Autoconf

Creating Automatic Configuration Scripts

Edition 2.13, for Autoconf version 2.13

December 1998

by David MacKenzie and Ben Elliston

目录

本文档由王立翻译。 1999.12.16

译者在此声明：不对任何由译文错误或者对译文的误解承担任何责任。

介绍

```
A physicist, an engineer, and a computer scientist were
discussing the nature of God.  Surely a Physicist, said the
physicist, because early in the Creation, God made Light; and you
know, Maxwell's equations, the dual nature of electro-magnetic
waves, the relativist consequences... An Engineer!, said the
engineer, because before making Light, God split the Chaos into
Land and Water; it takes a hell of an engineer to handle that big
amount of mud, and orderly separation of solids from
liquids... The computer scientist shouted: And the Chaos,
where do you think it was coming from, hmm?

---Anonymous
```

Autoconf是一个用于生成可以自动地配置软件源代码包以适应多种Unix类系统的 shell脚本的工具。由Autoconf生成的配置脚本在运行的时候与Autoconf是无关的， 就是说配置脚本的用户并不需要拥有Autoconf。

由Autoconf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。相反，它们对软件包可能需要的各种特征进行独立 的测试。（在每个测试之前，它们打印一个单行的消息以说明它们正在进行的检测， 以使得用户不会因为等待脚本执行完毕而焦躁。）因此，它们在混合系统或者从各种 常见Unix变种定制而成的系统中工作的很好。没有必要维护文件以储存由各个Unix变种 、各个发行版本所支持的特征的列表。

对于每个使用了Autoconf的软件包，Autoconf从一个列举了该软件包需要的，或者可以 使用的系统特征的列表的模板文件中生成配置脚本。在shell代码识别并响应了一个被 列出的系统特征之后，Autoconf允许多个可能使用（或者需要）该特征的软件包共享该特征。 如果后来因为某些原因需要调整shell代码，就只要在一个地方进行修改； 所有的配置脚本都将被自动地重新生成以使用更新了的代码。

Metaconfig包在目的上与Autoconf很相似，但它生成的脚本需要用户的手工干预，在配置一个 大的源代码树的时候这是十分不方便的。不象Metaconfig脚本，如果在编写脚本时小心谨慎， Autoconf可以支持交叉编译（cross-compiling）。

Autoconf目前还不能完成几项使软件包可移植的工作。其中包括为所有标准的目标自动 创建`Makefile'文件，包括在缺少标准库函数和头文件的系统上提供替代品。 目前正在为在将来添加这些特征而工作。

对于在C程序中的`#ifdef`中使用的宏的名字，Autoconf施加了一些限制 （参见预处理器符号索引）。

Autoconf需要GNU `m4`以便于生成脚本。它使用了某些UNIX版本的`m4` 所不支持的特征。它还会超出包括GNU `m4` 1.0在内的某些`m4`版本的内部限制。你必须使用GNU `m4`的1.1版或者更新的版本。使用1.3版或者更新的版本将比1.1 或1.2版快许多。

关于从版本1中升级的详情，参见从版本1中升级。 关于Autoconf的开发历史，参见Autoconf的历史。 对与Autoconf有关的常见问题的回答，参见关于Autoconf的问题。

把关于Autoconf的建议和bug报告发送到`bug-gnu-utils@prep.ai.mit.edu`。 请把你通过运行`autoconf --version'`而获得的Autoconf的版本号包括在内。

创建`configure`脚本

由Autoconf生成的配置脚本通常被称为`configure`。在运行的时候，`configure` 创建一些文件，在这些文件中以适当的值替换配置参数。由`configure`创建的文件有：

- 一个或者多个`Makefile'`文件，在包的每个子目录中都有一个（参见 Makefile中的替换）；
- 有时创建一个C头文件，它的名字可以被配置，该头文件包含一些`#define`命令 （参见配置头文件）；
- 一个名为`config.status'`的shell脚本，在运行时，它将重新创建上述文件。 （参见重新创建一个配置）；

- 一个名为`config.cache'的shell脚本，它储存了许多测试的运行结果 （参见缓存文件）；
- 一个名为`config.log'的文件，它包含了由编译器生成的许多消息，以便于 在configure出现错误时进行调试。

为了使用Autoconf创建一个configure脚本，你需要编写一个Autoconf的输入文件`configure.in'并且对它运行autoconf。如果你自行编写了特征测试以补充 Autoconf所提供的测试，你可能还要编写一个名为`aclocal.m4'的文件和一个名为`acsite.m4'的文件。如果你使用了包含#define指令的C头文件，你可能 还要编写`acconfig.h'，并且你需要与软件包一同发布由Autoconf生成的文件`config.h.in'。

下面是一个说明了在配置中使用的文件是如何生成的图。运行的程序都标以后缀`*'。 可能出现的文件被方括号(`[]')括起来。autoconf和autoheader 还读取安装了的Autoconf宏文件（通过读取`autoconf.m4'）。

在准备发布软件包的过程中使用的文件:

```
你的源文件 --> [autoscan*] --> [configure.scan] --> configure.in


configure.in --.   .------> autoconf* -----> configure
               +---+
[aclocal.m4] --+   `---.
[acsite.m4] ---'       |
                       +--> [autoheader*] -> [config.h.in]
[acconfig.h] ----.     |
                 +-----'
[config.h.top] --+
[config.h.bot] --'


Makefile.in ------------------------------> Makefile.in
```

在配置软件包的过程中使用的文件:

```
                       .-------------> config.cache
configure* ------------+-------------> config.log
                       |
[config.h.in] -.       v            .-> [config.h] -.
               +--> config.status* -+               +--> make*
Makefile.in ---'                    `-> Makefile ---'
```

编写`configure.in'

为了为软件包创建configure脚本，需要编写一个名为`configure.in' 的文件，该文件包含了对那些你的软件包需要或者可以使用的系统特征进行测试的Autoconf宏的调用。 现有的Autoconf宏可以检测许多特征； 对于它们的描述可以参见现有的测试。 对于大部分其他特征，你可以使用Autconf模板宏以创建定制的测试；关于它们的详情，参见 编写测试。对于特别古怪或者特殊的特征，`configure.in' 可能需要包含一些手工编写的shell命令。程序autoscan可以为你编写`configure.in' 开个好头（详情请参见用autoscan创建`configure.in'）。

除了少数特殊情况之外，在`configure.in'中调用Autoconf宏的顺序并不重要。 在每个`configure.in'中，必须在进行任何测试之间包含一个对AC_INIT的调用， 并且在结尾处包含一个对AC_OUTPUT的调用（参见创建输出文件）。 此外，有些宏要求其他的宏在它们之前被调用，这是因为它们通过检查某些变量在前面设定的值以决定作些什么。 这些宏在独立的说明中给出（参见现有的测试），而且如果没有 按照顺序调用宏，在生成configure时会向你发出警告。

为了提高一致性，下面是调用Autoconf宏的推荐顺序。通常，在本列表中靠后的项目依赖于表中靠前的项目。 例如，库函数可能受到typedefs和库的影响。

```
AC_INIT(file)
checks for programs
checks for libraries
checks for header files
checks for typedefs
```

```
checks for structures
checks for compiler characteristics
checks for library functions
checks for system services
AC_OUTPUT([file...])
```

最好让每个宏调用在`configure.in'中占据单独的一行。大部分宏并不添加额外的新行； 它们依赖于在宏调用之后的新行以结束命令。这种方法使得生成的configure脚本 在不必添加大量的空行的情况下比较容易阅读。在宏调用的同一行中设置shell变量通常是安全的，这是因为shell允许出现没有用新行间隔的赋值。

在调用带参数的宏的时候，在宏名和左括号之间不能出现任何空格。如果参数被m4 引用字符`['和`]'所包含，参数就可以多于一行。如果你有一个长行， 比如说一个文件名列表，你通常可以在行的结尾使用反斜线以便在逻辑上把它与下一行进行连接 （这是由shell实现的，Autoconf对此没有进行任何特殊的处理）。

有些宏处理两种情况：如果满足了某个给定的条件就做什么，如果没有满足某个给定的条件就做什么。 在有些地方，你可能希望在条件为真的情况下作些事，在为假时什么也不作。反之亦然。为了忽略 为真的情况，把空值作为参数*action-if-found*传递给宏。为了忽略为假的情况，可以 忽略包括前面的逗号在内的宏的参数*action-if-not-found*。

你可以在文件`configure.in'中添加注释。注释以m4预定义宏dnl 开头，该宏丢弃在下一个新行之前的所有文本。这些注释并不在生成的configure脚本中 出现。例如，把下面给出的行作为文件`configure.in'的开头是有好处的：

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

用autoscan创建`configure.in'

程序autoscan可以帮助你为软件包创建`configure.in'文件。如果在命令行中给出了目录， autoscan就在给定目录及其子目录树中检查源文件，如果没有给出目录，就在当前目录及其子目录树中 进行检查。它搜索源文件以寻找一般的移植性问题并创建一个文件`configure.scan'，该文件就是软件包 的`configure.in'预备版本。

在把`configure.scan'改名为`configure.in'之前，你应该手工地检查它；它可能需要一些调整。 autoscan偶尔会按照相对于其他宏的错误的顺序输出宏，为此autoconf将给出警告；你需要 手工地移动这些宏。还有，如果你希望包使用一个配置头文件，你必须添加一个对AC_CONFIG_HEADER的调用。 （参见配置头文件）。可能你还必须在你的程序中修改或者添加一些#if 指令以使得程序可以与Autoconf合作。（关于有助于该工作的程序的详情，参见 用ifnames列举条件）。

autoscan使用一些数据文件，它们是随发布的Autoconf宏文件一起安装的，以便当它在包中的源文件中发现 某些特殊符号时决定输出那些宏。这些文件都具有相同的格式。每一个都是由符号、空白和在符号出现时应该输出的Autoconf 宏。以`#'开头的行是注释。

只有在你安装了Perl的情况下才安装autoscan。 autoscan接受如下选项：

`--help`
: 打印命令行选项的概述并且退出。

`--macrodir=dir`
: 在目录*dir*中，而不是在缺省安装目录中寻找数据文件。你还可以把环境变量AC_MACRODIR设置成 一个目录；本选项将覆盖该环境变量。

`--verbose`
: 打印它检查的文件名称以及在这些文件中发现的可能感兴趣的符号。它的输出可能很冗长。

`--version`
: 打印Autoconf的版本号并且退出。

用ifnames列举条件

在为一个软件包编写`configure.in'时，ifnames可以提供一些帮助。它打印出包已经在C预处理条件 中使用的标识符。如果包已经被设置得具备了某些可移植性，该程序可以帮助你找到configure所需要进行 的检查。它可能有助于补足由autoscan生成的`configure.in'中的某些缺陷。 （参见用autoscan创建`configure.in'）。

ifnames扫描所有在命令行中给出的C源代码文件（如果没有给出，就扫描标准输入）并且把排序后的、由 所有出现在这些文件中的#if、#elif、#ifdef或者#ifndef 命令中的标识符列表输出到标准输出中。它为每个标识符输出单独的一行，行中标识符之后是一个由空格分隔的、使用 了该标识符的文件名列表。

ifnames接受如下选项:

--help
-h 打印命令行选项的概述并且退出。
--macrodir=*dir*
-m *dir*
在目录*dir*中，而不是缺省安装目录中寻找Autoconf宏文件。仅仅被用于获取版本号。 你还可以把环境变量AC_MACRODIR设置成一个目录；本选项将覆盖该环境变量。
--version
打印Autoconf的版本号并且退出。

用**autoconf**创建**configure**

为了从`configure.in'生成configure，不带参数地运行程序autoconf。 autoconf用使用Autoconf宏的m4宏处理器处理`configure.in'。 如果你为autoconf提供了参数，它读入给出的文件而不是`configure.in'并且 把配置脚本输出到标准输出而不是configure。如果你给autoconf以参数`-'， 它将从标准输入，而不是`configure.in'中读取并且把配置脚本输出到标准输出。

Autoconf宏在几个文件中定义。在这些文件中，有些是与Autconf一同发布的；autoconf首先读入它们。 而后它在包含了发布的Autoconf宏文件的目录中寻找可能出现的文件`acsite.m4'，并且在当前目录中寻找 可能出现的文件`aclocal.m4'。这些文件可以包含你的站点的或者包自带的Autoconf宏定义（详情请参见 编写宏）。如果宏在多于一个由autoconf读入了的文件中被 定义，那么后面的定义将覆盖前面的定义。

autoconf接受如下参数:

--help
-h 输出命令行选项的概述并且退出。
--localdir=*dir*
-l *dir*
在目录*dir*中，而不是当前目录中寻找包文件`aclocal.m4'。
--macrodir=*dir*
-m *dir*
在目录*dir*中寻找安装的宏文件。你还可以把环境变量AC_MACRODIR设置成 一个目录；本选项将覆盖该环境变量。
--version
打印Autoconf的版本号并且退出。

用**autoreconf**更新**configure**脚本

如果你有大量由Autoconf生成的configure脚本，程序autoreconf可以保留你的一些工作。 它重复地运行autoconf（在适当的情况下还运行autoheader）以重新创建以当前目录为根 的目录树的Autoconf configure脚本和配置头文件。在缺省情况下，它只重新创建那些比对应的`configure.in'或者（如果出现）`aclocal.m4'要旧的文件。由于在文件没有被改变的情况下， autoheader并不改变它的输出文件的时间标记（timestamp）。这是为了使工作量最小化，修改时间标记是不必要的。 如果你安装了新版本的Autoconf，你可以以选项`--force'调用autoreconf而重新创建 *所有*的文件。

如果你在调用autoreconf时给出选项`--macrodir=*dir*'或者 `--localdir=*dir*'，它将把它们传递给autoconf和autoheader （相对路径将被正确地调整）。

在同一个目录树中，autoreconf不支持两个目录作为同一个大包的一部分（共享`aclocal.m4'和 `acconfig.h'），也不支持每个目录都是独立包（每个目录都有它们自己的`aclocal.m4'和 `acconfig.h'）。 如果你使用了`--localdir'，它假定所有的目录都是同一个包的一部分。如果你没有使用 `--localdir'，它假定每个目录都是一个独立的包，这条限制在将来可能被取消。

关于在configure脚本的源文件发生变化的情况下自动地重新创建它们的`Makefile'规则的细节， 参见自动地重新创建。这种方法正确地处理了配置头文件模板的时间标记，但并不 传递`--macrodir=*dir*'或者`--localdir=*dir*'。

autoreconf接受如下选项:

--help
-h 打印命令行选项的概述并且退出。

--force

-f 即使在`configure'脚本和配置头文件比它们的输入文件（`configure.in'，如果出现 了`aclocal.m4'，也包括它）更新的时候，也要重新创建它们。

--localdir=*dir*

-l *dir*

让autoconf和autoheader在目录*dir*中，而不是在每个包含`configure.in' 的目录中寻找包文件`aclocal.m4'和（仅指autoheader）`acconfig.h' （但不包括`*file*.top'和`*file*.bot'）。

--macrodir=*dir*

-m *dir*

在目录*dir*中，而不是缺省安装目录中寻找Autoconf宏文件。你还可以把环境变量 AC_MACRODIR设置成一个目录；本选项将覆盖该环境变量。

--verbose

打印autoreconf运行autoconf（如果适当，还有autoheader） 的每个目录的目录名。

--version

打印Autoconf的版本号并且退出。

初始化和输出文件

Autoconf生成的configure脚本需要一些关于如何进行初始化，诸如如何寻找包的源文件，的信息； 以及如何生成输出文件的信息。本节叙述如何进行初始化和创建输出文件。

寻找configure的输入文件

所有configure脚本在作任何其他事情之前都必须调用AC_INIT。此外唯一必须调用的宏是 AC_OUTPUT（参见创建输出文件）。

宏: AC_INIT (*unique-file-in-source-dir*)

处理所有命令行参数并且寻找源代码目录。*unique-file-in-source-dir*是一些在包的源代码目录中文件； configure在目录中检查这些文件是否存在以确定该目录是否包含源代码。 人们可能偶尔会用`--srcdir'给出错误的目录；这是一种安全性检查。 详情请参见运行configure脚本。

对于需要手工配置或者使用install程序的包来说，虽然在缺省源代码位置在大部分情况 下看起来是正确的，包还是可能需要通过调用AC_CONFIG_AUX_DIR来告诉 configure到那里去寻找一些其他的shell脚本。

宏: AC_CONFIG_AUX_DIR (*dir*)

在目录*dir*中使用`install-sh'、`config.sub'、`config.guess'和 Cygnus configure配置脚本。它们是配置中使用的辅助文件。*dir*既可以是绝对路径， 也可以是相对于`*srcdir*'的相对路径。缺省值是在`*srcdir*'或者 `*srcdir*/..'或者`*srcdir*/../..'中首先找到`install-sh' 的目录。不对其他文件进行检查，以便使AC_PROG_INSTALL不会自动地发布其他辅助文件。它还要 检查`install.sh'，但因为有些make程序包含了在没有`Makefile'的情况下 从`install.sh'中创建`install'的规则，所以那个名字过时了。

创建输出文件

每个Autoconf生成的configure脚本必须以对AC_OUTPUT的调用结尾。它是一个创建 作为配置结果的`Makefile'以及其他一些可能的文件的宏。此外唯一必须调用的宏是AC_INIT （参见寻找configure的输入文件）。

宏: AC_OUTPUT *([file… [, extra-cmds [, init-cmds]]])*

创建输出文件。在`configure.in'的末尾调用本宏一次。参数*file…*是一个以空格分隔的输出文件 的列表；它可能为空。本宏通过从一个输入文件（缺省情况下名为`*file*.in'）中复制，并替换输出 变量的值以创建每个给出的`*file*'。 关于使用输出变量的详情，请参见Makefile中的替换。 关于创建输出变量的详情，请参见设定输出变量。 如果输出文件所在的目录不存在，本宏将创建该目录（但不会创建目录的父目录）。通常，`Makefile'是按照这种 方式创建的，但其他文件，例如`.gdbinit'，也可以这样创建。

如果调用了AC_CONFIG_HEADER、AC_LINK_FILES或者AC_CONFIG_SUBDIRS，本宏 也将创建出现在它们的参数中的文件。

一个典型的对AC_OUTPUT调用如下:

```
AC_OUTPUT(Makefile src/Makefile man/Makefile X/Imakefile)
```

你可以通过在*file*之后添加一个用冒号分隔的输入文件列表以自行设定输入文件名。例如:

```
AC_OUTPUT(Makefile:templates/top.mk lib/Makefile:templates/lib.mk)
AC_OUTPUT(Makefile:templates/vars.mk:Makefile.in:templates/rules.mk)
```

这样做可以使得你的文件名能够被MS-DOS接受，或者可以把模板文件（boilerplate）添加到文件的开头或者结尾。

如果你给出了*extra-cmds*，那么这些命令将被插入到`config.status'中以便在`config.status' 完成了其他的所有处理之后运行*extra-cmds*。如果给出了*init-cmds*，它们就被插入 *extra-cmds*之前，并且在configure中将对它们进行shell变量、命令和反斜线替换。你可以用 *init-cmds*把变量从configure中传递到*extra-cmds*。如果调用了 AC_OUTPUT_COMMANDS，在其中给出的命令将紧贴在由本宏给出的命令之前运行。

