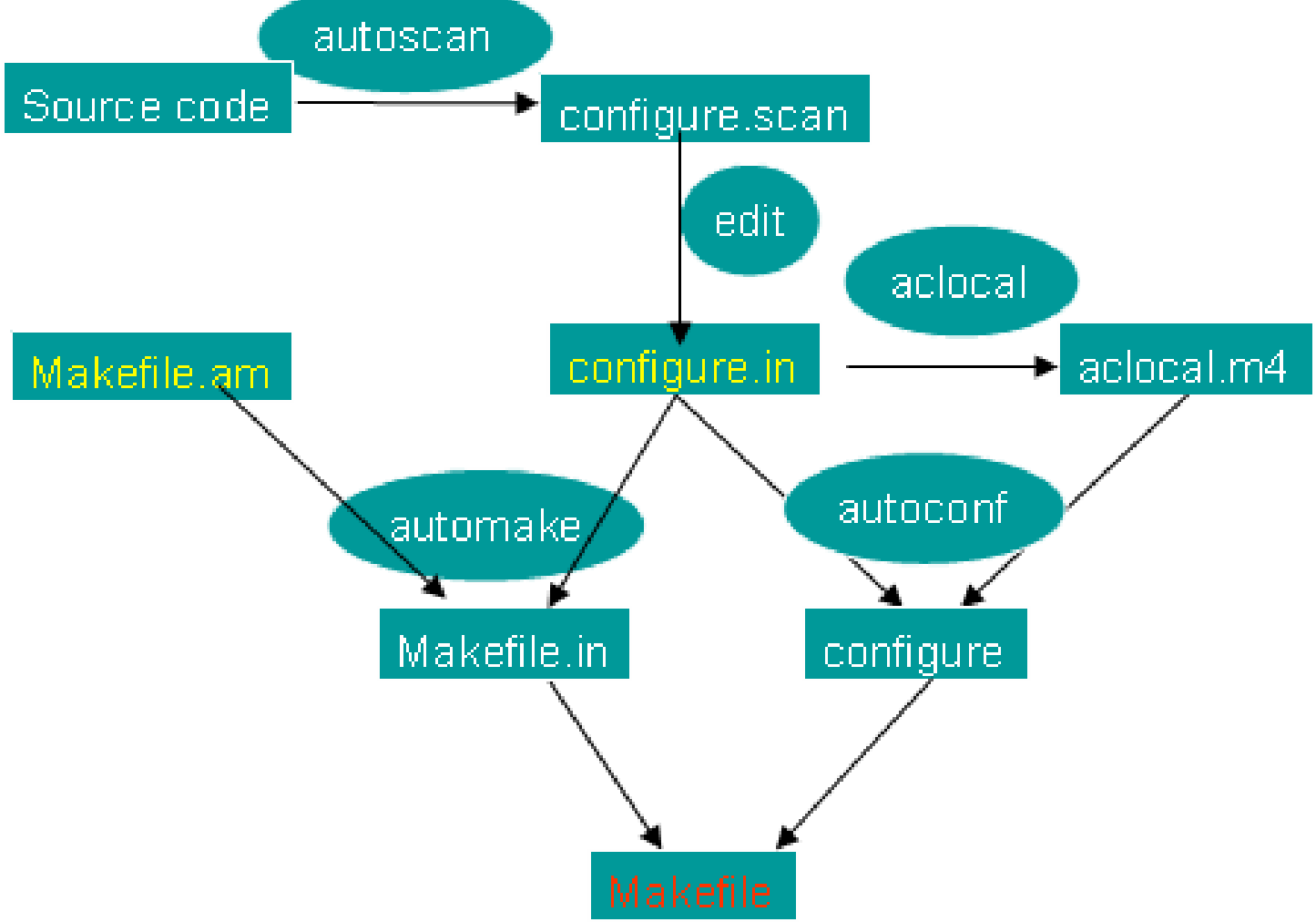

Figure: Makefile 的创建过程

流程描述：

Autotools 工作流程 III

- 准备好源代码，并将其置于工作目录。
- 准备 configure.ac，Makefile.am, aclocal.m4。
  - configure.ac 可以手动创建，也可以由 autoscan 命令生成的模板文件 configure.scan 重命名得到，autoscan 只需在源代码树根目录下运行一次既可。
  - aclocal.m4 由 aclocal 程序生成。
- 必要时，准备 config.h.in，可通过 autoheader 命令生成。
- 准备好 configure.ac 和 aclocal.m4 后，可运行 autoconf 以生成 configure 可执行文件。
- 准备好 Makefile.am 文件后，可运行 automake 生成 Makefile.in 文件
- 最后在编译以前运行一次 configure，检查系统并生成 Makefile，接下来即可 make,make install

实例 1 I

- cpp 源文件

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "config.h"
4
5  int main(int argc, char **argv) {
6      double value=0.0,denom=1.0,sig=1.0;
7      unsigned long t,prec;
8
9    printf("PI Approximator version %s\n",VERSION);
10
11     prec=10;
12     if (argc>1) prec=atol(argv[1]);
13
14     for(t=prec;t;t--) {
15             value+=sig/denom;
16             sig=-sig;
```

实例 1 II

```
17                denom+=2.0;
18        }
19        value*=4.0;
20        printf("pi ~= %.8f , with %lu iterations\n",value,prec);
21        return 0;
22  }
```

- 手工编写的 configure.ac

```
1   AC_INIT(pi.c)
2   AC_CONFIG_HEADER(config.h)
3
4   dnl find and test the C compiler
5   AC_PROG_CC
6   AC_LANG_C
7
8   AC_PROG_MAKE_SET
9
```

实例 1 III

```
10 AC_HEADER_STDC
11 AC_CHECK_FUNC(atol,,AC_MSG_ERROR(oops! no atol ?!?))
12
13 VERSION="0.1"
14 AC_SUBST(VERSION)
15
16 dnl read Makefile.in and write Makefile
17 AC_OUTPUT(Makefile)
```

- 手工编写的 Makefile.in

```
1 CC = @CC@
2 VERSION = @VERSION@
3 CFLAGS = @CFLAGS@
4
5 all: pi-bin
6
7 pi-bin: pi.c
```

实例 1 IV