宏: AC_OUTPUT_COMMANDS (*extra-cmds [, init-cmds]*)

指定在`config.status'末尾运行的附加的shell命令，以及用于初始化来自于configure 的所有变量的shell命令。本宏可以被调用多次。下面是一个不太实际的例子:

```
fubar=27
AC_OUTPUT_COMMANDS([echo this is extra $fubar, and so on.], fubar=$fubar)
AC_OUTPUT_COMMANDS([echo this is another, extra, bit], [echo init bit])
```

如果你在子目录中运行make，你应该通过使用make变量MAKE来运行它。 make的大部分版本把MAKE设置成make的程序名以及它所需要的任何选项。 （但许多版本并没有把在命令行中设定的变量的值包括进来，因此它们没有被自动地传递。）一些老版本的 make并不设定这个变量。以下的宏使你可以在这些版本上使用它。

宏: AC_PROG_MAKE_SET

如果make预定义了变量MAKE，把输出变量SET_MAKE定义为空。否则，把 SET_MAKE定义成`MAKE=make'。为SET_MAKE调用AC_SUBST。

为了使用这个宏，在每个其他的、运行MAKE的目录中的`Makefile.in'添加一行:

```
@SET_MAKE@
```

Makefiles中的替换

发布版本中每个包含了需要被编译或者被安装的文件的目录都应该含有一个文件`Makefile.in'， configure将利用它在那个目录中创建一个`Makefile'。 为了创建`Makefile'，configure进行一个简单的变量替换：用configure 为`@*variable*@'选取的值，在`Makefile.in'中对它们进行替换。 按照这种方式被替换到输出文件中的变量被称为*输出变量*。在configure中，它们是普通 的shell变量。为了让configure把特殊的变量替换到输出文件中，必须把那个变量的名字作为调用 AC_SUBST的参数。其他变量的任何`@*variable*@'都保持不变。关于 使用AC_SUBST创建输出变量的详情，请参见设定输出变量。

使用configure脚本的软件应该发布文件`Makefile.in'，而不是`Makefile'； 这样，用户就可以在编译它之前正确地为本地系统进行配置了。

关于应该把哪些东西放入`Makefile'的详情，请参见*GNU编码标准*中的`Makefile惯例'。

预定义输出变量

有些输出变量是由Autoconf宏预定义的。一部分Autoconf宏设置一些附加的输出变量，这些变量在对这些宏的描述 中被说明。关于输出变量的完整列表，参见输出变量索引。下面是每个预 定义变量所包含的内容。关于变量名以`dir'结尾的变量，参见*GNU编码标准*中的`为安装目录而提供的变量'。

变量: bindir
用于安装由用户运行的可执行文件的目录。

变量: configure_input
一个用于说明文件是由configure自动生成的，并且给出了输入文件名的注释。 AC_OUTPUT在它创建的每个`Makefile'文件的开头添加一个包括了这个变量的注释行。 对于其他文件，你应该在每个输入文件开头处的注释中引用这个变量。例如，一个输入shell脚本应该以如下 行开头:

```
#! /bin/sh
# @configure_input@
```

这一行的存在也提醒了人们在编辑这个文件之后需要用configure进行处理以使用它。

变量: datadir
用于安装只读的与结构无关的数据的目录。

变量: exec_prefix
与结构有关的文件的安装前缀。

变量: includedir
用于安装C头文件的目录。

变量: infodir
用于安装Info格式文档的目录。

变量: libdir
用于安装目标代码库的目录。

变量: libexecdir
用于安装由其他程序运行的可执行文件的目录。

变量: localstatedir
用于安装可以被修改的单机数据的目录。

变量: mandir
用于安装man格式的文档的顶层目录。

变量: oldincludedir
用于安装由非gcc编译器使用的C头文件的目录。

变量: prefix
与结构无关的文件的安装前缀。

变量: sbindir
用于安装由系统管理员运行的可执行文件的目录。

变量: sharedstatedir
用于安装可以修改的、与结构无关的数据的目录。

变量: srcdir
包含了由`Makefile'使用的源代码的目录。

变量: sysconfdir
用于安装只读的单机数据的目录。

变量: top_srcdir
包的顶层源代码目录。在目录的顶层，它与srcdir相同。

变量: CFLAGS
为C编译器提供的调试和优化选项。如果在运行configure时，没有在环境中设置它，就在你 调用AC_PROG_CC的时候设置它的缺省值（如果你没有调用AC_PROG_CC，它就为空）。 configure在编译程序以测试C的特征时，使用本变量。

变量: CPPFLAGS
为C预处理器和编译器提供头文件搜索目录选项（`-I*dir*'）以及其他各种选项。如果在运行 configure时，在环境中没有设置本变量，缺省值就是空。configure在编译或者预处理 程序以测试C的特征时，使用本变量。

变量: CXXFLAGS
为C++编译器提供的调试和优化选项。如果在运行configure时，没有在环境中设置本变量，那么 就在你调用AC_PROG_CXX时设置它的缺省值（如果你没有调用AC_PROG_CXX，它就为空）。 configure在编译程序以测试C++的特征时，使用本变量。

变量: FFLAGS
为Fortran 77编译器提供的调试和优化选项。如果在运行configure时，在环境中没有设置本变量，那么它的 缺省值就在你调用AC_PROG_F77时被设置（如果你没有调用AC_PROG_F77，它就为空）。 configure在编译程序以测试Fortran 77的特征时，使用本变量。

变量: DEFS
传递给C编译器的`-D'选项。如果调用了AC_CONFIG_HEADER，configure就用 `-DHAVE_CONFIG_H'代替`@DEFS@'（参见配置头文件）。 在configure进行它的测试时，本变量没有被定义，只有在创建输出文件时候才定义。关于如何检查从前的 测试结果，请参见设定输出变量。

变量: LDFLAGS
为连接器提供的Stripping（`-s'）选项和其他各种选项。如果在运行configure时， 在环境中没有设置本变量，它的缺省值就是空。 configure在连接程序以测试C的特征时使用本变量。

变量: LIBS
传递给连接器的`-l'和`-L'选项。

创建目录

你可以支持从一个软件包的一份源代码拷贝中为多种结构同时进行编译的功能。为每种结构生成的目标文件都在 它们自己的目录中储存。

为了支持这个功能，make用变量VPATH来寻找储存在源代码目录中的文件。 GNU make和其他大部分近来的make程序都可以这样做。老版本的make 程序不支持VPATH；在使用它们的时候，源代码必须与目标代码处于同一个目录。

为了支持VPATH，每个`Makefile.in'文件都应该包含下列两行:

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

不要把VPATH设置成其他变量的值，比如说`VPATH = $(srcdir)'，这是因为 某些版本的make并不对VPATH的值进行变量替换。

在configure生成`Makefile'的时候，它用正确的值对srcdir进行替换。

除非在隐含规则中，不要使用make变量$<，它将被展开成到源代码目录的文件 的路径（通过VPATH找到的）。（诸如`.c.o'的隐含规则用于说明如何从`.c' 文件创建`.o'文件）有些版本的make在隐含规则中不设置$<； 它们被展开成空值。

`Makefile'命令行总是应该通过使用前缀`$(srcdir)/'来引用源代码文件。例如:

```
time.info: time.texinfo
        $(MAKEINFO) $(srcdir)/time.texinfo
```

自动地重新创建

你可以在包的顶层目录中的`Makefile.in'文件中添加如下的规则，以使得在你更新了配置文件之后 可以自动地更新配置信息。这个例子包括了所有可选的文件，例如`aclocal.m4'和那些与配置头文件 有关的文件。从`Makefile.in'规则中忽略所有你的所不需要的文件。

因为VPATH机制的限制，应该包含`${srcdir}/'前缀。

在重新创建不改变`config.h.in'和`config.h'的内容的情况下，就不会改变这两个文件的时间标记，因此需要`stamp-'文件。这个特征避免了不必要的重新编译工作。你应该把文件`stamp-h.in' 包含在你的包的发布中，以便make能够把`config.h.in'看作是更新了的文件。在一些 老的BSD系统中，touch或者任何可能导致空文件的命令不会更改时间标记，所以使用诸如echo 之类的命令。

```
${srcdir}/configure: configure.in aclocal.m4
        cd ${srcdir} && autoconf

# autoheader might not change config.h.in, so touch a stamp file.
${srcdir}/config.h.in: stamp-h.in
${srcdir}/stamp-h.in: configure.in aclocal.m4 acconfig.h \
    config.h.top config.h.bot
        cd ${srcdir} && autoheader
        echo timestamp > ${srcdir}/stamp-h.in

config.h: stamp-h
stamp-h: config.h.in config.status
        ./config.status

Makefile: Makefile.in config.status
        ./config.status

config.status: configure
        ./config.status --recheck
```

此外，你应该把`echo timestamp > stamp-h'作为*extra-cmds*参数传递给AC_OUTPUT， 以便`config.status'能够确认`config.h'是更新了的。关于AC_OUTPUT的详情，请参见 创建输出文件。

关于处理与配置相关的依赖性问题的更多例子，请参见重新创建一个配置。

配置头文件

在包测试的C预处理器符号比较多的时候，用于把`-D'传递给编译器的命令行就会变得很长。 这导致了两个问题。一个是通过观察寻找make输出中的错误变得困难了。更严重的是，命令行 可能超过某些操作系统的长度限制。作为把`-D'选项传递给编译器的替代办法，configure 脚本可以创建一个包含了`#define'指令的C头文件。宏AC_CONFIG_HEADER 选择了这种输出。它应该在AC_INIT之后立即调用。

包应该在引入其他任何头文件之前`#include'配置头文件，以防止出现声明中的不一致性 （例如，配置头文件可能重定义了const）。使用`#include <config.h>' 并且把选项`-I.'（或者是`-I..'；或者是任何包含`config.h' 的目录）传递给C编译器，而不是使用`#include "config.h"'。按照这种方式，即使源代码 自行进行配置（可能是创建发布版本），其他创建目录也可以在没有找到`config.h'的情况下， 从源代码目录进行配置。

宏: AC_CONFIG_HEADER (*header-to-create …*)

使得AC_OUTPUT创建出现在以空格分隔的列表*header-to-create*中的文件， 以包含C预处理器#define语句，并在生成的文件中用`-DHAVE_CONFIG_H' ，而不是用DEFS的值，替换`@DEFS@'。常用在*header-to-create* 中的文件名是`config.h'。

如果*header-to-create*给出的文件已经存在并且它的内容和AC_OUTPUT将要生成的 内容完全一致，这些文件就保持不变。这样做就使得对配置的某些修改不会导致对依赖于头文件的目标文件进行 不必要的重新编译。

通常输入文件被命名为`*header-to-create*.in'；然而，你可以通过在*header-to-create* 之后添加由冒号分隔的输入文件列表来覆盖原输入文件名。 例:

```
AC_CONFIG_HEADER(defines.h:defines.hin)
AC_CONFIG_HEADER(defines.h:defs.pre:defines.h.in:defs.post)
```

这样做使得你的文件名能够被MS-DOS所接受，或者可以把模板（boilerplate）添加到文件的开头和/或结尾。

配置头文件模板

你的发布版本应该包含一个如你所望的最终的头文件那样的模板文件，它包括注释、以及#define 语句的缺省值。例如，假如你的`configure.in'进行了下列调用:

```
AC_CONFIG_HEADER(conf.h)
AC_CHECK_HEADERS(unistd.h)
```

那么你就应该在`conf.h.in'中包含下列代码。 在含有`unistd.h'的系统中，configure应该把0改成1。在其他系统中，这一行将保持不变。

```
/* Define as 1 if you have unistd.h.  */
#define HAVE_UNISTD_H 0
```

如果你的代码使用#ifdef而不是#if来测试配置选项，缺省值就可能是取消对一个变量 的定义而不是把它定义成一个值。在含有`unistd.h'的系统中，configure将修改读入的第二行 `#define HAVE_UNISTD_H 1'。在其他的系统中，（在系统预定义了那个符号的情况下） configure将以注释的方式排除这一行。

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

用**autoheader**创建**`config.h.in'**

程序autoheader可以创建含有C的`#define'语句的模板文件以供configure使用。 如果`configure.in'调用了AC_CONFIG_HEADER(*file*)，autoheader就创建 `*file*.in'；如果给出了多文件参数，就使用第一个文件。否则，autoheader就创建 `config.h.in'。

如果你为autoheader提供一个参数，它就使用给出的文件而不是`configure.in'，并且把头文件输出 到标准输出中去，而不是输出到`config.h.in'。如果你把`-'作为参数提供给autoheader ，它就从标准输入中，而不是从`configure.in'中读出，并且把头文件输出到标准输出中去。

autoheader扫描`configure.in'并且找出它可能要定义的C预处理器符号。它从一个名为 `acconfig.h'的文件中复制注释、#define和#undef语句，该文件与Autoconf一同发布 并且一同安装。如果当前目录中含有`acconfig.h'文件，它也会使用这个文件。如果你用AC_DEFINE 定义了任何附加的符号，你必须在创建的那个`acconfig.h'文件中包含附加的符号。对于由AC_CHECK_HEADERS、AC_CHECK_FUNCS、AC_CHECK_SIZEOF或者 AC_CHECK_LIB定义的符号，autoheader生成注释和#undef语句，而不是从一个 文件中复制它们，这是因为可能的符号是无限的。

autoheader创建的文件包含了大部分#define和#undef语句，以及相关的注释。 如果`./acconfig.h'包含了字符串`@TOP@'，autoheader就把在包含`@TOP@' 的行之前的所有行复制到它生成的文件的开头。相似地，如果`./acconfig.h'包含了字符串`@BOTTOM@'， autoheader就把那一行之后的所有行复制到它生成的文件的末尾。这两个字符串的任何一个都可以被忽略， 也可以被同时忽略。

产生相同效果的另一种办法是在当前目录中创建文件`*file*.top'（通常是`config.h.top'）和/或 文件`*file*.bot'。如果它们存在，autoheader就把它们分别复制到它的输出的开头和末尾。 不鼓励使用它们是因为它们的文件名含有两个点，并因此不能在MS-DOS中储存；它们在目录中多创建了两个文件。但如果你给出 选项`--localdir=*dir*'以使用在其他目录中的`acconfig.h'，它们就为你提供了一种把 定制的模板（boilerplate）放入各个独立的`config.h.in'中的方式。

autoheader接受如下选项:

`--help`
`-h`
: 打印对命令行选项的概述并且退出。

`--localdir=`*dir*
`-l` *dir*
: 在目录*dir*中，而不是在当前目录中，寻找包文件`aclocal.m4'和`acconfig.h' （但不包括`*file*.top'和`*file*.bot'）。

`--macrodir=`*dir*
`-m` *dir*
: 在目录*dir*中寻找安装的宏文件和`acconfig.h'。你还可以把环境变量AC_MACRODIR 设置成一个目录；本选项将覆盖该环境变量。

`--version`
: 打印Autoconf的版本号并且退出。

在子目录中配置其它包

在大多数情况下，调用AC_OUTPUT足以在子目录中生成`Makefile'。然而，控制了多于一个 独立包的configure脚本可以使用AC_CONFIG_SUBDIRS来为每个子目录中的其他包运行 configure脚本。

宏: AC_CONFIG_SUBDIRS (*dir …*)

使得AC_OUTPUT在每个以空格分隔的列表中给出的子目录*dir*中运行configure。 如果没有发现某个给出的*dir*，不会作为错误报告，所以一个configure脚本可以配置一个 大的源代码树中出现的任何一个部分。如果在给出的*dir*中包含了`configure.in'，但没有包含 configure，就使用由AC_CONFIG_AUXDIR找到的Cygnus configure脚本。

用与本configure脚本完全相同的命令行参数调用子目录中的configure脚本，如果需要， 会有较小的修改（例如，为缓冲文件或者源代码目录调整相对路径）。本宏还把输出变量subdirs设置成 目录列表`*dir*...'。`Makefile'规则可以使用该变量以确定需要进入那些子目录。 这个宏可以多次调用。

缺省的前缀

在缺省状态下，configure把它所安装的文件的前缀设置成`/usr/local'。 configure的用户可以通过选项`--prefix'和`--exec-prefix'选择一个不同的前缀。 有两种方式修改缺省的行为：在创建configure时，和运行configure时。

有些软件包在缺省情况下可能需要安装到`/usr/local'以外的目录中。为此，使用宏AC_PREFIX_DEFAULT。

宏: AC_PREFIX_DEFAULT (*prefix*)

把缺省的安装前缀设置成*prefix*，而不是`/usr/local'。

对于用户来说，让configure根据它们已经安装的相关程序的位置来猜测安装前缀，可能会带来方便。如果你 希望这样做，你可以调用AC_PREFIX_PROGRAM。

宏: AC_PREFIX_PROGRAM (*program*)

如果用户没有给出安装前缀（使用选项`--prefix'），就按照shell的方式，在PATH中寻找 *program*，从而猜出一个安装前缀。如果找到了*program*，就把前缀设置成包含*program* 的目录的父目录；否则，就不改变在`Makefile.in'中给定的前缀。例如，如果*program*是 gcc，并且PATH包括了`/usr/local/gnu/bin/gcc'，就把前缀设置为 `/usr/local/gnu'。

configure中的版本号

以下的宏为configure脚本管理版本号。使用它们是可选的。

宏: AC_PREREQ (*version*)

确保使用的是足够新的Autoconf版本。如果用于创建configure的Autoconf的版本比*version* 要早，就在标准错误输出打印一条错误消息并不会创建configure。例如:

```
AC_PREREQ(1.8)
```

如果你的`configure.in'依赖于在不同Autoconf版本中改变了的、不明显的行为，本宏就是有用的。 如果它仅仅是需要近来增加的宏，那么AC_PREREQ就不太有用，这是因为程序autoconf已经 告诉了用户那些宏没有被找到。如果`configure.in'是由一个在提供AC_PREREQ之前的更旧的 Autoconf版本处理的，也会发生同样的事。

宏: AC_REVISION (*revision-info*)

把删除了任何美元符或者双引号的修订标记（revision stamp）复制到configure脚本中。 本宏使得你的从`configure.in'传递到configure的修订标记不会在你提交（check in） configure的时候被RCS或者CVS修改。你可以容易地决定一个特定的configure 对应与`configure.in'的哪个修订版。

把本宏放在AC_INIT之前是个好主意，它可以使修订号接近`configure.in'和configure 的开头。为了支持你这样做，AC_REVISION就像configure通常作的那样，以 `#! /bin/sh'开始它的输出。

例如，在`configure.in'中这一行为:

```
AC_REVISION($Revision: 1.30 $)dnl
```

在configure中产生了:

```
#! /bin/sh
# From configure.in Revision: 1.30
```

现有的测试

这些宏测试了包可能需要或者需要使用的特定的系统特征。如果你要测试这些宏所不能测试的特征，可能你可以用 适当的参数调用主测试宏来达到目的（参见编写测试）。

这些宏打印消息以告诉用户它们正在测试的特征，以及它们的测试结果。它们为未来运行的configure 储存测试结果（参见缓存结果）。

在这些宏中，有的宏设置输出变量。关于如何获取它们的值，请参见Makefile中的替换。 在下面出现的术语“定义*name*”是“把C预处理符号*name*定义成1”的简称。 关于如何把这些符号的定义放入你的程序中，参见定义C预处理器符号。