```
8   $(CC) $(CFLAGS) -DVERSION=\"$(VERSION)\" pi.c -o pi
9
10  clean:
11    rm -f pi
12
13  distclean:
14    rm -f pi config.* Makefile
```

- 利用 autoheader 命令，创建 config.h.in
- 运行 autoconf
- 运行./configure; make 进行测试

autoconf 解读 - configure.ac 文件 I

- configure.ac 文件由 m4 宏处理器进行处理，此外，该文件也是一个标准的 shell 脚本，用户可以在该文件内部写任何合法的 shell 脚本语句
- 该文件内部定义的 shell 变量不会自动的在AC_OUTPUT 指定的文件中进行替代，用户必须显式地调用AC_SUBST 命令进行声明，这个限制只对用户自定义的变量生效
- autoconf 提供的宏都以 AC 开头，automake 提供的宏都以 AM 开头
- autoconf 内部设置变量中，最重要的是 prefix，Makefile.in 文件中通过 @prefix@ 来访问，该变量可以通过运行 configure 命令时指定 –prefix=path 来修改，其默认值是/usr/local，也即：默认安装时，可执行文件安装至/usr/local/bin，库文件安装至/usr/local/lib，手册安装至/usr/local/man...

autoconf 解读 - 常用宏 I

以下是最常见的 autoconf 宏：(需要说明的是，如果用户想扩充宏，可以将扩充内容写入 configure.ac 同一目录下的 acinclude.m4 文件，这样，每次运行 aclocal 命令时，产生的 aclocal.m4 文件中会包括上述自己的内容)

Macro Prototype	Comments
`AC_INIT(sourcefile)`	Initializes autoconf, should be the first macro called in configure.ac. sourcefile is the name (relative to current directory) of a source file from your code.
`AC_PROG_CC`	Determines a C compiler to use, sets the CC variable. If this is GCC, set the GCC variable to 'yes', otherwise 'no'. Initializes the CFLAGS variable if it hasn't been set already (to override CFLAGS, do it in configure.ac BEFORE calling this macro)

autoconf 解读 - 常用宏 II

AC_PROG_CXX	Determines a C++ compiler to use, sets the CXX variable. If this is the GNU C++ compiler, set GXX to 'yes', otherwise 'no'. Initializes the CXXFLAGS variable if it hasn't been set already (to override CXXFLAGS, do it in configure.ac BEFORE calling this macro)
AC_LANG_C	Tests the C compiler
AC_LANG_CPLUSPLUS	Tests the C++ compiler
AC_PROG_INSTALL	Set variable INSTALL to the path of a BSD-compatible install program (see install (1)). If not found, set it to 'dir/install-sh -c' , looking in the directories specified to AC_CONFIG_AUX_DIR. Also sets INSTALL_SCRIPT and INSTALL_PROGRAM to $(INSTALL), and INSTALL_DATA to '$(INSTALL) -m 644'. You must provide a install-sh file in the current directory (unless you use AC_CONFIG_AUX_DIR – the common practice is to provide a install-sh file) else autoconf will refuse to run.

autoconf 解读 - 常用宏 III

AC_PATH_X	Try to locate the X window system's includes and libraries, and sets the variables x_includes and x_libraries to their locations.
AC_PATH_XTRA	Like the previous, but adds the required include flags to X_CFLAGS and required linking flags to X_LIBS.
AC_PATH_PROG(a,b[,c[,d]]) a=variable-name b=prog-to-check c=value-if-not-found d=path	Looks for prog-to-check in PATH, and sets variable-name to the full path if found, or to value-if-not-found if not.
AC_PROG_MAKE_SET	If make predefines the variable MAKE, define output variable SET_MAKE to be empty. Otherwise, define SET_MAKE to contain 'MAKE=make'.