对程序的选择

这些宏检查了特定程序的存在或者特定程序的特征。它们被用于在几个可以相互替代的程序间进行选择，并且在决定选用 某一个的时候作些什么。如果没有为你要使用的程序定义特定的宏，并且你不需要检查它的任何特殊的特征，那么你就 可以选用一个通用程序检查宏。

对特定程序的检查

这些宏检查特定的程序--它们是否存在，并且在某些情况下它们是否支持一些特征。

宏: AC_DECL_YYTEXT

如果yytext的类型是`char *'而不是`char []'，就定义YYTEXT_POINTER。 本宏还把输出变量LEX_OUTPUT_ROOT设置由lex生成的文件名的基文件名；通常是`lex.yy'， 但有时是其他的东西。它的结果依使用lex还是使用flex而定。

宏: AC_PROG_AWK

按顺序查找mawk、gawk、nawk和awk，并且把输出变量AWK 的值设置成第一个找到的程序名。首先寻找mawk是因为据说它是最快的实现。

宏: AC_PROG_CC

确定C的编译器。如果在环境中没有设定CC，就查找gcc，如果没有找到，就使用cc。 把输出变量CC设置为找到的编译器的名字。

如果要使用GNU C编译器，把shell变量GCC设置为`yes'，否则就设置成空。如果还没有设置输出变量 CFLAGS，就为GNU C编译器把CFLAGS设置成`-g -O2'（在GCC不接受`-g' 的系统中就设置成`-O2'），为其他编译器把CFLAGS设置成`-g'。

如果被使用的C编译器并不生成可以在configure运行的系统上运行的可执行文件，就把shell变量 cross_compiling设置成`yes'，否则设置成`no'。换句话说，它检查创建系统类型 是否与主机系统类型不同（目标系统与本测试无关）。关于对交叉编译的支持，参见手工配置。

宏: AC_PROG_CC_C_O

对于不能同时接受`-c'和`-o'选项的C编译器，定义NO_MINUS_C_MINUS_O。

宏: AC_PROG_CPP

把输出变量CPP设置成运行C预处理器的命令。如果`$CC -E'不能工作，就使用`/lib/cpp'。 只有对以`.c'为扩展名的文件运行CPP才是可以移植的（portable）。

如果当前语言是C（参见对语言的选择），许多特定的测试宏通过调用AC_TRY_CPP、 AC_CHECK_HEADER、AC_EGREP_HEADER或者AC_EGREP_CPP，间接地使用了CPP的值。

宏: AC_PROG_CXX

确定C++编译器。检查环境变量CXX或者CCC（按照这个顺序）是否被设置了；如果设置了，就把输出变量 CXX设置成它的值。否则就搜索类似名称（c++、g++、gcc、CC、 cxx和cc++）的C++编译器。如果上述测试都失败了，最后的办法就是把CXX设置成gcc。

如果使用GNU C++编译器，就把shell变量GXX设置成`yes'，否则就设置成空。 如果还没有设置输出变量CXXFLAGS，就为GNU C++编译器把CXXFLAGS设置成`-g -O2'（在G++不接受`-g'的系统上设置成`-O2'），或者为其他编译器把CXXFLAGS设置成`-g'。.

如果使用的C++编译器并不生成在configure运行的系统上运行的可执行文件，就把shell变量cross_compiling 设置成`yes'，否则就设置成`no'。换句话说，它检查创建系统类型是否与主机系统类型不同 （目标系统类型与本测试无关）。关于对交叉编译的支持，参见手工配置。

宏: AC_PROG_CXXCPP

把输出变量CXXCPP设置成运行C++预处理器的命令。如果`$CXX -E'不能工作，使用`/lib/cpp'。 只有对以`.c'、`.C'或者`.cc'为扩展名的文件运行CPP才是可以移植的（portable）。

如果当前语言是C++（参见对语言的选择），许多特定的测试宏通过调用 AC_TRY_CPP、AC_CHECK_HEADER、AC_EGREP_HEADER或者AC_EGREP_CPP， 间接地使用了CXXCPP的值。

宏: AC_PROG_F77

确定Fortran 77编译器。如果在环境中没有设置F77，就按顺序检查g77、f77和 f2c。把输出变量F77设置成找到的编译器的名字。

如果使用g77（GNU Fortran 77编译器），那么AC_PROG_F77将把shell变量G77设置成 `yes'，否则就设置成空。如果在环境中没有设置输出变量FFLAGS，那么就为g77 把FFLAGS设置成`-g -02'（或者在g77不支持`-g'的时候设置成`-O2'）。否则，就为所有其它的Fortran 77编译器把FFLAGS设置成`-g'。

宏: AC_PROG_F77_C_O

测试Fortran 77编译器是否能够同时接受选项`-c'和`-o'，并且如果不能同时接受的话，就 定义F77_NO_MINUS_C_MINUS_O。

宏: AC_PROG_GCC_TRADITIONAL

如果在没有给出`-traditional'的情况下，用GNU C和ioctl不能正确地工作，就把 `-traditional'添加到输出变量CC中。这通常发生在旧系统上没有安装修正了的头文件 的时候。因为新版本的GNU C编译器在安装的时候自动地修正了头文件，它就不是一个普遍的问题了。

宏: AC_PROG_INSTALL

如果在当前PATH中找到了一个与BSD兼容的install程序，就把输出变量INSTALL设置 成到该程序的路径。否则，就把INSTALL设置成`*dir*/install-sh -c'，检查由 AC_CONFIG_AUX_DIR指明的目录（或者它的缺省目录）以确定*dir*（参见 创建输出文件）。本宏还把变量INSTALL_PROGRAM和INSTALL_SCRIPT 设置成`${INSTALL}'，并且把INSTALL_DATA设置成`${INSTALL} -m 644'。

本宏忽略各种已经确认的不能工作的install程序。为了提高速度，它更希望找到一个C程序而不是shell脚本。 除了`install-sh'，它还能够使用`install.sh'，但因为有些make含有一条在没有 `Makefile'的情况下，从`install.sh'创建`install'的规则，所以这个名字过时了。

你可能使用的`install-sh'的一个副本来自于Autoconf。如果你使用AC_PROG_INSTALL，你必须在你的 发布版本中包含`install-sh'或者`install.sh'，否则即使你所在的系统含有一个好的install 程序，configure也将输出一条找不到它们的错误消息。

如果你因为你自己的安装程序提供了一些在标准install程序中没有的特征，而需要使用你自己的安装程序， 就没有必要使用AC_PROG_INSTALL；直接把你的程序的路径名放入你的`Makefile.in'文件即可。

宏: AC_PROG_LEX

如果找到了flex，就把输出变量LEX设置成`flex'，并且在flex库在标准位置的时候， 把LEXLIB设置成`-lfl'。否则，就把LEX设置成`lex'并且把 LEXLIB设置成`-ll'。

宏: AC_PROG_LN_S

如果`ln -s'能够在当前文件系统中工作（操作系统和文件系统支持符号连接），就把输出变量 LN_S设置成`ln -s'，否则就把它设置成`ln'。

如果连接出现在其他目录而不是在当前目录中，它的含义依赖于是使用了`ln'，还是使用了`ln -s'。 为了用`$(LN_S)'安全地创建连接，既可以找到正在使用的形式并且调整参数，也可以总是在创建连接的 目录中调用ln。

换句话说，它不能像下面那样工作:

```
$(LN_S) foo /x/bar
```

而是要:

```
(cd /x && $(LN_S) foo bar)
```

宏: AC_PROG_RANLIB

如果找到了ranlib，就把输出变量RANLIB设置成`ranlib'，否则就设置成 `:'（什么也不作）。

宏: AC_PROG_YACC

如果找到了bison，就把输出变量YACC设置成`bison -y'。 否则，如果找到了byacc。就把YACC设置成`byacc'。否则， 就把YACC设置成`yacc'。

对普通程序和文件的检查

这些宏用于寻找没有包含在特定程序测试宏中的程序。如果你除了需要确定程序是否存在，还需要检测程序的行为，你就 不得不为它编写你自己的测试了（参见编写测试）。在缺省情况下，这些宏使用 环境变量PATH。如果你需要检查可能不会出现在PATH中的程序，你可能要按照下面的方式 给出修改了的路径:

```
AC_PATH_PROG(INETD, inetd, /usr/libexec/inetd,
  $PATH:/usr/libexec:/usr/sbin:/usr/etc:etc)
```

宏: AC_CHECK_FILE (*file [, action-if-found [, action-if-not-found]]*)

检查文件*file*是否出现在本地系统中。如果找到了，就执行*action-if-found*。否则，就在给出了 *action-if-not-found*的时候执行*action-if-not-found*。

宏: AC_CHECK_FILES (*files[, action-if-found [, action-if-not-found]]*)

为每个在*files*中给出的文件运行AC_CHECK_FILE。并且为每个找到的文件定义 `HAVE*file*'，定义成1。

宏: AC_CHECK_PROG (*variable, prog-to-check-for, value-if-found [, value-if-not-found [, path, [reject]]]*)
检查程序*prog-to-check-for*是否存在于PATH之中。如果找到了，就把变量 *variable*设置成*value-if-found*，否则就在给出了*value-if-not-found*的时候 把*variable*设置成它。即使首先在搜索路径中找到*reject*（一个绝对文件名），本宏也会忽略它； 在那种情况下，用找到的*prog-to-check-for*，不同于*reject*的绝对文件名来设置*variable*。 如果*variable*已经被设置了，就什么也不作。为*variable*调用AC_SUBST。

宏: AC_CHECK_PROGS (*variable, progs-to-check-for [, value-if-not-found [, path]]*)
在PATH中寻找每个出现在以空格分隔的列表*progs-to-check-for*中的程序。 如果找到了，就把*variable*设置成那个程序的名字。否则，继续寻找列表中的下一个程序。如果列表 中的任何一个程序都没有被找到，就把*variable*设置成*value-if-not-found*; 如果没有 给出*value-if-not-found*，*variable*的值就不会被改变。为*variable*调用 AC_SUBST。

宏: AC_CHECK_TOOL (*variable, prog-to-check-for [, value-if-not-found [, path]]*)
除了把AC_CANONICAL_HOST确定的主机类型和破折号作为前缀之外，类似于AC_CHECK_PROG， 寻找*prog-to-check-for*（参见获取规范的系统类型）。 例如，如果用户运行`configure --host=i386-gnu'，那么下列调用:

```
AC_CHECK_TOOL(RANLIB, ranlib, :)
```

当`i386-gnu-ranlib'在PATH中存在的时候，就把RANLIB设置成`i386-gnu-ranlib'， 或者当`ranlib'在PATH中存在的时候，就把RANLIB设置成`ranlib'， 或者在上述两个程序都不存在的时候，把RANLIB设置成`:'。

宏: AC_PATH_PROG (*variable, prog-to-check-for [, value-if-not-found [, path]]*)
类似于AC_CHECK_PROG，但在找到*prog-to-check-for*的时候，把*variable*设置 成*prog-to-check-for*的完整路径。

宏: AC_PATH_PROGS (*variable, progs-to-check-for [, value-if-not-found [, path]]*)
类似于AC_CHECK_PROGS，但在找到任何一个*progs-to-check-for*的时候，把*variable* 设置成找到的程序的完整路径。

库文件

下列的宏检查某些C、C++或者Fortran 77库文件是否存在。

宏: AC_CHECK_LIB (*library, function [, action-if-found [, action-if-not-found [, other-libraries]]]*)
依赖于当前的语言（参见对语言的选择），试图通过检查一个测试程序是否可以和 库*library*进行连接以获取C、C++或者Fortran 77函数*function*，从而确认函数*function* 是可以使用的。*library*是库的基本名字；例如，为了检查`-lmp'，就把`mp'作为 参数*library*。

*action-if-found*是一个在与库成功地进行了连接的时候运行的shell命令列表； *action-if-not-found*是一个在与库的连接失败的时候运行的shell命令列表。 如果没有给出*action-if-found*，缺省的动作就是把`-l*library*'添加到 LIBS中，并且定义`HAVE_LIB*library*'（全部使用大写字母）。

如果与*library*的连接导致了未定义符号错误（unresolved symbols），而这些错误可以通过与其他库的连接来解决， 就把这些库用空格分隔，并作为*other-libraries*参数给出: `-lXt -lX11'。否则，本宏 对*library*是否存在的检测将会失败，这是因为对测试程序的连接将总是因为含有未定义符号错误而失败。

宏: AC_HAVE_LIBRARY (*library, [, action-if-found [, action-if-not-found [, other-libraries]]]*)
本宏等价于*function*参数为main的，对AC_CHECK_LIB的调用。 此外，*library*可以写作`foo'、`-lfoo'或者`libfoo.a'。 对于以上任一种形式，编译器都使用`-lfoo'。但是，*library*不能是一个shell变量； 它必须是一个文字名（literal name）。本宏是一个过时的宏。

宏: AC_SEARCH_LIBS (*function, search-libs [, action-if-found [, action-if-not-found [, other-libraries]]]*)
如果*function*还不可用，就寻找一个定义了*function*的库。这等同于首先不带库调用 AC_TRY_LINK_FUNC，而后为每个在*search-libs*中列举的库调用AC_TRY_LINK_FUNC。

如果找到了函数，就运行*action-if-found*。否则运行*action-if-not-found*。

如果与库*library*的连接导致了未定义符号错误，而这些错误可以通过与附加的库进行连接来解决，就把这些库 用空格分隔，并作为*other-libraries*参数给出: `-lXt -lX11'。否则，本宏对*function* 是否存在的检测将总是失败，这是因为对测试程序的连接将总是因为含有未定义符号错误而失败。

宏: AC_SEARCH_LIBS (*function, search-libs[, action-if-found [, action-if-not-found]]*)
本宏等价于为每个在*search-libs*中列举的库调用一次AC_TRY_LINK_FUNC。为找到的第一个含有 *function*的库，把`-l*library*'添加到LIBS中，并且执行 *action-if-found*。否则就执行*action-if-not-found*。

库函数

以下的宏用于检测特定的C库函数。如果没有为你需要的函数定义特定的宏，而且你不需要检查它的任何特殊性质， 那么你可以使用一个通用函数检测宏。

对特定函数的检查

这些宏用于检测特定的C函数--它们是否存在，以及在某些情况下，当给出了特定的参数时，它们是如何响应的。

宏: AC_FUNC_ALLOCA

检测如何获得**alloca**。本宏试图通过检查`alloca.h'或者预定义C预处理器宏 __GNUC__和_AIX来获得**alloca**的内置（builtin）版本。 如果本宏找到了`alloca.h'，它就定义**HAVE_ALLOCA_H**。

如果上述尝试失败了，本宏就在标准C库中寻找函数。如果下列任何方法成功了，本宏就定义**HAVE_ALLOCA**。 否则，它把输出变量**ALLOCA**设置成`alloca.o'并且定义**C_ALLOCA** （这样程序就可以周期性地调用`alloca(0)'以进行垃圾的收集）。本变量是从**LIBOBJS**中 分离出来的，因此在只有一部分程序使用**LIBOBJS**中的代码时，多个程序就可以不必创建实际的库而 共享**ALLOCA**的值。

本宏并不试图从System V R3的`libPW'中，或者从System V R4的`libucb'中获取**alloca**， 这是因为这些库包含了一些造成麻烦的不兼容的函数。有些版本甚至不含有**alloca**或者含有带bug的版本。 如果你仍然需要使用它们的**alloca**，用**ar**把`alloca.o'从这些库中提取出来， 而不是编译`alloca.c'。

使用**alloca**的源文件应该以如下一段代码开头，以正确地声明它。在某些AIX版本中，对**alloca** 的声明必须在除了注释和预处理指令之前的任何东西之前出现。**#pragma**指令被缩进（indented），以便让 预标准C编译器（pre-ANSI C compiler）忽略它，而不是导致错误（choke on it）。

```
/* AIX requires this to be the first thing in the file.  */
#ifndef __GNUC__
# if HAVE_ALLOCA_H
#  include <alloca.h>
# else
#  ifdef _AIX
 #pragma alloca
#  else
#   ifndef alloca /* predefined by HP cc +Olibcalls */
char *alloca ();
#   endif
#  endif
# endif
#endif
```

宏: AC_FUNC_CLOSEDIR_VOID

如果函数**closedir**不返回有意义的值，就定义**CLOSEDIR_VOID**。否则，调用者就应该 把它的返回值作为错误指示器来进行检查。

宏: AC_FUNC_FNMATCH

如果可以使用**fnmatch**函数，并且能够工作（不象SunOS 5.4中的**fnmatch**那样）， 就定义**HAVE_FNMATCH**。

宏: AC_FUNC_GETLOADAVG

检查如何才能获得系统平均负载。如果系统含有getloadavg函数，本宏就定义HAVE_GETLOADAVG，并且把为了获得该函数而需要的库添加到LIBS中。

否则，它就把`getloadavg.o'添加到输出变量LIBOBJS之中，并且可能定义几个其他的C预处理器 宏和输出变量:

1. 如果在相应的系统中，就根据系统类型定义宏SVR4、DGUX、UMAX或者UMAX4_3。
2. 如果它找到了`nlist.h'，就定义NLIST_STRUCT。
3. 如果结构`struct nlist'含有成员`n_un'，就定义NLIST_NAME_UNION。
4. 如果在编译`getloadavg.c'时定义了LDAV_PRIVILEGED，为了使getloadavg能够 工作，程序就必须特殊地安装在系统中，并且本宏定义GETLOADAVG_PRIVILEGED。
5. 本宏设置输出变量NEED_SETGID。如果需要进行特别的安装，它的值就是`true'，否则 值就是`false'。如果NEED_SETGID为`true'，本宏把KMEM_GROUP 设置成将拥有被安装的程序的组（group）的名字。

宏: AC_FUNC_GETMNTENT

为Irix 4、PTX和Unixware在库`sun'、`seq'和`gen'中分别查找getmntent函数。 那么，如果可以使用getmntent，就定义HAVE_GETMNTENT。

宏: AC_FUNC_GETPGRP

如果getpgrp不接受参数（POSIX.1版），就定义GETPGRP_VOID。否则，它就是一个把 进程ID作为参数的BSD版本。本宏根本不检查getpgrp是否存在；如果你需要检查它的存在性，就首先为 getpgrp函数调用AC_CHECK_FUNC。

宏: AC_FUNC_MEMCMP

如果不能使用memcmp函数，或者不能处理8位数据（就像SunOS 4.1.3中的那样），就把`memcmp.o' 添加到输出变量LIBOBJS中去。

宏: AC_FUNC_MMAP

如果函数mmap存在并且能够正确地工作，就定义HAVE_MMAP。只检查已经映射（already-mapped） 的内存的私有固定映射（private fixed mapping）。

宏: AC_FUNC_SELECT_ARGTYPES

确定函数select的每个参数的正确类型，并且把这些类型分别定义成SELECT_TYPE_ARG1、SELECT_TYPE_ARG234和SELECT_TYPE_ARG5。SELECT_TYPE_ARG1的缺省值 是`int'，SELECT_TYPE_ARG234的缺省值是`int *'， SELECT_TYPE_ARG5的缺省值是`struct timeval *'。