autoconf 解读 - 常用宏 IV

`AC_OUTPUT(files [,a[,b]])`	Create output files. Perform substitutions on files, which contains a list of files separated by spaces. (is writes, say, Makefile, from a Makefile.in file, spec from spec.in, and so on. The name given here is without the .in suffix. The other 2 parameters are seldom used, consult the autoconf docs if needed. If `AC_CONFIG_HEADER`, `AC_LINK_FILES` or `AC_CONFIG_SUBI` were called, the files named as their arguments are created too.
`AC_CONFIG_HEADER(files)`	Make `AC_OUTPUT` create the headers listed in the files list (space-separated). Replaces @DEFS@ in generated files with `-DHAVE_CONFIG_H`. The usual name for the header is config.h (created from config.h.in. The autoheader generates config.h.in files automatically for you (it is documented in the next sections).
`AC_CONFIG_SUBDIRS(dirs)`	run configure scripts in the subdirectories listed in dirs (space-separated). This is meant for when you nest child packages to your program (like including libraries as subdirs).

autoconf 解读 - 常用宏 V

AC_CHECK_FUNC(a[,b[,c]]) a=function b=action if found c=action if not found	checks if the given C function is available in the standard library (i.e., the libraries that are linked by default to any C program).
AC_CHECK_FUNCS(a[,b[,c]]) a=list of functions (space-separated) b=action if found c=action if not found	similar to AC_CHECK_FUNC, but looks for many functions at once, setting HAVE_function for each function found (in the given set).
AC_CHECK_LIB(a,b[,c[,d[,e]]]) a=library name b=function name c=action if found d=action if not found e=see autoconf docs	Checks whether a function exists in the given library (library names without the leading lib, e.g., for libxml, use just xml here)
AC_HEADER_STDC	Checks for stdlib.h, stdarg.h , string.h and float.h, defines STDC_HEADERS on success.
AC_CHECK_HEADER(header[,a[,b]]) a=action if found b=action if not found	Checks whether a given header file exists.

autoconf 解读 - 常用宏 VI

`AC_MSG_ERROR(message)`	Notifies the user an error has occurred and exits configure with a non-zero status. This is what you should do when a required library or header is missing.
`AC_MSG_WARNING(message)`	Notifies the user with the given message. This is what you should do when an optional library is missing (thus the final result will be not as good as it could)
`AC_ARG_ENABLE(feature,help[,a[,b]])` a=action if given b=action if not given	Checks whether the user gave –enable-feature in the configure command-line. The help string is shown in configure –help

autoconf 解读 - 常用宏 VII

autoconf 解读 - autoheader I

- 用户经常需要利用 configure 过程中获取的信息，来对源代码有选择地进行编译（利用#ifdefs 等），以便达到跨平台应用的目的。
- autoconf 宏可以方便地获取平台信息，利用这些信息，用户很容易地可能会如下对源代码进行编译：

```
gcc -g -O2 -Wall -fexpensive-optimizations -DSTDC_HEADERS=1
-DHAVE___clone=0 -DHAVE_localtime=1 -DVERSION=\"0.0.1beta-fb-rc4\"
-DSYSTEM_STRING=\"FreeBSD-5.0-CURRENT-i386\" -DSYSTEM=\"FreeBSD\"
-DPERL_VERSION_MAJOR=5 -DPERL_VERSION_MINOR=6 -DPERL_VERSION_MICRO=0
-I. -I.. -I/usr/X11R6/include -I/opt/include -c gibberish.c
-o gibberish.o
```

- 这种方式很令人不舒服，好的方式是，将所有的平台相关内容，都放入某个头文件，autoconf 在执行过程中更新其内容，使其与平台特性一致，在源程序中，可以根据其具体内容来进行选择性编译

autoconf 解读 - autoheader II

- 为此，需要在 configure.ac 文件中添加AC_CONFIG_HEADER 宏，指明头文件名称（一般为 config.h），并且提供一个 config.h.in 文件，把所有可能的定义统统列出，这样，autoconf 就只需要根据 config.h.in 中的内容做简单操作即可为当前平台生成相应的头文件 config.h
- 通过在 configure.ac 目录运行 autoheader 命令，可以获得相应的 config.h.in 文件
- 如果用户有自己的特殊定义，可以将其写入 acconfig.h 文件，并将该文件置于 configure.ac 所在目录

autoconf 解读 - automake I

- 上面例子，Makefile.in 文件为手工撰写，其中只包含了几个简单的目标
- 实际应用中，我们一般需要更多并且表现一致的目标
- automake 工具可以基于 Makefile.am 来创建 Makefile.in 文件，继而，Makefile.in 文件作为 configure 的输入文件，创建 Makefile 文件

实例 2 I

- cpp 源文件

```
1  /* hello.c: A program to show the time since the Epoch */
2
3  #include <stdio.h>
4  #include "config.h"
5
6  #ifdef HAVE_SYS_TIME_H
7  #include <sys/time.h>
8  #else
9  #include <time.h>
10 #endif
11
12
13 double get_sec_since_epoch()
14 {
15     double sec;
16
```

实例 2 II

```
17      #ifdef HAVE_GETTIMEOFDAY
18          struct timeval tv;
19
20          gettimeofday(&tv, NULL);
21          sec = tv.tv_sec;
22          sec += tv.tv_usec / 1000000.0;
23      #else
24          sec = time(NULL);
25      #endif
26
27      return sec;
28  }
29
30
31  int main(int argc, char* argv[])
32  {
33      printf("%f\n", get_sec_since_epoch());
34
```

实例 2 III

```
35      return 0;
36  }
```

- configure.ac 文件内容，注意，在AC_INIT 后调用了AM_INIT_AUTOMAKE，用AM_CONFIG_HEADER 替代了原来的AC_CONFIG_HEADER

```
1   #                                               -*- Autoconf
2   # Process this file with autoconf to produce a configure scri
3
4   AC_PREREQ(2.61)
5   AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
6   AM_INIT_AUTOMAKE(hello,1.0)
7
8   AC_CONFIG_SRCDIR([hello.c])
9   #AC_CONFIG_HEADER([config.h])
10  AM_CONFIG_HEADER(config.h)
11
```

实例 2 IV