宏: AC_FUNC_SETPGRP

如果setpgrp不接受参数（POSIX.1版），就定义SETPGRP_VOID。否则，该函数就是一个 把两个进程ID作为参数的BSD版本。本宏并不检查函数setpgrp是否存在；如果你需要检查该函数的存在 性，就首先为setpgrp调用AC_CHECK_FUNC。

宏: AC_FUNC_SETVBUF_REVERSED

如果函数setvbuf的第二个参数是缓冲区的类型并且第三个参数是缓冲区指针，而不是其他形式， 就定义SETVBUF_REVERSED。这是在System V第3版以前的情况。

宏: AC_FUNC_STRCOLL

如果函数strcoll存在并且可以正确地工作，就定义HAVE_STRCOLL。 由于有些系统包含了错误定义的strcoll，这时就不应该使用strcoll， 因此本宏要比`AC_CHECK_FUNCS(strcoll)'多作一些检查。

宏: AC_FUNC_STRFTIME

对于SCO UNIX，在库`intl'中查找strftime。而后，如果可以使用strftime， 就定义HAVE_STRFTIME。

宏: AC_FUNC_UTIME_NULL

如果`utime(*file*, NULL)'把*file*的时间标记设置成现在，就定义 HAVE_UTIME_NULL。

宏: AC_FUNC_VFORK

如果找到了`vfork.h'，就定义HAVE_VFORK_H。如果找不到可以工作的vfork， 就把vfork定义成fork。本宏检查一些已知的vfork实现中的错误 并且认为如果vfork的实现含有任何一个错误，系统就不含有可以工作的vfork。 由于子进程很少改变它们的信号句柄（signal handler），所以如果子进程的signal调用（invocation） 修改了父进程的信号句柄，将不会被当作实现的错误。

宏: AC_FUNC_VPRINTF

如果找到了vprintf，就定义HAVE_VPRINTF。否则，如果找到了_doprnt，就定义HAVE_DOPRNT。（如果可以使用vprintf，你就可以假定也可以使用vfprintf 和vsprintf。）

宏: AC_FUNC_WAIT3

如果找到了wait3并且该函数填充它的第三个参数的内容（`struct rusage *'），就定义HAVE_WAIT3。在HP-UX中，该函数并不这样做。

对普通函数的检查

这些宏被用于寻找没有包括在特定函数测试宏中的函数。如果函数可能出现在除了缺省C库以外的库中，就 要首先为这些库调用AC_CHECK_LIB。如果你除了需要检查函数是否存在之外，还要检查函数 的行为，你就不得不为此而编写你自己的测试（参见编写测试）。

宏: AC_CHECK_FUNC (*function, [action-if-found [, action-if-not-found]]*)

如果可以使用C函数*function*，就运行shell命令*action-if-found*，否则运行 *action-if-not-found*。如果你只希望在函数可用的时候定义一个符号，就考虑使用 AC_CHECK_FUNCS。由于C++比C更加标准化，即使在调用了AC_LANG_CPLUSPLUS 的时候，本宏仍然用C的连接方式对函数进行检查。（关于为测试选择语言的详情，请参见 对语言的选择）

宏: AC_CHECK_FUNCS (*function… [, action-if-found [, action-if-not-found]]*)

对于每个在以空格分隔的函数列表*function*中出现的函数，如果可用，就定义HAVE_*function* （全部大写）。如果给出了*action-if-found*，它就是在找到一个函数的时候执行的附加的shell代码。你可以给出 `break'以便在找到第一个匹配的时候跳出循环。如果给出了*action-if-not-found*，它就在找不到 某个函数的时候执行。

宏: AC_REPLACE_FUNCS (*function…*)

本宏的功能就类似于以将`*function*.o'添加到输出变量LIBOBJS的shell 代码为参数*action-if-not-found*，调用AC_CHECK_FUNCS。你可以通过用 `#ifndef HAVE_*function*'包围你为函数提供的替代版本的原型来声明函数。 如果系统含有该函数，它可能在一个你应该引入的头文件中进行声明，所以你不应该重新声明它，以避免声明冲突。

头文件

下列宏检查某些C头文件是否存在。如果没有为你需要检查的头文件定义特定的宏，而且你不需要检查它的任何特殊 属性，那么你就可以使用一个通用的头文件检查宏。

对特定头文件的检查

这些宏检查特定的系统头文件--它们是否存在，以及在某些情况下它们是否定义了特定的符号。

宏: AC_DECL_SYS_SIGLIST

如果在系统头文件，`signal.h'或者`unistd.h'，中定义了变量sys_siglist， 就定义SYS_SIGLIST_DECLARED。

宏: AC_DIR_HEADER

类似于调用AC_HEADER_DIRENT和AC_FUNC_CLOSEDIR_VOID，但为了指明找到了 哪个头文件而定义了不同的一组C预处理器宏。本宏和它定义的名字是过时的。它定义的名字是:

`dirent.h'
: DIRENT

`sys/ndir.h'
: SYSNDIR

`sys/dir.h'
: SYSDIR

`ndir.h'
: NDIR

此外，如果closedir不能返回一个有意义的值，就定义VOID_CLOSEDIR。

宏: AC_HEADER_DIRENT

对下列头文件进行检查，并且为第一个找到的头文件定义`DIR'，以及列出的C预处理器宏:

`dirent.h'
HAVE_DIRENT_H
`sys/ndir.h'
HAVE_SYS_NDIR_H
`sys/dir.h'
HAVE_SYS_DIR_H
`ndir.h'
HAVE_NDIR_H

源代码中的目录库声明应该以类似于下面的方式给出:

```
#if HAVE_DIRENT_H
# include <dirent.h>
# define NAMLEN(dirent) strlen((dirent)->d_name)
#else
# define dirent direct
# define NAMLEN(dirent) (dirent)->d_namlen
# if HAVE_SYS_NDIR_H
#  include <sys/ndir.h>
# endif
# if HAVE_SYS_DIR_H
#  include <sys/dir.h>
# endif
# if HAVE_NDIR_H
#  include <ndir.h>
# endif
#endif
```

使用上述声明，程序应该把变量定义成类型struct dirent，而不是struct direct，并且应该 通过把指向struct direct的指针传递给宏NAMLEN来获得目录项的名称的长度。

本宏还为SCO Xenix检查库`dir'和`x'。

宏: AC_HEADER_MAJOR

如果`sys/types.h'没有定义major、minor和makedev，但`sys/mkdev.h'定义了它们，就定义MAJOR_IN_MKDEV；否则，如果`sys/sysmacros.h'定义了它们，就定义MAJOR_IN_SYSMACROS。

宏: AC_HEADER_STDC

如果含有标准C（ANSI C）头文件，就定义STDC_HEADERS。 特别地，本宏检查`stdlib.h'、`stdarg.h'、`string.h'和`float.h'； 如果系统含有这些头文件，它可能也含有其他的标准C头文件。本宏还检查`string.h'是否定义了memchr （并据此对其他mem函数做出假定），`stdlib.h'是否定义了free（并据此对malloc和其他相关函数做出假定），以及`ctype.h'宏是否按照标准C的要求而可以 用于被设置了高位的字符。

因为许多含有GCC的系统并不含有标准C头文件，所以用STDC_HEADERS而不是__STDC__ 来决定系统是否含有服从标准（ANSI-compliant）的头文件（以及可能的C库函数）。

在没有标准C头文件的系统上，变种太多，以至于可能没有简单的方式对你所使用的函数进行定义以 使得它们与系统头文件声明的函数完全相同。某些系统包含了ANSI和BSD函数的混合；某些基本上是标准（ANSI） 的，但缺少`memmove'；有些系统在`string.h'或者`strings.h'中以宏的方式 定义了BSD函数；有些系统除了含有`string.h'之外，只含有BSD函数；某些系统在`memory.h' 中定义内存函数，有些在`string.h'中定义；等等。对于一个字符串函数和一个内存函数的检查可能 就够了；如果库含有这些函数的标准版，那么它就可能含有其他大部分函数。如果你在`configure.in' 中安放了如下代码：

```
AC_HEADER_STDC
AC_CHECK_FUNCS(strchr memcpy)
```

那么，在你的代码中，你就可以像下面那样放置声明：

```
#if STDC_HEADERS
# include <string.h>
#else
# ifndef HAVE_STRCHR
#  define strchr index
#  define strrchr rindex
# endif
char *strchr (), *strrchr ();
# ifndef HAVE_MEMCPY
#  define memcpy(d, s, n) bcopy ((s), (d), (n))
#  define memmove(d, s, n) bcopy ((s), (d), (n))
# endif
#endif
```

如果你使用没有等价的BSD版的函数，诸如memchr、memset、strtok 或者strspn，那么仅仅使用宏就不够了；你必须为每个函数提供一个实现。以memchr为例， 一种仅仅在需要的时候（因为系统C库中的函数可能经过了手工优化）与你的实现协作的简单方式是把实现放入 `memchr.c'并且使用`AC_REPLACE_FUNCS(memchr)'。

宏: AC_HEADER_SYS_WAIT

如果`sys/wait.h'存在并且它和POSIX.1相兼容，就定义HAVE_SYS_WAIT_H。 如果`sys/wait.h'不存在，或者如果它使用老式BSD union wait，而不是 int来储存状态值，就可能出现不兼容。如果`sys/wait.h'不与POSIX.1兼容， 那就不是引入该头文件，而是按照它们的常见解释定义POSIX.1宏。下面是一个例子：

```
#include <sys/types.h>
#if HAVE_SYS_WAIT_H
# include <sys/wait.h>
#endif
#ifndef WEXITSTATUS
# define WEXITSTATUS(stat_val) ((unsigned)(stat_val) >> 8)
#endif
#ifndef WIFEXITED
# define WIFEXITED(stat_val) (((stat_val) & 255) == 0)
#endif
```

宏: AC_MEMORY_H

在`string.h'中，如果没有定义memcpy,memcmp等函数，并且`memory.h' 存在，就定义NEED_MEMORY_H。本宏已经过时；可以用AC_CHECK_HEADERS(memory.h)来代替。 参见为AC_HEADER_STDC提供的例子。

宏: AC_UNISTD_H

如果系统含有`unistd.h'，就定义HAVE_UNISTD_H。本宏已经过时；可以用`AC_CHECK_HEADERS(unistd.h)'来代替。

检查系统是否支持POSIX.1的方式是：

```
#if HAVE_UNISTD_H
# include <sys/types.h>
# include <unistd.h>
#endif

#ifdef _POSIX_VERSION
/* Code for POSIX.1 systems.  */
#endif
```

在POSIX.1系统中包含了`unistd.h'的时候定义_POSIX_VERSION。 如果系统中没有`unistd.h'，那么该系统就一定不是POSIX.1系统。但是，有些非POSIX.1（non-POSIX.1） 系统也含有`unistd.h'。

宏: AC_USG

如果系统并不含有`strings.h'、rindex、bzero等头文件或函数，就定义USG。 定义USG就隐含地表明了系统含有`string.h'、strrchr、memset等头文件或函数。

符号USG已经过时了。作为本宏的替代，参见为AC_HEADER_STDC提供的例子。

对普通头文件的检查

这些宏被用于寻找没有包括在特定测试宏中的系统头文件。如果你除了检查头文件是否存在之外还要检查它的内容， 你就不得不为此而编写你自己的测试（参见编写测试）。

宏: AC_CHECK_HEADER (*header-file*, [*action-if-found* [, *action-if-not-found*]])

如果系统头文件*header-file*存在，就执行shell命令*action-if-found*， 否则执行*action-if-not-found*。如果你只需要在可以使用头文件的时候定义一个符号，就考虑使用 AC_CHECK_HEADERS。

宏: AC_CHECK_HEADERS (*header-file… [, action-if-found [, action-if-not-found]]*)

对于每个在以空格分隔的参数列表*header-file*出现的头文件，如果存在，就定义 HAVE_*header-file*（全部大写）。如果给出了*action-if-found*， 它就是在找到一个头文件的时候执行的附加shell代码。你可以把`break'作为它的值 以便在第一次匹配的时候跳出循环。如果给出了*action-if-not-found*，它就在找不到 某个头文件的时候被执行。

结构

以下的宏检查某些结构或者某些结构成员。为了检查没有在此给出的结构，使用AC_EGREP_CPP （参见检验声明）或者使用AC_TRY_COMPILE （参见检验语法）。

宏: AC_HEADER_STAT

如果在`sys/stat.h'中定义的S_ISDIR、S_ISREG等宏不能正确 地工作（返回错误的正数），就定义STAT_MACROS_BROKEN。这种情况出现在Tektronix UTekV、 Amdahl UTS和Motorola System V/88上。

宏: AC_HEADER_TIME

如果程序可能要同时引入`time.h'和`sys/time.h'，就定义TIME_WITH_SYS_TIME。 在一些老式系统中，`sys/time.h'引入了`time.h'，但`time.h'没有用多个包含保护 起来，所以程序不应该显式地同时包含这两个文件。例如，本宏在既使用struct timeval或 struct timezone，又使用struct tm程序中有用。它最好和 HAVE_SYS_TIME_H一起使用，该宏可以通过调用AC_CHECK_HEADERS(sys/time.h)来检查。

```
#if TIME_WITH_SYS_TIME
# include <sys/time.h>
# include <time.h>
#else
# if HAVE_SYS_TIME_H
#  include <sys/time.h>
# else
#  include <time.h>
# endif
#endif
```

宏: AC_STRUCT_ST_BLKSIZE

如果struct stat包含一个st_blksize成员，就定义HAVE_ST_BLKSIZE。

宏: AC_STRUCT_ST_BLOCKS

如果struct stat包含一个st_blocks成员，就定义HAVE_ST_BLOCKS。 否则，就把`fileblocks.o'添加到输出变量LIBOBJS中。

宏: AC_STRUCT_ST_RDEV

如果struct stat包含一个st_rdev成员，就定义HAVE_ST_RDEV。

宏: AC_STRUCT_TM

如果`time.h'没有定义struct tm，就定义TM_IN_SYS_TIME，它意味着 引入`sys/time.h'将得到一个定义得更好的struct tm。

宏: AC_STRUCT_TIMEZONE

确定如何获取当前的时区。如果struct tm有tm_zone成员，就定义HAVE_TM_ZONE。 否则，如果找到了外部数组tzname，就定义HAVE_TZNAME。

类型定义

以下的宏检查C typedefs。如果没有为你需要检查的typedef定义特定的宏，并且你不需要检查该类型 的任何特殊的特征，那么你可以使用一个普通的typedef检查宏。

对特定类型定义的检查

这些宏检查在`sys/types.h'和`stdlib.h'（如果它存在）中定义的特定的C typedef。

宏: AC_TYPE_GETGROUPS

把GETGROUPS_T定义成getgroups的数组参数的基类型gid_t或者int。

宏: AC_TYPE_MODE_T

如果没有定义mode_t，就把mode_t定义成int。

宏: AC_TYPE_OFF_T

如果没有定义off_t，就把off_t定义成long。

宏: AC_TYPE_PID_T

如果没有定义pid_t，就把pid_t定义成int。

宏: AC_TYPE_SIGNAL

如果`signal.h'把signal声明成一个指向返回值为void的函数的指针， 就把RETSIGTYPE定义成void；否则，就把它定义成int。

把信号处理器（signal handler）的返回值类型定义为RETSIGTYPE:

```
RETSIGTYPE
hup_handler ()
{
...
}
```

宏: AC_TYPE_SIZE_T

如果没有定义size_t，就把size_t定义成unsigned。

宏: AC_TYPE_UID_T

如果没有定义uid_t，就把uid_t定义成int并且把 gid_t定义成int。

对普通类型定义的检查

本宏用于检查没有包括在特定类型测试宏中的typedef。

宏: AC_CHECK_TYPE(*type, default*)

如果`sys/types.h'或者`stdlib.h'或者`stddef.h'存在，而类型 *type*没有在它们之中被定义，就把*type*定义成C（或者C++）预定义类型 *default*；例如，`short'或者`unsigned'。

C编译器的特征

下列宏检查C编译器或者机器结构的特征。为了检查没有在此列出的特征，使用AC_TRY_COMPILE （参见检验语法）或者AC_TRY_RUN （参见检查运行时的特征）

宏: AC_C_BIGENDIAN

如果字（word）按照最高位在前的方式储存（比如Motorola和SPARC，但不包括Intel和VAX，CPUS），就定义 WORDS_BIGENDIAN。

宏: AC_C_CONST

如果C编译器不能完全支持关键字const，就把const定义成空。有些编译器并不定义 __STDC__，但支持const；有些编译器定义__STDC__，但不能完全支持 const。程序可以假定所有C编译器都支持const，并直接使用它；对于那些不能完全 支持const的编译器，`Makefile'或者配置头文件将把const定义为空。

宏: AC_C_INLINE

如果C编译器支持关键字inline，就什么也不作。如果C编译器可以接受__inline__或者__inline，就把inline定义成可接受的关键字，否则就把inline定义为空。

宏: AC_C_CHAR_UNSIGNED

除非C编译器预定义了__CHAR_UNSIGNED__，如果C类型char是无符号的，就定义 __CHAR_UNSIGNED__。

宏: AC_C_LONG_DOUBLE

如果C编译器支持long double类型，就定义HAVE_LONG_DOUBLE。 有些C编译器并不定义__STDC__但支持long double类型；有些编译器定义 __STDC__但不支持long double。

宏: AC_C_STRINGIZE

如果C预处理器支持字符串化操作符（stringizing operator），就定义HAVE_STRINGIZE。字符串化操作符是`#'并且它在宏定义中以如下方式出现:

```
#define x(y) #y
```

宏: AC_CHECK_SIZEOF(*type [, cross-size]*)

把SIZEOF_*uctype*定义为C（或C++）预定义类型*type*的，以字节为单位的大小， 例如`int' or `char *'。如果编译器不能识别`type'，它就被定义为0。 *uctype*就是把*type*中所有小写字母转化为大写字母，空格转化成下划线，星号转化成`P' 而得到的名字。在交叉编译中，如果给出了*cross-size*，就使用它，否则configure就 生成一个错误并且退出。

例如，调用

```
AC_CHECK_SIZEOF(int *)
```

在DEC Alpha AXP系统中，把SIZEOF_INT_P定义为8。

宏: AC_INT_16_BITS

如果C类型int是16为宽，就定义INT_16_BITS。本宏已经过时；更常见的方式是用 `AC_CHECK_SIZEOF(int)'来代替。

宏: AC_LONG_64_BITS

如果C类型long int是64位宽，就定义LONG_64_BITS。 本宏已经过时；更常见的方式是用`AC_CHECK_SIZEOF(long)'来代替。

Fortran 77编译器的特征

下列的宏检查Fortran 77编译器的特征。为了检查没有在此列出的特征，使用AC_TRY_COMPILE （参见检验语法）或者AC_TRY_RUN （参见检验运行时的特征），但首先必须确认当前语言被设置成 Fortran 77 AC_LANG_FORTRAN77（参见对语言的选择）。

宏: AC_F77_LIBRARY_LDFLAGS

为成功地连接Fotran 77或者共享库而必须的Fortran 77内置函数（intrinsic）和运行库确定连接选项 （例如，`-L'和`-l'）。输出变量FLIBS被定义为这些选项。

本宏的目的是用于那些需要把C++和Fortran 77源代码混合到一个程序或者共享库中的情况 （参见*GNU Automake*中的`Mixing Fortran 77 With C and C++'节）。