```
12 # Checks for programs.
13 AC_PROG_CC
14
15 # Checks for libraries.
16
17 # Checks for header files.
18 AC_CHECK_HEADERS([sys/time.h])
19
20 # Checks for typedefs, structures, and compiler characteristi
21 AC_HEADER_TIME
22
23 # Checks for library functions.
24 AC_CHECK_FUNCS([gettimeofday])
25
26 AC_CONFIG_FILES([Makefile])
27 AC_OUTPUT
```

实例 2 V

- 依次运行 aclocal, autoheader, autoconf，其功能如下（前面已经介绍过）：

  aclocal 根据 configure.ac 文件内容，创建 aclocal.m4 文件
  autoheader 根据 configure.ac 文件内容，创建 config.h.in 文件
  autoconf 根据 configure.ac 和 aclocal.m4 文件内容，创建 configure 脚本

- 准备 Makefile.am 文件

```
1 bin_PROGRAMS=hello
2 hello_SOURCES=hello.c
```

- Makefile.am 中可以使用的变量：
  - man_MANS，需要安装的手册页文件
  - noinst_HEADERS，软件分发包中必须包含，但是用 make install 命令不安装的头文件
  - EXTRA_DIST，额外的、必须添加到分发包中的文件

实例 2 VI

- 运行 automake 命令，此时，automake 会提示缺文件，可以使用 automake –add-missing 命令，让 automake 创建一些需要的文件（其实是从/usr/share/automake 目录拷贝过来，包括：install-sh, mkinstalldirs, missing, INSTALL, COPYING），但是像 NEWS, README, AUTHORS, ChangeLog 之类的文件，还是需要手工创建
- 创建好全部需要的文件后，再次运行 automake
- 测试：./configure; make; make dist; make clean; make distclean

大纲

cmp 命令和 diff 命令

cmp 命令

- 语法：cmp [options] file1 file2
- 功能：比较两个文件，给出差别字符的位置和行号。同时可以设置选项使得 cmp 给出结果时同时显示差别字符
- 参数说明

  -c 显示第一个差别字符

  -l 以十进制方式显示差别字符的位置，并以八进制显示其数值

cmp 命令和 diff 命令 I

diff 命令

- 语法：diff [options] file1 file2
- 输出格式：

无参数 普通格式，仅按顺序显示差别行

-C 上下文输出格式，以一些行作为上下文来显示差别行，以便用户更清楚地知道所比较文件的差别

-U 统一输出格式，修改了上下文格式，取消重复的上下文，并简化输出

补丁操作 I

- 创建补丁
  - diff -c sigrot.1 sigrot.2 > sigrot.patch
  - diff -u sigrot.1 sigrot.2 > sigrot.patch
  - 可以使用 -r 参数来遍历目录
- 应用补丁
  - patch -p0 < sigrot.patch
  - p0 表示指定使用补丁前补丁中所包含的文件名中需要剥离的“/”的重数，-p 则剥离了除最终文件名之外的所有部分
- 取消补丁
  - patch -p0 -R < sigrog.patch

大纲

版本控制 I

- 版本控制即跟踪和管理文件变化的自动过程
- 版本控制系统必须能管理那些在软体计划发展时源代码所做的改变，并维护一个完整的修改历史记录，这样，当问题发生时，能够帮助开发者追溯到以前稳定的版本
- 最老的版本控制系统是 RCS(Revision Control System 修改控制系统)
- 其他类似的 GNU 软件
  - SCCS : Source Code Control System
  - CVS : Concurrent Version System
  - SVN : Subversion

版本控制的必要性

- 也许有一天你对源代码做了关键改动，删除了老的文件并且忘记了所作改动的确切位置
- 同时跟踪关于当前版本，下一版本以及修改过的错误的情况等信息是冗长并且容易出错的事情
- 也许你的同事不经意间修改了你的代码，会使得你不得不在备份磁带上疯狂查找以找回合适的版本

RCS、CVS 以及 SVN