例如，如果来自C++和Fortran 77编译器的目标文件必须被连接到一起，那么必须用C++编译器/连接器来连接 （因为有些C++特定的任务要在连接时完成，这样的任务有调用全局构造函数、模板的实例化、启动例外 （exception）支持，等等）。

然而，Fortran 77内置函数和运行库也必须被连接，但C++编译器/连接器在缺省情况下不知道如何添加这些 Fortran 77库。因此，就创建AC_F77_LIBRARY_LDFLAGS宏以确认这些Fortran 77库。

系统服务

下列宏检查操作系统服务或者操作系统能力。

宏: AC_CYGWIN

检查Cygwin环境。如果存在，就把shell变量CYGWIN设置成`yes'。 如果不存在，就把CYGWIN设置成空字符串。

宏: AC_EXEEXT

根据编译器的输出，定义替换变量EXEEXT，但不包括.c、.o和.obj文件。 对于Unix来说典型的值为空，对Win32来说典型的值为`.exe'或者`.EXE'。

宏: AC_OBJEXT

根据编译器的输出，定义替换变量OBJEXT，但不包括.c文件。 对于Unix来说典型的值为`.o'，对Win32来说典型的值为`.obj'。

宏: AC_MINGW32

检查MingW32编译环境。如果存在，就把shell变量MINGW32设置成`yes'。 如果不存在，就把MINGW32设置成空。

宏: AC_PATH_X

试图找到X Window系统的头文件和库文件。如果用户给出了命令行选项`--x-includes=*dir*'和 `--x-libraries=*dir*'，就使用这些目录。如果没有给出任一个选项，或者都没有给出，就通过 运行xmkmf以处理一个测试`Imakefile'，并且检查它所生成的`Makefile'，来得到没有 给出的目录。如果这失败了（比如说，xmkmf不存在），就在它们通常存在的几个目录中寻找。如果任何一种 方法成功了，就把shell变量x_includes和x_libraries设置成相应的位置，除非这些目录就在 编译器搜索的缺省目录中。

如果两种方法都失败了，或者用户给出命令行选项`--without-x'，就把shell变量no_x 设置成`yes'；否则就把它设置成空字符串。

宏: AC_PATH_XTRA

AC_PATH_X的增强版。它把X需要的C编译器选项添加到输出变量X_CFLAGS，并且把 X的连接选项添加到X_LIBS。如果不能使用X系统，就把`-DX_DISPLAY_MISSING' 设置成X_CFLAGS。

本宏还检查在某些系统中为了编译X程序而需要的特殊库。它把所有系统需要的库添加到输出变量X_EXTRA_LIBS。 并且它检查需要在`-lX11'之前被连接的特殊X11R6库，并且把找到的所有库添加到输出变量X_PRE_LIBS。

宏: AC_SYS_INTERPRETER

检查系统是否支持以形式为`#! /bin/csh'的行开头的脚本选择执行该脚本的解释器。 在运行本宏之后，configure.in中的shell代码就可以检查shell变量interpval； 如果系统支持`#!'，interpval将被设置成`yes'，如果不支持 就设置成`no'。

宏: AC_SYS_LONG_FILE_NAMES

如果系统支持长于14个字符的文件名，就定义HAVE_LONG_FILE_NAMES。

宏: AC_SYS_RESTARTABLE_SYSCALLS

如果系统自动地重新启动被信号所中断的系统调用，就定义HAVE_RESTARTABLE_SYSCALLS。

UNIX变种

下列宏检查对于有些程序来说需要特殊处理的一些操作系统，这是因为它们的头文件或库文件中含有特别 怪异的东西。这些宏不讨人喜欢；它们将根据它们所支持的函数或者它们提供的环境，被更加系统化的方法所代替。

宏: AC_AIX

如果在AIX系统中，就定义_ALL_SOURCE。允许使用一些BSD函数。应该在所有运行C编译器的宏之前 调用本宏。

宏: AC_DYNIX_SEQ

如果在Dynix/PTX (Sequent UNIX)系统中，就把`-lseq'添加到输出变量LIBS中。 本宏已经过时；用AC_FUNC_GETMNTENT来代替。

宏: AC_IRIX_SUN

如果在IRIX(Silicon Graphics UNIX)系统中，就把`-lsun'添加到输出变量LIBS中。 本宏已经过时。如果你用本宏来获取getmntent，就用AC_FUNC_GETMNTENT来代替。 如果你为了口令（password）和组函数的NIS版本而使用本宏，就用`AC_CHECK_LIB(sun, getpwnam)'来代替。

宏: AC_ISC_POSIX

如果在POSIX化（POSIXized） ISC UNIX系统中，就定义_POSIX_SOURCE，并且把`-posix' （对于GNU C编译器）或者`-Xp' （对于其他C编译器）添加到输出变量CC中。本宏允许使用 POSIX工具。必须在调用AC_PROG_CC之后，在调用其他任何运行C编译器的宏之前，调用本宏。

宏: AC_MINIX

如果在Minix系统中，就定义_MINIX和_POSIX_SOURCE，并且把_POSIX_1_SOURCE 定义成2。本宏允许使用POSIX工具。应该在所有运行C编译器的宏之前调用本宏。

宏: AC_SCO_INTL

如果在SCO UNIX系统中，就把`-lintl'添加到输出变量LIBS。本宏已经过时； 用AC_FUNC_STRFTIME来代替。

宏: AC_XENIX_DIR

如果在Xenix系统中，就把`-lx'添加到输出变量LIBS。还有，如果使用了`dirent.h'， 就把`-ldir'添加到LIBS。本宏已经过时；用AC_HEADER_DIRENT来代替。

编写测试

如果现有的特征测试不能完成你所需要的工作，你就必须编写一个新的。这些宏是创建模块。它们为其它宏提供了检查各种 特征是否存在并且报告结果的方式。

本章包括一些建议和一些关于现有的测试的为什么要那样编写的原因。通过阅读现有的测试，你还可以学到许多关于编写 Autoconf测试的方法。如果在一个或多个Autoconf测试中出现了错误，这些信息可以帮助你理解它们意味着什么，这有助 于你找到最佳的解决问题的办法。

这些宏检查C编译器系统的输出。它们并不为未来的使用而缓存测试的结果（参见缓存结果）， 这是因为它们没有足够的信息以生成缓存变量名。基于同样的原因，它们还不会输出任何消息。对特殊的C的特征进行的测试 调用这些宏并且缓存它们的结果、打印关于它们所进行的测试的消息。

当你编写了一个可以适用于多于一个软件包的特征测试时，最好的方式就是用一个新宏封装它。关于如何封装，参见 编写宏。

检验声明

宏AC_TRY_CPP用于检测某个特定的头文件是否存在。你可以一次检查一个头文件，或者如果你为了某些目的 而希望多个头文件都存在，也可以一次检查多个头文件。

宏: AC_TRY_CPP (*includes, [action-if-true [, action-if-false]]*)

*includes*是C或C++的**#include**语句和声明，对于它，将进行shell变量、反引用（backquote）、以及反斜线（backslash）替换。（实际上，它可以是任何C程序，但其它的语句可能没有用。）如果预处理器在处理它的时候没有报告错误，就运行shell命令*action-if-true*。否则运行shell命令*action-if-false*。

本宏使用**CPPFLAGS**，而不使用**CFLAGS**，这是因为**`-g'**、**`-O'**等选项对于 许多C预处理器来说都是不合法的选项。

下面是如何确认在某个头文件中是否包含一个特定的声明，比如说typedef、结构、结构成员或者一个函数。使用 AC_EGREP_HEADER而不是对头文件直接运行**grep**；在某些系统中，符号可能是在另一个你所检查的**`#include'**文件。

宏: AC_EGREP_HEADER (*pattern, header-file, action-if-found [, action-if-not-found]*)

如果对系统头文件*header-file*运行预处理器所产生的输出与**egrep**常规表达式*pattern*相匹配，就执行shell命令*action-if-found*，否则执行*action-if-not-found*。

为了检查由头文件或者C预处理器预定义的C预处理器符号，使用**AC_EGREP_CPP**。下面是后者的一个例子:

```
AC_EGREP_CPP(yes,
[#ifdef _AIX
  yes
#endif
], is_aix=yes, is_aix=no)
```

宏: AC_EGREP_CPP (*pattern, program, [action-if-found [, action-if-not-found]]*)

*program*是C或者C++的程序文本，对于它，将进行shell变量、反引号（backquote）以及反斜线（backslash）替换。 如果对*program*运行预处理器产生的输出与**egrep**常规表达式（regular expression）*pattern* 相匹配，就执行shell命令*action-if-found*，否则执行*action-if-not-found*。

如果宏还没有调用**AC_PROG_CPP**或者**AC_PROG_CXXCPP**（根据当前语言来确定使用那个宏，参见对语言的选择），本宏将调用它。

检验语法

为了检查C、C++或者Fortran 77编译器的语法特征，比如说它是否能够识别某个关键字，就使用**AC_TRY_COMPILE** 来尝试编译一个小的使用该特征的程序。你还可以用它检查不是所有系统都支持的结构和结构成员。

宏: AC_TRY_COMPILE (*includes, function-body, [action-if-found [, action-if-not-found]]*)

创建一个C、C++或者Fortran 77测试程序（依赖于当前语言，参见对语言的选择），来察看由*function-body*组成的函数是否可以被编译。

对于C和C++，*includes*是所有*function-body*中的代码需要的**#include**语句（如果当前选择的语言是Fortran 77，*includes*将被忽略）。如果当前选择的语言是C或者C++，本宏还将 在编译的时候使用**CFLAGS**或者**CXXFLAGS**，以及**CPPFLAGS**。如果当前选择的语言是Fortran 77，那么就在编译的时候使用**FFLAGS**。

如果文件被成功地编译了，就运行shell命令*action-if-found*，否则运行*action-if-not-found*。

本宏并不试图进行连接；如果你希望进行连接，使用**AC_TRY_LINK**（参见检验库）。

检验库

为了检查一个库、函数或者全局变量，Autoconf **configure**脚本试图编译并连接一个使用它的小程序。 不像Metaconfig，它在缺省情况下对C库使用**nm**或者**ar**以试图确认可以使用那个函数。 由于与函数相连接避免了处理**nm**和**ar**的各个变种的选项及输出格式，而且不必处理标准库的位置，所以与函数连接通常是更加可靠的办法。如果需要，它还允许进行交叉配置或者检查函数的运行是特征。另一方面，它比一次性扫描库要慢一些。

少数系统的连接器在出现找不到的函数错误（unresolved functions）时不返回失败的退出状态。这个错误使得由Autoconf 生成的配置脚本不能在这样的系统中使用。然而，有些这样的连接器允许给出选项以便正确地返回错误状态。 Autoconf目前还不能自动地处理这个问题。如果用户遇到了这样的问题，他们可能可以通过在环境中设置**LDFLAGS** 以把连接器所需要的选项（例如，**`-Wl,-dn'** on MIPS RISC/OS）传递给连接器，从而解决这个问题。

AC_TRY_LINK用于编译测试程序，以测试函数和全局变量。AC_CHECK_LIB还用本宏把被测试的库 暂时地加入LIBS并试图连接一个小程序，从而对库进行检查（参见库文件）。

宏: AC_TRY_LINK (*includes, function-body, [action-if-found [, action-if-not-found]]*)
根据当前语言（参见对语言的选择），创建一个测试程序以察看一个 函数体为*function-body*的函数是否可以被编译和连接。

对C和C++来说，*includes*给出了所有*function-body*中的代码需要的#include语句 （如果当前选定的语言是Fortran 77，*includes*将被忽略）。如果当前语言是C或者C++，本宏在编译时还将使用 CFLAGS或者CXXFLAGS，以及CPPFLAGS。如果当前选定的语言是Fortran 77，那么 在编译时将使用FFLAGS。然而，在任何情况下，连接都将使用LDFLAGS和LIBS。

如果文件被成功地编译和连接了，就运行shell命令*action-if-found*，否则就运行*action-if-not-found*。

宏: AC_TRY_LINK_FUNC (*function, [action-if-found [, action-if-not-found]]*)
根据当前语言（参见对语言的选择），创建一个测试程序以察看一个含有*function*原型和对它的调用的程序是否可以被编译和连接。

如果文件被成功地编译和连接了，就运行shell命令*action-if-found*，否则就运行*action-if-not-found*。

宏: AC_TRY_LINK_FUNC (*function, [action-if-found [, action-if-not-found]]*)
试图编译并且连接一个与*function*相连接的小程序。如果文件被成功地编译和连接了，就运行shell命令 *action-if-found*，否则就运行*action-if-not-found*。

宏: AC_COMPILE_CHECK (*echo-text, includes, function-body, action-if-found [, action-if-not-found]*)
本宏是AC_TRY_LINK的一个过时的版本。此外，如果*echo-text*不为空，它首先还要把 `checking for *echo-text*'打印到标准输出。用AC_MSG_CHECKING 和AC_MSG_RESULT来代替本宏的打印消息的功能（参见打印消息）。

检验运行时的特征

有时候，你需要知道系统在运行时作了些什么，比如说某个给定的函数是否具备某种能力或者是否含有错误。如果你能， 你可以在你的程序初始化时自行检查这类事件（比如说machine's endianness）。

如果你实在需要在配置时刻检查运行时的特征，你可以编写一个测试程序以确定结果，并且通过AC_TRY_RUN 来编译和运行它。如果可能就避免运行测试程序，这是因为使用它们使得人们不能对你的包进行交叉编译。

运行测试程序

如果你希望在配置的时候测试系统运行时的特征，就使用如下的宏。

宏: AC_TRY_RUN (*program, [action-if-true [, action-if-false [, action-if-cross-compiling]]]*)
*program*是C程序的文本，将对该文本进行shell变量和反引用（backquote）替换。如果它被成功地编译和连接了并且 在执行的时候返回的退出状态为0，就运行shell命令*action-if-true*。否则就运行shell命令*action-if-false*； 程序的退出状态可以通过shell变量`$?'得到。本宏在编译时使用CFLAGS或者CXXFLAGS以及 CPPFLAGS、LDFLAGS和LIBS。

如果使用的C编译器生成的不是在configure运行的系统上运行的可执行文件，那么测试程序就不运行。 如果给出了可选的shell命令*action-if-cross-compiling*，它们就代替生成的可执行文件执行。否则， configure打印一条错误消息并且退出。

当交叉编译使运行时测试变得不可能的时候，就尝试提供一个应急（pessimistic）的缺省值以供使用。你通过把可选的最后一个参数 传递给AC_TRY_RUN来完成这个工作。在每次生成configure的过程中，每次遇到没有提供 *action-if-cross-compiling*参数的AC_TRY_RUN调用时，autoconf都打印一条警告消息。 虽然用户将不能为交叉编译你的包而进行配置，你仍可以忽略该警告。与Autoconf一同发行的少数宏产生该警告消息。

为了为交叉编译进行配置，你还可以根据规范系统名（canonical system name）为这些参数选择值 （参见手工配置）。另一种方式是把测试缓存文件设置成目标系统的正确值 （参见缓存结果）。

为了给嵌入到其它宏（包括少数与Autoconf一同发行的宏）中的，对AC_TRY_RUN的调用提供缺省值， 你可以在它们运行之前调用AC_PROG_CC。那么，如果shell变量cross_compiling被设置成 `yes'，就使用另一种方法来获取结果，而不是调用宏。

宏: AC_C_CROSS
本宏已经过时；它不作任何事情。

测试程序指南

测试程序不应该向标准输出输出任何信息。如果测试成功，它们应该返回0，否则返回非0，以便于把成功的执行 从core dump或者其它失败中区分出来；段冲突（segmentation violations）和其它失败产生一个非0的退出状态。 测试程序应该从main中exit，而不是return，这是因为在某些系统中 （至少在老式的Sun上），main的return的参数将被忽略。

测试程序可以使用#if或者#ifdef来检查由已经执行了的测试定义的预处理器宏的值。 例如，如果你调用AC_HEADER_STDC，那么在`configure.in'的随后部分，你可以使用一个有 条件地引入标准C头文件的测试程序:

```
#if STDC_HEADERS
# include <stdlib.h>
#endif
```

如果测试程序需要使用或者创建数据文件，其文件名应该以`conftest'开头，例如`conftestdata'。 在运行测试程序之后或者脚本被中断时，configure将通过运行`rm -rf conftest*'来清除数据文件。

测试函数

在测试程序中的函数声明应该条件地含有为C++提供的原型。虽然实际上测试程序很少需要带参数的函数。

```
#ifdef __cplusplus
foo(int i)
#else
foo(i) int i;
#endif
```

测试程序声明的函数也应该有条件地含有为C++提供的，需要`extern "C"'的原型。要确保不要引入 任何包含冲突原型的头文件。

```
#ifdef __cplusplus
extern "C" void *malloc(size_t);
#else
char *malloc();
#endif
```

如果测试程序以非法的参数调用函数（仅仅看它是否存在），就组织程序以确保它从不调用这个函数。你可以 在另一个从不调用的函数中调用它。你不能把它放在对exit的调用之后，这是因为GCC第2版知道 exit永远不会返回，并且把同一块中该调用之后的所有代码都优化掉。

如果你引入了任何头文件，确保使用正确数量的参数调用与它们相关的函数，即使它们不带参数也是如此， 以避免原型造成的编译错误。GCC第2版为有些它自动嵌入（inline）的函数设置了内置原型；例如， memcpy。为了在检查它们时避免错误，既可以给它们正确数量的参数，也可以以不同的返回 类型（例如char）重新声明它们。

可移植的She11编程

在编写你自己的测试时，为了使你的代码可以移植，你应该避免使用某些she11脚本编程技术。 Bourne she11和诸如Bash和Korn she11之类的向上兼容的she11已经发展了多年，但为了避免麻烦，不要利用 在UNIX版本7，circa 1977之后添加的新特征。你不应该使用she11函数、别名、负字符集（negated character classes） 或者其它不是在所有与Bourne兼容的she11中都能找到的特征；把你自己限制到最低的风险中去。 （the lowest common denominator）。即使是unset都不能够被所有的she11所支持！ 还有，像下面那样在指定解释器的惊叹号之后给出空格:

```
#! /usr/bin/perl
```

如果你忽略了路径之前的空格，那么基于4.2BSD的系统（比如说Sequent DYNIX）将忽略这一行，这是因为它们把 `#! /'看作一个四字节的魔数（magic number）。

你在configure脚本中运行的外部程序，应该是一个相当小的集合。关于可用的外部程序列表，参见 *GNU编码标准*中的‘Makefile中的工具’一节。这个限制允许用户在只拥有相当少的程序时进行配置 和编译，这避免了软件包之间过多的依赖性。

此外，这些外部工具中的某些工具只有一部分特征是可移植的。例如，不要依赖ln支持`-f'选项， 也不要依赖cat含有任何选项。sed脚本不应该含有注释，也不应该使用长于8个字符的分支标记。 不要使用`grep -s'来禁止（suppress）输出。而要把grep的标准输出和标准错误输出 （在文件不存在的情况下会输出信息到标准错误输出）重新定向到`/dev/null'中。检查grep的退出 状态以确定它是否找到了一个匹配。

测试值和文件

configure脚本需要测试许多文件和字符串的属性。下面是在进行这些测试的时候需要提防的一些移植性问题。

程序test是进行许多文件和字符串测试的方式。人们使用替代（alternate）名`['来调用它， 但因为`['是一个m4的引用字符，在Autoconf代码中使用`['将带来麻烦。

如果你需要通过test创建多个检查，就用shell操作符`&&'和`||' 把它们组合起来，而不是使用test操作符`-a'和`-o'。在System V中， `-a'和`-o'相对于unary操作符的优先级是错误的；为此，POSIX并未给出它们，所以使用它们是 不可移植的。如果你在同一个语句中组合使用了`&&'和`||'，要记住它们的 优先级是相同的。

为了使得configure脚本可以支持交叉编译，它们不能作任何测试主系统而不是测试目标系统的事。但你偶尔 可以发现有必要检查某些特定（arbitrary）文件的存在。为此，使用`test -f'或者`test -r'。 不要使用`test -x'，因为4.3BSD不支持它。

另一个不可移植的shell编程结构是

var=${*var*:-*value*}

它的目的是仅仅在没有设定*var*的值的情况下，把*var*设置成*value*， 但如果*var*已经含有值，即使是空字符串，也不修改*var*。老式BSD shell，包括 Ultrix sh，不接受这个冒号，并且给出错误并停止。一个可以移植的等价方式是

: ${*var*=*value*}

多种情况

有些操作是以几种可能的方式完成的，它依赖于UNIX的变种。检查它们通常需要一个"case 语句"。Autoconf不能直接提供该语句； 然而，通过用一个shell变量来记录是否采用了操作的某种已知的方式，可以容易地模拟该语句。

下面是用shell变量fstype记录是否还有需要检查的情况的例子。

```
AC_MSG_CHECKING(how to get filesystem type)
fstype=no
# The order of these tests is important.
AC_TRY_CPP([#include <sys/statvfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_STATVFS) fstype=SVR4)
if test $fstype = no; then
AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/fstyp.h>], AC_DEFINE(FSTYPE_USG_STATFS) fstype=SVR3)
fi
if test $fstype = no; then
AC_TRY_CPP([#include <sys/statfs.h>
#include <sys/vmount.h>], AC_DEFINE(FSTYPE_AIX_STATFS) fstype=AIX)
fi
# (more cases omitted here)
AC_MSG_RESULT($fstype)
```

对语言的选择

既使用C又使用C++的包需要同时测试两个编译器。Autoconf生成的configure脚本 在缺省情况下检查C的特征。以下的宏决定在`configure.in'的随后部分使用那个语言的编译器。

宏: AC_LANG_C
使用CC和CPP进行编译测试并且把`.c'作为测试程序的扩展名。 如果已经运行过AC_PROG_CC，就把把shell变量cross_compiling的值设置成该宏计算的结果， 否则就设置为空。

宏: AC_LANG_CPLUSPLUS
使用CXX和CXXPP进行编译测试并且把`.C'作为测试程序的扩展名。 如果已经运行过AC_PROG_CXX，就把把shell变量cross_compiling的值设置成该宏计算的结果， 否则就设置为空。

宏: AC_LANG_FORTRAN77
使用F77进行编译测试并且把`.f'作为测试程序的扩展名。 如果已经运行过AC_PROG_F77，就把把shell变量cross_compiling的值设置成该宏计算的结果， 否则就设置为空。

宏: AC_LANG_SAVE
在堆栈中记录当前的语言（由AC_LANG_C、AC_LANG_CPLUSPLUS或者AC_LANG_FORTRAN77 所设定）。不改变当前使用的语言。在需要暂时地切换到其它特殊语言的宏之中使用本宏和AC_LANG_RESTORE。

宏: AC_LANG_RESTORE
选择储存在栈顶的，由AC_LANG_SAVE设置的语言，并且把它从栈顶删除。本宏等价于运行在最后被调用的 AC_LANG_SAVE之前最近的AC_LANG_C、AC_LANG_CPLUSPLUS或者 AC_LANG_FORTRAN77。

调用本宏的次数不要多于调用AC_LANG_SAVE的次数。

宏: AC_REQUIRE_CPP
确认已经找到了当前用于测试的预处理器。本宏根据当前选择的语言，以AC_PROG_CPP或者AC_PROG_CXXCPP 为参数调用AC_REQUIRE（参见首要的宏）。

测试的结果

一旦configure确定了某个特征是否存在，它将如何记录这一信息？这里有四种记录方式： 定义一个C预处理器符号、在输出文件中设置一个变量、为将来运行configure而把结果储存到一个缓存文件中， 以及打印一条消息以便让用户知道测试的结果。

定义C预处理器符号

对一个特征的检测的常见回应是定义一个表示测试结果的C预处理器符号。这是通过调用AC_DEFINE 或者AC_DEFINE_UNQUOTED来完成的。

在缺省状态下，AC_OUTPUT把由这些宏定义的符号放置到输出变量DEFS中，该变量为每个 定义了的符号添加一个选项`-D*symbol*=*value*'。与Autoconf第1版不同，在运行时 不定义DEFS变量。为了检查Autoconf宏是否已经定义了某个C预处理器符号，就检查适当的缓存变量的值， 例子如下：

```
AC_CHECK_FUNC(vprintf, AC_DEFINE(HAVE_VPRINTF))
if test "$ac_cv_func_vprintf" != yes; then
AC_CHECK_FUNC(_doprnt, AC_DEFINE(HAVE_DOPRNT))
fi
```

如果已经调用了AC_CONFIG_HEADER，那么就不是创建DEFS，而是由AC_OUTPUT 创建一个头文件，这是通过在一个暂时文件中把正确的值替换到#define语句中来实现的。 关于这种输出的详情，请参见配置头文件。

宏: AC_DEFINE (*variable [, value [, description]]*)
定义C预处理器变量*variable*。如果给出了*value*，就把*variable*设置成那个值（不加任何改变）， 否则的话就设置为1。*value*不应该含有新行，同时如果你没有使用AC_CONFIG_HEADER，它就不应该含有 任何`#'字符，这是因为make将删除它们。为了使用shell变量（你需要使用该变量定义一个包含了 m4引用字符`['或者`]'的值），就使用AC_DEFINE_UNQUOTED。只有在 你使用AC_CONFIG_HEADER的时候，*description*才有用。在这种情况下，*description*被 作为注释放置到生成的`config.h.in'的宏定义之前；不必在`acconfig.h'中提及该宏。下面的例子把 C预处理器变量EQUATION的值定义成常量字符串`"$a > $b"':

```
AC_DEFINE(EQUATION, "$a > $b")
```

宏: AC_DEFINE_UNQUOTED (*variable [, value [, description]]*)
类似于AC_DEFINE，但还要对*variable*和*value*进行三种shell替换（每种替换只进行一次）： 变量扩展（`$'），命令替换（``'），以及反斜线传义符（`\'）。值中的单引号和双引号 没有特殊的意义。在*variable*或者*value*是一个shell变量的时候用本宏代替AC_DEFINE。例如：

```
AC_DEFINE_UNQUOTED(config_machfile, "${machfile}")
AC_DEFINE_UNQUOTED(GETGROUPS_T, $ac_cv_type_getgroups)
AC_DEFINE_UNQUOTED(${ac_tr_hdr})
```

由于Bourne shell在语法上的特异性，不要用分号来分隔对AC_DEFINE或者AC_DEFINE_UNQUOTED的调用和 其它的宏调用或者shell代码；这将在最终的configure脚本中导致语法错误。你既可以使用空格，也可以使用 换行。就是这样：

```
AC_CHECK_HEADER(elf.h, AC_DEFINE(SVR4) LIBS="$LIBS -lelf")
```

或者：

```
AC_CHECK_HEADER(elf.h,
  AC_DEFINE(SVR4)
  LIBS="$LIBS -lelf")
```

而不是:

```
AC_CHECK_HEADER(elf.h, AC_DEFINE(SVR4); LIBS="$LIBS -lelf")
```

设置输出变量

记录测试结果的一种方式是设置*输出变量*，该变量是shell变量，它的值将被替换到configure输出的文件中。 下面的两个宏创建新的输出变量。关于总是可用的输出变量的列表，参见预定义输出变量。

宏: AC_SUBST (*variable*)

从一个shell变量创建一个输出变量。让AC_OUTPUT把变量*variable*替换到输出文件中（通常是一个或多个 `Makefile'）。这意味着AC_OUTPUT将把输入文件中的`@*variable*@'实例替换成 调用AC_OUTPUT时shell变量*variable*的值。*variable*的值不能包含新行。

宏: AC_SUBST_FILE (*variable*)

另一种从shell变量创建输出变量的方式。让AC_OUTPUT把由shell变量*variable*给出的文件名的文件的内容 （不进行替换）插入到输出文件中。这意味着AC_OUTPUT将在输出文件中（比如`Makefile.in'）把输入文件中 的`@*variable*@'实例替换为调用AC_OUTPUT时shell变量*variable*的值指明的文件 的内容。如果没有文件可以插入，就把变量设置成`/dev/null'。

本宏用于把包含特殊依赖性或者为特殊主机或目标机准备的其它make指令的`Makefile'片断插入 `Makefile'。例如，`configure.in'可以包含:

```
AC_SUBST_FILE(host_frag)dnl
host_frag=$srcdir/conf/sun4.mh
```

那么`Makefile.in'就应该包含:

```
@host_frag@
```

缓存结果

为了避免在各种configure脚本中重复地对相同的特征进行检查（或者重复地运行同一个脚本）， configure把它的检查的许多结果储存在*缓存文件*。如果在configure脚本运行时，它找到了 缓存文件，它就从中读取从前运行的结果并且不再重新运行这些检查。因此，configure将比每次都运行所有的检查 要快得多。

宏: AC_CACHE_VAL (*cache-id, commands-to-set-it*)

确认由*cache-id*指定的检查的结果是可用的。如果检查的结果在读入的缓存文件中，并且configure 没有用`--quiet'或者`--silent'调用，就打印一条消息以说明该结果已经被缓存了；否则，就运行 shell命令*commands-to-set-it*。这些命令不应具有副作用，但设置变量*cache-id*除外。它们尤其不应该调用 AC_DEFINE；紧随与对AC_CACHE_VAL的调用之后的代码应该根据缓存的值调用AC_DEFINE 作这件事。此外，它们不应该打印任何消息，比如说使用AC_MSG_CHECKING；应该在调用AC_CACHE_VAL 之前打印，以便不论测试的结果是从缓存中检索而得到的，还是通过运行shell命令而确定的，都会打印消息。如果是运行 shell命令以确定值，该值将在configure创建它的输出文件之前被储存到缓存文件中。关于如何选择 *cache-id*变量的名称，参见缓存变量名。

宏: AC_CACHE_CHECK (*message, cache-id, commands*)

这是一个更详尽地处理了打印消息的AC_CACHE_VAL版本。本宏为这些宏的最常见的应用提供了便捷的缩写。 它为*message*调用AC_MSG_CHECKING，而后以*cache-id*和*commands*为参数 调用AC_CACHE_VAL，最后以*cache-id*为参数调用AC_MSG_RESULT。

宏: AC_CACHE_LOAD

从已经存在的缓存文件中装入值，如果找不到缓存文件，就创建一个新的缓存文件。本宏由AC_INIT自动调用。

宏: AC_CACHE_SAVE

把所有缓存的值刷新到缓存文件中。本宏由AC_OUTPUT自动调用，但在configure.in的关键点调用 AC_CACHE_SAVE是十分有用的。假如配置脚本中途失败（abort）了，这些关键点仍然可以缓存一部分结果。

缓存变量名

缓存变量的名字应该符合如下格式:

*package-prefix*_cv_*value-type*_*specific-value*[_*additional-options*]

例如，`ac_cv_header_stat_broken'或者`ac_cv_prog_gcc_traditional'。 变量名的各个部分为:

package-prefix
: 你的包或者组织的缩写；除了为了方便而使用小写字母以外，与你使用的作为本地Autoconf宏的开头的前缀一样。 对于由发布的Autoconf宏使用的缓存值，它是`ac'。

_cv_
: 表明本shell变量是一个缓存值。

value-type
: 关于缓存值类别的惯例，以生成一个合理的命名系统。在Autoconf中使用的值在宏名中列出。

specific-value
: 指明本测试应用于缓存值类的那个成员。 例如，那个函数（`alloca'）、程序（`gcc'）或者输出变量（`INSTALL'）。

additional-options
: 给出应用本测试的特定成员的任何特殊行为。例如，`broken'或者`set'。 如果没有用，名字的这个部分可能被忽略掉。

赋予缓存变量的值不能含有新行。通常，它们的是将是布尔（`yes'或`no'）或者文件名或者函数名； 所以，这并不是一个重要的限制。

缓存文件

缓存文件是一个缓存了在一个系统上进行配置测试的结果，以便在配置脚本和配置的运行之间共享的shell脚本。 它对于其他系统来说是没有用的。如果它的内容因为某些原因而变得无效了，用户可以删除或者编辑它。

在缺省情况下，configure把`./config.cache'作为缓存文件，如果它还不存在，就创建它。 configure接受选项`--cache-file=*file*'以使用不同的缓存文件； 这就是configure在调用子目录中的configure脚本时所作的工作。 关于使用宏AC_CONFIG_SUBDIRS在子目录中进行配置的信息，参见 在子目录中配置其它包。

给出`--cache-file=/dev/null'会关闭缓存，这是为调试configure提供的。 只有在调用`config.status'时给出选项`--recheck'，这将导致它重新运行configure， 它才会注意到缓存文件。如果你预计需要一个长的调试时期，你还可以通过在`configure.in'的开头重新定义 缓存宏而关闭对configure脚本的装入和储存:

```
define([AC_CACHE_LOAD], )dnl
define([AC_CACHE_SAVE], )dnl
AC_INIT(whatever)
 ... rest of configure.in ...
```

试图为特定的系统类型发布缓存文件是错误的。这里存在太多的导致错误的空间，并带来太多的用于维护它们的管理开销。 对于任何不能被自动猜测出来的特征，应使用规范系统类型和连接文件的方法（参见手工配置）。

在特定系统中，每当有人运行configure脚本时，缓存文件将逐渐积累起来；缓存文件在一开始并不存在。 运行configure会把新的缓存结果与现存的缓存文件结合起来。为了让它透明地工作，只要每次都使用相同的C编译器， 站点初始化（site initialization）脚本可以指定一个站点范围（site-wide）的缓存文件以代替缺省的缓存文件。 （参见设定本地缺省值）。

如果你的配置脚本，或者configure.in中的宏调用，偶尔导致配置过程的失败，在几个关键点进行缓存可能是有用的。 在有希望修正导致上次运行的错误的时候，这样做将减少重新运行configure脚本的时间。

```
 ... AC_INIT, etc. ...
dnl checks for programs
AC_PROG_CC
AC_PROG_GCC_TRADITIONAL
 ... more program checks ...
AC_CACHE_SAVE

dnl checks for libraries
AC_CHECK_LIB(nsl, gethostbyname)
AC_CHECK_LIB(socket, connect)
 ... more lib checks ...
AC_CACHE_SAVE

dnl Might abort...
AM_PATH_GTK(1.0.2, , exit 1)
AM_PATH_GTKMM(0.9.5, , exit 1)
```

打印消息

configure脚本需要为运行它们的用户提供几种信息。下列的宏为每种信息以适当的方式打印消息。 所有宏的参数都应该由shell双引号括起来，以便shell可以对它们进行变量替换和反引号替换。你可以把消息用 m4引用字符括起来以打印包含括号的消息:

```
AC_MSG_RESULT([never mind, I found the BASIC compiler])
```

这些宏都是对shell命令echo的封装。configure应该很少需要直接运行echo来为 用户打印消息。使用这些宏使得修改每种消息如何打印及何时打印变得容易了；这些修改只需要对宏的定义进行就行了， 而所有的调用都将自动地改变。

宏: AC_MSG_CHECKING (*feature-description*)

告知用户configure正在检查特定的特征。本宏打印一条以`checking '开头，以`...'结尾，而且不带新行的消息。它必须跟随一条对AC_MSG_RESULT的调用以打印检查的结果和新行。 *feature-description*应该是类似于`whether the Fortran compiler accepts C++ comments'或者`for c89'的东西。

如果运行configure给出了选项`--quiet'或者选项`--silent'，本宏什么也不打印。

宏: AC_MSG_RESULT (*result-description*)

告知用户测试的结果。*result-description*几乎总是检查的缓存变量的值，典型的值是`yes'、`no'或者文件名。本宏应该在AC_MSG_CHECKING之后调用，并且*result-description* 应该完成由AC_MSG_CHECKING所打印的消息。

如果运行configure给出了选项`--quiet'或者选项`--silent'，本宏什么也不打印。

宏: AC_MSG_ERROR (*error-description*)

告知用户一条使configure不能完成的错误。本宏在标准错误输出中打印一条错误消息并且以非零状态退出configure。*error-description*应该是类似于`invalid value $HOME for \$HOME'的东西。

宏: AC_MSG_WARN (*problem-description*)

告知configure的使用者可能出现的问题。本宏在标准错误输出中打印消息；configure继续向后运行， 所以调用AC_MSG_WARN的宏应该为它们所警告的情况提供一个缺省的（备份）行为。 *problem-description*应该是类似于`ln -s seems to make hard links'的东西。

下列两个宏是AC_MSG_CHECKING和AC_MSG_RESULT的过时的替代版本。

宏: AC_CHECKING (*feature-description*)

除了在*feature-description*之后打印新行，本宏与AC_MSG_CHECKING相同。 它主要用于打印对一组特征测试的整体目的的描述，例如:

```
AC_CHECKING(if stack overflow is detectable)
```

宏: AC_VERBOSE(*result-description*)

除了应该在AC_CHECKING，而不是在AC_MSG_CHECKING之后调用，本宏与AC_MSG_RESULT相同； 它在打印消息前首先打印一个tab。它已经过时了。

编写宏

当你编写了一个可以用于多个软件包的特征测试时，最好用一个新宏把它封装起来。下面是一些关于编写 Autoconf宏的要求（instructions）和指导（guidelines）。

宏定义

Autoconf宏是用宏AC_DEFUN定义的，该宏与m4的内置define宏相似。 除了定义一个宏，AC_DEFUN把某些用于限制宏调用顺序的代码添加到其中。 （参见首要的宏）。

一个Autoconf宏像下面那样定义:

AC_DEFUN(*macro-name*, [*macro-body*])

这里的方括号并不表示可选的文本: 它们应当原样出现在宏定义中，以避免宏扩展问题 （参见引用）。你可以使用`$1'、`$2'等等来访问 传递给宏的任何参数。

为使用m4注释，使用m4内置的dnl； 它使m4放弃本行中其后的所有文本。因为在调用AC_INIT之前，所有的输出都被取消， 所以在`acsite.m4'和`aclocal.m4'中的宏定义之间不需要它。

关于编写m4宏的更完整的信息，参见*GNU m4*中的`如何定义新宏'。

宏名

所有Autoconf宏都以`AC_'起头以防止偶然地与其它文本发生冲突。所有它们用于内部目的的shell变量 几乎全部是由小写字母组成的，并且以`ac_'开头的名字。为了确保你的宏不会与现在的或者将来的Autoconf宏冲突， 你应该给你自己的宏名和任何它们因为某些原因而需要使用的shell变量添加前缀。它可能是你名字的开头字符，或者 你的组织或软件包名称的缩写。