- RCS 是较老的版本控制，一个受 RCS 管制的档案看起来是这样子的 proj1.c,v ，CVS 沿用了一些 RCS 的规定
- CVS 是一个能让很多程式开发者同时做软件开发的强大工具，它使用了 RCS 的档案规定格式但多了一层像应用程式介面的包装，架在 RCS 的上层
- SVN 是最新的版本控制系统，相比 CVS，添加了极其丰富的开发者需要的新特性，可以毫不夸张地说，SVN 是下一代版本控制系统
  - 官方网站，http://subversion.tigris.org
  - SVNBook，http://svnbook.red-bean.com/
  - 中译版，中译版本，或者另一链接
  - 使用 Subversion 进行版本控制

基本术语

术语

Repository	仓库，所有历史代码都存储在这里，一个仓库中，可以存放多个项目的代码
Check out	检出代码，用户从仓库中取出文件时的操作
Check in	也称为 commit 操作，提交操作，指用户向仓库提交文件的操作
Working dir	工作目录，用户将代码从仓库检出时所存放的本地目录
Working file	工作文件，用户工作目录中处于可编辑状态的文件
Lock	锁定，在早期版本控制系统中，当某个用户以编辑为目的检出某个工作文件时，别人就不能同时编辑这个文件。此时，文件由第一个编辑它的人锁定

基本术语

术语

Conflict 冲突，如果两个用户同时对取出的同一个文件分别进行了修改，当第二个用户提交时，会遇到文件修改操作冲突问题，一般需要手工解决

Merge 合并，当冲突发生时，客户手工进行地解除冲突操作

Revision 源文件的一个特定版本，用数字标识。Revision 的编号一般从 1.1 开始，并依次递增

一般情况下，各种版本控制系统都能自动处理各版本的存储、检索、更改日志、存取控制、发行管理、修订标识和自动合并工作。

SVN 简介

Subversion 简介

- 开放源码的全新版本控制系统
- 支持可在本地访问或通过网络访问的数据库和文件系统存储库
- 提供常见的比较、修补、标记、提交、回复和分支功能，还增加了追踪移动和删除的能力
- 支持非 ASCII 文本和二进制数据

服务器架设 I

- Windows 平台
  - 连接http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91
  - 下载 svn-x.x.x-setup.exe(最新版本) 并安装
  - 下载 SVNService.exe，SVNService -install -d -r c:/home/svn, 将 subversion 应用封装成服务
- Linux 平台
  - apt-get install subversion subversion-tools
- 创建仓库（repository），比方说/home/svn
- svnadmin create /home/svn (or C:/home/svn)
- 修改/home/svn/conf/passwd 文件，根据提示信息修改
- 向 subversion 仓库中导入项目，svn import /data/ldap file:///home/svn/ldap，subversion 支持如下协议

服务器架设 II

协议	访问方法
file:///	通过本地磁盘访问。
http://	与 Apache 组合，通过 WebDAV 协议访问。
https://	同上，但支持 SSL 协议加密连接。
svn://	通过 svnserve 服务自定义的协议访问。
svn+ssh://	同上，但通过 SSH 协议加密连接。

Table: Subversion 支持的访问协议

Subversion 常用命令 I

常用命令

- svn list file:///home/svn/ldap，显示存储库内容
- svn co file:///home/svn/ldap，将 ldap 项目检出到本地
- svn ci (commit)，提交本地修改，记得撰写描述信息
- svn update，更新本地存储内容，使其和服务器一致
- svn add file (or dir)，向 svn 仓库增加文件或目录
- svn diff，显示版本差异

Subversion 客户端 I

SVN 客户端

- 命令行方式（Windows 和 Linux 均可），包括：svn、svnadmin、svnlook、svnmerge、svnserve、svnshell、svnsync、svnversion、svnwrap 等
- 图形用户界面方式：
  - tortoiseSVN，最有名的专用图形化 svn 客户端，与 windows 系统集成很好
  - 基于 Eclipse 的 SUBCLIPSE 插件
  - Visual Studio 的 AnkhSVN 插件
  - 很多软件已经内置了 subversion 的支持，例如 NetBeans、IDEA 等

The End

The End of Appendix C.