大部分Autoconf宏的名字服从一个表明特征检查的种类命名惯例。宏名由几个单词组成，由下划线分隔，可以是最常见的， 也可以是最特殊的。它们的缓存变量名服从相同的惯例。（关于它们的详细信息， 参见缓存变量名）。

`AC_'之后的第一个单词通常给出被测试特征的类别。下面是Autoconf为特殊测试宏使用的类别， 它们是你很可能要编写的宏。它们的全小写形式还用于缓存变量。在可能的地方使用它们；如果不能，就发明一个你自己的类别。

C C语言内置特征。

DECL 在头文件中对C变量的声明。

FUNC 库中的函数。

GROUP 文件的UNIX组拥有者（group owner）。

HEADER
头文件。

LIB C库。

PATH 包括程序在内的，到文件的全路径名。

PROG 程序的基本名（base name）。

STRUCT
头文件中对C结构的定义。

SYS 操作系统特征。

TYPE C内置或者声明类型。

VAR 库中的C变量。

在类别之后就是特定的被测试特征的名称。宏名中所有的其它单词指明了特征的特殊方面。 例如，AC_FUNC_UTIME_NULL检查用NULL指针调用utime函数时该函数的行为。

一个作为另一个宏的内部子程序的宏的名字应该以使用它的宏的名字开头，而后是说明内部宏作了什么的一个或多个单词。 例如，AC_PATH_X有内部宏AC_PATH_X_XMKMF和AC_PATH_X_DIRECT。

引用

由其他的宏调用的宏将被m4进行几次求值；每次求值都可能需要一层引号以防止对宏或者m4 内置宏的不必要扩展，例如说`define'和`$1'。引号还需要出现在含有逗号的宏参数中， 这是因为逗号把参数与参数分隔开来。还有，把所有含有新行和调用其它宏的宏参数引起来是一个好主意。

Autoconf把m4的引用字符从缺省的``'和`''改为`['和`]'， 这是因为许多宏使用``'和`''，这不方便。然而，在少数情况下，宏需要使用方括号（通常在C程序文本 或者常规表达式中）。在这些情况下，它们使用m4内置命令changequote暂时地把引用字符改为`<<'和`>>'。 （有时，如果它们不需要引用任何东西，它们就通过把引用字符设置成空字符串以完全关闭引用。）下面是一个例子：

```
AC_TRY_LINK(
changequote(<<, >>)dnl
<<#include <time.h>
#ifndef tzname /* For SGI.  */
extern char *tzname[]; /* RS6000 and others reject char **tzname.  */
#endif>>,
changequote([, ])dnl
[atoi(*tzname);], ac_cv_var_tzname=yes, ac_cv_var_tzname=no)
```

当你用新编写的宏创建configure脚本时，仔细地验证它以检查你是否需要在你的宏之中添加更多的引号。 如果一个或多个单词在m4的输出中消失了，你就需要更多的引号。当你不能确定的时候，就使用引号。

但是，还有放置了过多层的引号的可能。如果这发生了，configure脚本的结果将包含未扩展的宏。 程序autoconf通过执行`grep AC_configure'来检查这个问题。

宏之间的依赖性

为了正确地工作，有些Autoconf宏要求在调用它们之前调用其它的宏。Autoconf提供了一种方式以确保在需要时， 某个宏已经被调用过了，以及一种在宏可能导致不正确的操作时给出警告的方式。

首要的宏

你编写的宏可能需要使用从前有其它宏计算出来的结果。例如，AC_DECL_YYTEXT要检验flex或 lex的输出，所以它要求首先调用AC_PROG_LEX以设置shell变量LEX。

比强制宏的用户跟踪宏以前的依赖性更好的是，你可以使用宏AC_REQUIRE以自动地完成这一任务。 AC_REQUIRE可以确保只在需要的时候调用宏，并且只被调用一次。

宏: AC_REQUIRE (*macro-name*)
: 如果还没有调用m4宏*macro-name*，就调用它（不带任何参数）。确保*macro-name* 用方括号引起来了。*macro-name*必须已经用AC_DEFUN定义了，或者包含一个对AC_PROVIDE 的调用以指明它已经被调用了。

一个替代AC_DEFUN的方法是使用define并且调用AC_PROVIDE。 因为这个技术并不防止出现嵌套的消息，它已经是过时的了。

宏: AC_PROVIDE (*this-macro-name*)
: 记录*this-macro-name*已经被调用了的事实。*this-macro-name*应该是调用AC_PROVIDE的宏的名字。 一个获取它的简单方式是从m4内置变量$0中获得，就像：

  ```
  AC_PROVIDE([$0])
  ```

建议的顺序

有些宏在都被调用的时候，一个宏就需要在另一个宏之前运行，但是它们并不要求调用另一个宏。例如，应该在任何运行C编译器的宏 之前调用修改了C编译器行为的宏。在文档中给出了许多这样的依赖性。

当`configure.in'文件中的宏违背了这类依赖性，Autoconf就提供宏AC_BEFORE以警告用户。 警告出现在从`configure.in'创建configure的时候，而不是在运行configure的时候。 例如，AC_PROG_CPP检查C编译器是否可以在给出`-E'的情况下运行C预处理器。因而应该在任何 改变将要使用的C编译器的宏之后调用它 。所以AC_PROG_CC包含：

```
AC_BEFORE([$0], [AC_PROG_CPP])dnl
```

如果在调用AC_PROG_CC时，已经调用了AC_PROG_CPP，它就警告用户。

宏: AC_BEFORE (*this-macro-name, called-macro-name*)

如果已经调用了*called-macro-name*，就让m4在标准错误输出上打印一条警告消息。 *this-macro-name*应该是调用AC_BEFORE的宏的名字。*macro-name*必须已经用 AC_DEFUN定义了，或者包含一个对AC_PROVIDE的调用以指明它已经被调用了。

过时的宏

配置和移植技术已经演化了好些年了。对于特定的问题，通常已经提出了更好的解决办法，或者同类的方法（ad-hoc approaches） 已经被系统化了。结果就是有些宏现在已经被认为是*过时*了；它们仍然能工作，但不再被认为是最佳选择。 Autoconf提供了宏AC_OBSOLETE，当用户使用过时的宏时，就在生成configure脚本的时候 对用户提出警告，以鼓励他们跟上潮流。一个调用实例是:

```
AC_OBSOLETE([$0], [; use AC_CHECK_HEADERS(unistd.h) instead])dnl
```

宏: AC_OBSOLETE (*this-macro-name [, suggestion]*)

让m4在标准错误输出上打印一条消息以警告*this-macro-name*是过时的，并且给出调用 过时的宏的文件名和行号。*this-macro-name*应该是调用AC_OBSOLETE的宏的名字。 如果给出了*suggestion*，就在警告消息的末尾打印它；例如，它可以建议用某个宏来代替 *this-macro-name*。

手工配置

有几种特征不能通过运行测试程序而自动猜测出来。例如，目标文件格式的细节，或者需要传递给编译器或连接器的特殊选项。 你可以使用同类手段（ad-hoc means）来检查这类特征，比如说让configure检查uname程序的 输出，或者寻找仅仅在特定系统中出现的库。然而，Autoconf为处理不可猜测的特征提供了统一的手段。

指定系统的类型

类似与其它GNU configure脚本，Autoconf生成的configure脚本可以根据系统类型的规范名(canonical name)做出决定，该规范系统名的形式为:

cpu-company-system

configure通常可以猜测出它正在运行的系统类型的规范名。为此，它运行一个称为config.guess 的脚本，该脚本使用uname或者预定义的C预处理器符号来推断系统类型的规范名。

另外，用户可以通过给configure传递命令行参数而指定系统类型。在交叉编译时必须这样作。 在大多数交叉编译的复杂情况下，要涉及到三种系统类型。用于指定它们的选项是:

`--build=`*build-type*
: 对包进行配置和编译的系统类型（很少用到）；

`--host=`*host-type*
: 包将运行的系统类型;

`--target=`*target-type*
: 包中任何编译器工具将生成的代码的系统类型。

如果用户给configure一个非选项参数，如果用户没有显式地用选项指明，它就作为缺省情况表示主机类型、 目标类型和创建系统类型。如果给出了主机类型而没有给出目标类型和创建类型，目标类型和创建类型就被设置为主机类型。 如果你正在交叉编译，你仍然必须在configure的命令行中给出你使用的交叉工具（cross-tools）的名称， 特别是C编译器。例如,

```
CC=m68k-coff-gcc configure --target=m68k-coff
```

configure能够识别许多系统类型的短别名；例如，可以在命令行中给出`decstation'而不是`mips-dec-ultrix4.2'。configure运行一个被称为config.sub的脚本以使 系统类型别名规范化。

获取规范的系统类型

下列的宏使得configure脚本可以获得系统类型。它们运行shell脚本config.guess以确定 用户在命令行中没有给出的、它们需要的关于主机、目标和创建类型的所有值。它们运行config.sub对 用户给出的任何别名进行规范化。如果你使用这些宏，你必须把这两个shell脚本与你的源代码一同发布。关于 AC_CONFIG_AUX_DIR的信息，你可以通过该宏设置configure查找这些脚本的目录，请参见 创建输出文件。如果你没有使用这些宏中的任意一个，configure 就忽略任何传递给它的`--host'、`--target'和`--build'选项。

宏: AC_CANONICAL_SYSTEM
检测系统类型并把输出变量设置成规范的系统类型。关于该宏设置变量的细节，参见系统类型变量。

宏: AC_CANONICAL_HOST
只执行AC_CANONICAL_SYSTEM中关于主机类型功能的子集。 对于不是编译工具链（compiler toolchain）一部分的程序，这就是所需要的全部功能。

宏: AC_VALIDATE_CACHED_SYSTEM_TUPLE(*cmd*)
如果缓存文件与当前主机、目标和创建系统类型不一致，就执行*cmd*或者打印一个缺省的错误消息。

系统类型变量

在调用了AC_CANONICAL_SYSTEM之后，下列输出变量包含了系统类型信息。在调用了AC_CANONICAL_HOST 之后，只设置了下列host变量。

build, host, target
规范系统名称;
build_alias, host_alias, target_alias
如果使用了config.guess，就是用户指定的名称或者规范名称;
build_cpu, build_vendor, build_os
host_cpu, host_vendor, host_os
target_cpu, target_vendor, target_os
为方便而提供的规范名称的独立部分。

使用系统类型

你将如何使用规范的系统类型? 通常，你在`configure.in'中的一个或多个case语句中使用它来 选择系统特定的C文件。而后把那些使用基于系统名的文件名的文件连接到诸如`host.h'或`target.c'的 普通的文件上。case语句模型允许使用shell通配符对多种情况进行编组，就像下面的片断:

```
case "$target" in
i386-*-mach* | i386-*-gnu*) obj_format=aout emulation=mach bfd_gas=yes ;;
i960-*-bout) obj_format=bout ;;
esac
```

宏: AC_LINK_FILES(*source…, dest…*)
使得AC_OUTPUT把每个存在文件的*source*连接到对应连接名*dest*。 如果可能，创建一个符号连接，否则就创建硬连接。*dest*和*source*应该是相对于顶层源代码目录或者 创建目录的相对路径。可以多次调用本宏。

例如，下列调用:

```
AC_LINK_FILES(config/${machine}.h config/${obj_format}.h, host.h object.h)
```

在当前目录中创建`host.h'，它是一个到`*srcdir*/config/${machine}.h'的连接， 并且创建`object.h'，它是一个到`*srcdir*/config/${obj_format}.h'的连接。

你还可以使用主机系统类型以寻找交叉编译工具。关于完成该任务的宏AC_CHECK_TOOL的信息， 参见对普通程序和文件的检查。

站点配置

configure脚本支持几种本地配置决策方式。它们是用户指明外部软件的位置，包括或除去可选的特征， 以修改过的名称安装的程序，以及为configure选项设置缺省值的手段。

与外部软件一起工作

有些软件包需要，或者可选地使用其它已经安装的软件包。用户可以把命令行选项传递给configure 以指明使用那个外部软件。选项采用下列形式之一:

--with-*package*[=*arg*]
--without-*package*

例如，`--with-gnu-ld'的意思是使用GNU连接器而不是任何其它连接器。`--with-x'的意思是 使用X Window系统。

用户可以给出包名加`='加参数的命令行参数。`no'是关于包的缺省参数；它表示*不*使用 包。既不是`yes'又不是`no'的参数将包含其它包的名字或者版本号，以便更精确地指定本程序可以 与之协同工作的包。如果没有给出参数，`--without-*package*'的缺省参数为`yes'。 `--without-*package*'等价于`--with-*package*=no'。

configure脚本并不对它们不支持的`--with-*package*'选项发出警告。 本特征允许顶层目录中的configure脚本配置一个包含多个包的源代码树。 在包支持不同的选项的时候，不会因为给出了只有一部分包支持的选项而导致不必要的错误消息。 一个不幸的副作用是选项的拼写错误就不能被检查出来了。迄今为止还没有处理该问题的更好办法。

对于每个可能使用的外部软件包，`configure.in'都应该调用AC_ARG_WITH以检测 configure的用户是否要求使用它。确定在缺省状态下，是使用还是不使用每个包，以及那个参数是合法的， 是你的任务。

宏: AC_ARG_WITH (*package, help-string [, action-if-given [, action-if-not-given]]*)

如果用户以选项`--with-*package*'或者`--without-*package*'调用 configure，就运行shell命令*action-if-given*。如果两个选项都没有给出，就运行shell命令 *action-if-not-given*。名字*package*给出了本程序应该与之协同工作的其它软件包。它应该仅仅由 字母、数字和破折号（dashes）组成。

shell命令*action-if-given*可以通过shell变量withval得到选项的参数，该变量的值实际上就是把 shell变量with_*package*的值中的所有`-'字符替换为`_'而得的。 如果你愿意，可以使用变量with_*package*。

参数*help-string*是对选项的描述，它看起来应该像:

```
  --with-readline         support fancy command line editing
```

如果需要给出更多的细节，*help-string*可能多于一行。只要确保`configure --help'中的列的排列 就可以了。不要在求助字符串中使用tab。你将需要用`['和`]'包围它以生成前导空格。

宏: AC_WITH (*package, action-if-given [, action-if-not-given]*)

这是不支持求助字符串的AC_ARG_WITH的过时版本。

选择包选项

如果软件包含有可选的编译时（compile-time）特征，用户就可以在调用configure时使用命令行选项来指明 是否编译它们。选项采用如下形式之一:

```
--enable-feature[=arg]
--disable-feature
```

这些选项允许用户选择可选的选项进行创建和安装。`--enable-*feature*'选项永远不要使特征的行为 变得不同或者导致一个特征代替另一个特征。它们只应该导致程序的一部分被创建而另一部分不创建。

用户可以通过在特征名之后添加`='和参数来给出参数。给出参数`no'表示 *不能*使用该特征。一个带有参数的特征看起来就像`--enable-debug=stabs'。如果没有给出参数， 它的缺省值就是`yes'。`--disable-*feature*'等价于`--enable-*feature*=no'。

configure脚本并不对它们所不支持的`--enable-*feature*'选项发出警告。 本特征允许顶层目录中的configure脚本配置一个包含多个包的源代码树。 在包支持不同的选项的时候，不会因为给出了只有一部分包支持的选项而导致不必要的错误消息。 一个不幸的副作用是选项的拼写错误就不能被检查出来了。迄今为止还没有处理该问题的更好办法。

对于每个可选的特征，`configure.in'都应该调用AC_ARG_ENABLE以检测configure 的用户是否要求把该特征包含进来。确定在缺省情况下，每个特征是否被包含进来，以及那些选项是合法的，是你的任务。

宏: AC_ARG_ENABLE (*feature, help-string [, action-if-given [, action-if-not-given]]*)

如果用户以选项`--enable-*feature*'或者`--disable-*feature*'调用 configure，就运行shell命令*action-if-given*。如果两个选项都没有给出，就运行shell命令 *action-if-not-given*。名称*feature*表示可选的用户级功能。它应该仅仅由字母、数字和破折号 （dashes）组成。

shell命令可以通过访问shell变量enableval来得到选项的参数，该变量的值实际上就是把shell变量 enable_*feature*的值中所有的`-'字符替换成`_'而得到的。 如果你愿意，可以使用变量enable_*feature*。*help-string*参数类似于 AC_ARG_WITH中相应的参数（参见与外部软件一起工作）。

宏: AC_ENABLE(*feature, action-if-given [, action-if-not-given]*)

这是不支持求助字符串的AC_ARG_ENABLE的过时版本。

配置站点细节

有些软件包需要复杂的与站点相关（site-specific）的信息。例如用于某种服务、公司名称和email联系地址的主名（host names）。因为有些配置脚本是通过Metaconfig方式交互地询问这些信息生成的，人们有时对于按非交互方式，由Autoconf生成配置脚本如何获取这些信息感到困惑。

这些站点配置信息应该被储存在一个*仅仅由用户*，而不是程序，编辑的文件中。文件的位置既可以基于 prefix变量，也可以是一个标准的位置，比如说用户的home目录。它甚至可能通过一个环境变量给出。程序应该在运行时，而不是在编译时，检查那个文件。运行时配置对于用户来说更为方便，并且使得配置过程比 在配置时获取这些信息要简单。关于存放数据文件的地点的详细信息，参见*GNU编码标准*中的`为安装目录而提供的变量'。

在安装的时候改变程序的名称

Autoconf支持在安装程序的时候修改程序的名称。为了使用这些变换，`configure.in'必须调用宏 AC_ARG_PROGRAM。

宏: AC_ARG_PROGRAM

把对被安装的程序的名称进行替换的sed命令序列存入输出变量program_transform_name中。

如果把下列任意选项传递给了configure，程序名就据此进行变换。否则，如果已经调用了AC_CANONICAL_SYSTEM并且`--target'的值给出了与主机类型 （用`--host'给出的，或者是在config.sub中设置的缺省值）不同的类型，就把末尾附加了 破折号的目标类型作为前缀。否则，就不进行程序名变换。

转换选项

你可以把下列命令行选项传递给configure以指定名称的转换:

`--program-prefix=`*`prefix`*

为名称添加前缀*prefix*;

`--program-suffix=`*`suffix`*

为名称添加后缀*suffix*;

`--program-transform-name=`*`expression`*

对名字作sed替换*expression*。

转换的例子

这些变换对于作为交叉编译开发环境的一部分的程序是有用的。例如，用`--target=i960-vxworks'选项配置的 运行在Sun 4上的交叉汇编器通常以`i960-vxworks-as'为名称进行安装，而不是以`as'为名安装，该名称 将于原来的Sun 4汇编器混淆。

如果你不希望安装在你的系统上的GNU程序遮蔽具有相同名称的其它程序，你可以强行要求程序名以`g'开头。例如，如果你使用`--program-prefix=g'来配置GNU diff，那么在你运行`make install' 的时候，它就安装到`/usr/local/bin/gdiff'。

作为更复杂的例子，你可以使用

```
--program-transform-name='s/^/g/; s/^gg/g/; s/^gless/less/'
```

在源代码树中的大部分程序的名字之前附加`g'，已经含有一个`g'的程序，诸如gdb，和不是GNU程序的程序，比如说less和lesskey，除外。（它假定你有一个包含了设置成使用 这些特征的程序的源代码树。）

同时安装某些程序的多个版本的一种方法是为其中一个程序的名称或为所有程序的名称附加版本号。例如，如果你还希望把 Autoconf版本1保留一段时间，你可以使用`--program-suffix=2'来配置Autoconf第2版，并且以名称`/usr/local/bin/autoconf2'、`/usr/local/bin/autoheader2'等等来安装程序。

转换的规则

下面是如何在`Makefile.in'中使用变量program_transform_name:

```
transform=@program_transform_name@
install: all
        $(INSTALL_PROGRAM) myprog $(bindir)/`echo myprog|sed '$(transform)'`

uninstall:
        rm -f $(bindir)/`echo myprog|sed '$(transform)'`
```

如果你要安装多个程序，你可以通过一个循环来完成:

```
PROGRAMS=cp ls rm
install:
        for p in $(PROGRAMS); do \
          $(INSTALL_PROGRAM) $$p $(bindir)/`echo $$p|sed '$(transform)'`; \
        done

uninstall:
        for p in $(PROGRAMS); do \
          rm -f $(bindir)/`echo $$p|sed '$(transform)'`; \
        done
```

是否在文档文件中进行变换（Texinfo或者man）是个麻烦的问题；由于名称变换的几个原因，好像不存在完美的答案。 文档对于特定的结构来说并不特殊，并且Texinfo文件与系统文档并不冲突。但它们可能与同一文件的早期版本冲突，而且 man手册有时与系统文档冲突。作为一个折衷，可能最好是对man手册进行名称替换而不对Texinfo手册 进行替换。

设定站点缺省值

Autoconf生成的configure脚本允许你的站点（site）为某些配置值提供缺省值。你可以通过创建 站点范围（site-wide）或者系统范围（system-wide）的初始化文件来达到这个目的。

如果设置了环境变量CONFIG_SITE，configure就把它的值作为读入的shell脚本的名称。 否则如果`*prefix*/share/config.site'存在，它就读入该脚本， 否则如果`*prefix*/etc/config.site'存在，它就读入该脚本。因此，如果出现冲突， 在机器特定文件中的设置将覆盖那些与机器独立的文件中的设置。

站点文件（site files）可以是任意shell脚本，但只能包含某种适于包含在其中的代码。因为configure在它读入所有 站点文件之后读取任何缓存文件，站点文件可以定义一个缺省的缓存文件以便在本系统中运行的所有Autoconf生成的 configure之间共享。如果你在站点文件中设置了缺省缓存文件，那么再在那个站点文件中设置输出变量 CC就是个好主意，这是因为缓存文件仅仅对特定的编译器来说是合法的，但许多系统还有好几个可用的编译器。

你可以在站点文件中检验或者覆盖由命令行选项设置的值；与选项对应的shell变量的名称与选项的名字的唯一区别是选项名中所有 的破折号应变换成的下划线。选项`--without-'和`--disable-'是个例外，出现它们就如同出现对应的 `--with-'或者`--enable-'并且把值设置为`no'。因此， `--cache-file=localcache'把变量cache_file的值设置为`localcache'； `--enable-warnings=no'或者`--disable-warnings'把变量enable_warnings 的值设置为`no'； `--prefix=/usr'把变量prefix设置为`/usr'；等等。

如果你需要为其它输出变量设置与缺省值不同的值（你通常不得不在命令行中重复地进行设置），比如说CFLAGS， 站点文件就是进行这种设置的好地方。如果你为*prefix*或者*exec_prefix*设置了非缺省值 （你放置站点文件的地方），如果你用环境变量CONFIG_SITE给出了站点文件，你就可以在站点文件中设置 这些非缺省值。

你可以在站点文件中设置一些缓存值。如果你正在进行交叉编译，这样做就是有用的，以避免对需要运行测试程序的特征 进行检查。你可以为`*prefix*/etc/config.site'中的系统正确地设置这些值来“预备缓存（prime cache）”。 为了找到你要设置的缓存变量名，可以在受到影响的configure脚本中寻找带有`_cv_'的shell变量， 也可以在Autoconf m4源代码中寻找这些宏。

缓存文件将十分谨慎而不至于覆盖任何在站点文件中设置的变量。类似地，你不应该在站点文件中覆盖命令行选项。 你的代码应该在修改诸如prefix和cache_file的变量之前，检查它们的缺省值（就是在 靠近configure开头的地方设置的值）。

下面是一个例子文件`/usr/share/local/gnu/share/config.site'。 （如果没有把CONFIG_SITE设置成其它文件，）命令`configure --prefix=/usr/share/local/gnu' 将读入该文件。

```
# config.site for configure
#
# Change some defaults.
test "$prefix" = NONE && prefix=/usr/share/local/gnu
test "$exec_prefix" = NONE && exec_prefix=/usr/local/gnu
test "$sharedstatedir" = '${prefix}/com' && sharedstatedir=/var
test "$localstatedir" = '${prefix}/var' && localstatedir=/var
#
# Give Autoconf 2.x generated configure scripts a shared default
# cache file for feature test results, architecture-specific.
if test "$cache_file" = ./config.cache; then
  cache_file="$prefix/var/config.cache"
  # A cache file is only valid for one C compiler.
  CC=gcc
fi
```

运行**configure**脚本

下面是关于如何配置使用configure脚本的软件包的说明，适用于包中的`INSTALL'文件。 你可能要使用的普通文本的`INSTALL'与Autoconf一同发行。

重新创建一个配置

configure脚本创建一个名为`config.status'的文件，用它描述在包最后一次进行配置时 给出的配置选项。该文件是一个shell脚本文件，如果运行它，将重新创建相同的配置。

你可以用`--recheck'选项调用`config.status'以更新它自身。如果你修改了configure， 该选项是有用的，这是因为某些测试的结果可能会与上一次运行的结果不同。选项`--recheck'以与从前使用的参数 相同的参数，再加上`--no-create'选项以防止configure运行`config.status'并创建 `Makefile'和其它文件，再加上`--no-recursion'选项以防止configure在子目录中运行 其它的configure，来重新运行configure。（这样做是让其它的`Makefile'规则可以在 `config.status'改变时运行它；关于一个例子，参见自动地重新创建）。

`config.status'还接受选项`--help'，它打印`config.status'接受的选项的概述。 还接受选项`--version'，它打印用于创建生成`config.status'的configure脚本的 Autoconf的版本号。

`config.status'检查几个能够改变它的行为的可选的环境变量：

变量： CONFIG_SHELL
用于运行带有`--recheck'选项的configure的脚本。它必须是Bourne兼容的。 缺省shell是`/bin/sh'。

变量： CONFIG_STATUS
为shell脚本提供的，用于记录配置的文件名。缺省的文件名是`./config.status'。该变量在一个包使用了另一个 包的一部分，并且由于两个包是分开维护的而不能把configure合成一个的时候有用。

以下的变量为分开发布的包提供了一种共享由configure计算的结果的方式。如果某些包需要它们中某个包， 可能是一个通用库，所需要的特征的超集那么这样做就是有用的。这些变量允许一个`config.status'文件创建 由它的`configure.in'所指明的文件之外的文件，因此它就可以被用于不同的包。

变量： CONFIG_FILES
用于执行`@*variable*@'替换的文件。缺省的文件在`configure.in'中作为参数提供给 AC_OUTPUT。

变量： CONFIG_HEADERS
用于替换C #define语句的文件。缺省的文件作为参数提供给AC_CONFIG_HEADER； 如果没有调用那个宏，`config.status'就忽略本变量。

这些变量还允许你编写只重新生成一部分文件的`Makefile'规则。例如，在上面给出的依赖性之中（参见自动地重新创建），在`configure.in'发生改变时，`config.status'将运行两次。如果你对此感到厌烦，你可以使得每次运行都仅仅重新生成关于 那条规则的文件。

```
config.h: stamp-h
stamp-h: config.h.in config.status
        CONFIG_FILES= CONFIG_HEADERS=config.h ./config.status
        echo > stamp-h

Makefile: Makefile.in config.status
        CONFIG_FILES=Makefile CONFIG_HEADERS= ./config.status
```

（如果`configure.in'并不调用AC_CONFIG_HEADER，就不必在make规则中设置 CONFIG_HEADERS。）

关于Autoconf的问题

有时我们会遇到几个关于Autoconf的问题。下面是被提及的一些问题。

发布**configure**脚本

对发行由Autoconf生成的configure有什么限制？它们是如何影响我那些使用它们的程序的？

关于由Autoconf生成的配置脚本是如何发行和如何被使用的，并没有限制。在Autoconf第1版中，它们是服从GNU通用公共许可证的。 我们仍然鼓励软件的作者按照诸如GPL的条款发行他们的作品，但Autoconf并不要求这样做。

关于可能由configure使用的其它文件，`config.h.in'服从你为你的`configure.in'而使用的 任何版权规定，这是因为它是从那个文件和公有文件`acconfig.h'中派生出来的。当`config.sub'和 `config.guess'被用于由Autoconf生成的、允许你按照与你的软件包其它部分相同的条款发布的configure 脚本中时，它们就是GPL的一个例外。`install-sh'是来自于X Consortium并且是没有版权的。

为什么需要使用GNU **m4**?

为什么Autoconf需要使用GNU m4?

许多m4的实现含有编码性（hard-coded）的，对宏的大小和数量的限制，Autoconf超过了这些限制。 它们还缺少一些内置宏，没有它们，诸如Autoconf之类的复杂应用程序将难以应付，它们包括:

```
builtin
indir
patsubst
__file__
__line__
```

因为只有软件维护者需要使用Autoconf，并且因为GNU m4易于配置和安装，需要安装GNU m4 好像是合理的。许多GNU和其它自由软件的维护者，因为他们更喜爱GNU工具，都已经安装了大部分GNU工具。

我如何解开死结?

如果Autoconf需要GNU m4并且GNU m4还有一个Autoconf configure脚本，
我如何解开这个死结？它好像是一个类似于鸡和蛋的问题！

这实际上是一种误解。虽然GNU m4带有一个由Autoconf生成的configure脚本，但在运行脚本 及安装GNU m4的时候并不需要安装Autoconf。只有在你需要修改m4的configure 脚本的时候，这只是少数几个人（主要是它的维护者）必须去作的事，才需要Autoconf。

为什么不使用Imake?

为什么不用Imake来代替configure脚本?

有些人已经提出了这个问题，所以在改编之后，我把给他们的解释写在这里。

下面是对Richard Pixley的问题的回答：

由Autoconf生成的脚本经常地在它以前从未设置过的机器上工作。这就是说，它善于推断新系统的配置。而Imake不能做到。

Imake使用含有主机特定数据的通用数据库。对X11来说，这种方法具有意义是因为发布版本是由一个控制整个 数据库的总管机关管理的一组工具组成的。

GNU工具并不按这种方式发行。每个GNU工具都有一个维护者；这些维护者散布在世界各地。使用统一的数据库将使维护变成噩梦。Autoconf可能成为这类数据库，但实际上它没有。不是列举主机的依赖性，它列举的是程序的需求。

如果你把GNU套件看作一组本地工具，那么问题就很相似了。但GNU开发工具可以作为交叉工具（cross tools）而在几乎 所有主机+目标机的组合中进行配置。所有的这些配置都可以同时（concurrency）安装。它们甚至可以被配置成可以在不同主机上共享 与主机独立的信息的形式。Imake不能处理这些问题。

Imake模板是标准的一种形式。GNU编码标准在没有强加相同的限制的情况下，解决了相同的问题。

下面是一些由Per Bothner撰写的进一步的解释：

Imake的一个长处是它易于通过使用cpp的`#include'和宏机制生成大的Makefile。 然而，cpp是不可编程的：它含有有限的条件工具，而不含有循环。而且cpp不能检查它的环境。

所有这些问题可以通过使用sh而不是cpp来解决。shell是完全可编程的、含有宏替换、可以执行 （或者编制）其它的shell脚本，并且可以检查它的环境。

Paul Eggert更详细地阐述：

使用Autoconf，安装者不必假定Imake自身已经被安装并且正常地工作了。这对于习惯使用Imake的人们来说，看起来不是 突出的长处。但在许多主机上，并没有安装Imake或者缺省的安装不能很好地工作，为此，要求安装Imake就阻碍了在这些主机 上使用由Imake配置的软件包。例如，Imake模板和配置文件可能不能适当地安装在一个主机上，或者Imake创建过程可能 会错误地假定所有的源代码文件都在一个大目录树中，或者Imake配置可能使用某个编译器而包或者安装器需要使用另一个编译器， 或者包需要的Imake的版本号与系统支持的版本号不匹配。这些问题在Autoconf中很少出现，这是因为包附带属于它自己的 独立配置处理器。

还有，Imake通常会在make和安装者的C预处理器之间遇到难以预期的影响。这里的基本问题是，C预处理器 是为处理C程序而不是`Makefile'而设计的。这对Autoconf来说问题小得多，它使用通用目的预处理器m4， 并且包的作者（而不是安装者）以标准的方式进行预处理。

最后，Mark Eichin解释道：

Imake还不是完全可扩展的。为了把新特征添加到Imake中，你需要提供你自己的项目模板，并且复制已经存在的特征的主要部分。 这意味着对于复杂的项目来说，使用由买主提供的（vendor-provided）Imake模板不能提供任何平衡作用--这是因为它们不包括 你自己的项目的任何东西（除非它是一个X11程序）。

但是，另一方面：

一个Imake胜过configure的长处是：`Imakefile'总是趋向于比`Makefile.in'简短（同样地，冗余较少）。 但是，这儿有一个修正的方法--至少对于Kerberos V5树来说，我们已经在整个树中进行了修改以调用通用的`post.in'和`pre.in' `Makefile'片断。 这意味着大部分通用的东西，即使它们通常是在configure中设置的，也不必复制。

从版本1中升级

Autoconf第2版基本上与第1版是向后兼容的。但是，它给出了作某些事的更好方法，并且不再支持版本1中一些丑陋的东西。 因此，根据你的`configure.in'文件的复杂性，你可能必须作一些手工的工作以升级到版本2。本章指出了一些在 升级的时候需要注意的问题。还有，可能你的configure脚本可以从版本2中的新特征中获得一些好处； 在Autoconf发布包中的`NEWS'文件概括了改变的部分。

首先，确认你安装了1.1版或者更高版本的GNU m4，最好是1.3版或者更高版本。在1.1版之前的版本含有bug 以至于它不能与Autoconf版本2一同工作。版本1.3及其后的版本比早期的版本更快一些，这是因为1.3版的GNU m4 对转换（diversions）进行了更有效的实现并且能够在可以快速读回的文件中冻结（freeze）它的内部状态。

改变了的文件名

如果你随Autoconf一起安装了`aclocal.m4'（相对于特定软件包的源代码目录中的`aclocal.m4'）， 你必须把它重命名为`acsite.m4'。 参见用autoconf创建configure。

如果你与你的软件包一同发布`install.sh'，就把它重命名为`install-sh'以便make的内置 规则不会无意地从该文件创建一个称为`install'的文件。AC_PROG_INSTALL将寻找这两个名字的脚本， 但最好使用新名字。

如果你使用`config.h.top'或者`config.h.bot'，你仍然可以使用它们，但如果你把它们混合到 `acconfig.h'之中，将减少你的麻烦。 参见用autoheader创建`config.h.in'。

改变了的Makefile

在你的`Makefile.in'文件中添加`@CFLAGS@'、`@CPPFLAGS@'和`@LDFLAGS@'， 以便它们可以在configure运行的时候利用环境中的这些变量的值。这样做不是必须的，但对用户来说比较方便。

对于AC_OUTPUT的每个非`Makefile'的输入文件，你还应该添加一条含有 `@configure_input@'的注释， 以便输出文件将会包含一条注释以说明它们是由configure生成的。 自动地为每种人们在AC_OUTPUT中输出的文件选择正确的注释语法需要做太多的工作。

把`config.log'和`config.cache'添加到你要在distclean目标中删除的文件的列表中。

如果你的`Makefile.in'如下:

```
prefix = /usr/local
exec_prefix = ${prefix}
```

你必须把它修改成:

```
prefix = @prefix@
exec_prefix = @exec_prefix@
```

不使用`@'字符的老式的对这些变量的替换行为已经被删除了。

改变了的宏

在Autoconf第2版中，重新命名了许多宏。你仍然可以使用旧名字，但新名字更清晰，并且易于找到相关文档。 关于为旧宏名提供新宏名的列表，参见陈旧的宏名。 用autoupdate程序转换你的`configure.in'以使用新的宏名。 参见用autoupdate更新configure。

有些宏已经被能够更好地完成工作的类似宏所代替，但在调用上并不兼容。 如果你在运行autoconf时受到了关于调用过时宏的警告，你可以安全地忽略它们，但如果你按照打印的建议 替换过时的宏，你的configure脚本通常可以工作的更好。特别地，报告测试结果的机制已经改变了。 如果你使用了echo或者AC_VERBOSE（可能是通过AC_COMPILE_CHECK）， 如果你改用AC_MSG_CHECKING和AC_MSG_RESULT，你的configure脚本的输出将 更加美观。参见打印消息。这些宏能够更好地与缓存变量协同工作。 参见缓存结果。

用autoupdate更新configure

程序autoupdate把使用Autoconf旧宏名的`configure.in'文件更新为使用当前宏名的文件。 在Autoconf第2版中，大部分宏被重命名以使用一个更统一、更具有描述性的命名机制。关于对新的命名机制的描述， 参见宏名。虽然旧宏名仍然可以工作（关于旧宏名和对应的新宏名的列表， 参见陈旧的宏名），如果你更新它们以使用新的宏名，你可以使你的 `configure.in'文件更加可读并且易于使用当前的Autoconf文档。

如果没有给出参数，autoupdate就更新`configure.in'，并且通过添加后缀`~' （或者在设置了环境变量SIMPLE_BACKUP_SUFFIX的时候，使用该环境变量的值）以备份原始版本。 如果你带参数调用autoupdate，它就读入那个文件而不是读入`configure.in'，并且把 更新的文件输出到标准输出。

autoupdate接受下列选项:

--help
-h 打印命令行选项的概述并且退出。

--macrodir=*dir*
-m *dir*
在目录*dir*中，而不是在缺省安装目录中寻找Autoconf宏文件。 你还可以把环境变量AC_MACRODIR设置成一个目录；本选项覆盖该环境变量。

--version
打印autoupdate的版本号并且退出。

改变了的结果

如果你通过检验shell变量DEFS来检验以前测试的结果，你需要把这些检验替换为对那些测试的缓存变量的检查。 在configure运行的时候，DEFS不再存在；它仅仅在生成输出文件的时候才被创建。这种与第1版 的不同是因为正确地对变量实行引用（quoting）实在太麻烦而且在每次调用AC_DEFINE都要实行引用是低效的。 参见缓存变量名。

例如，下面是为Autoconf第1版编写的`configure.in'的片断：

```
AC_HAVE_FUNCS(syslog)
case "$DEFS" in
*-DHAVE_SYSLOG*) ;;
*) # syslog is not in the default libraries.  See if it's in some other.
  saved_LIBS="$LIBS"
  for lib in bsd socket inet; do
    AC_CHECKING(for syslog in -l$lib)
    LIBS="$saved_LIBS -l$lib"
    AC_HAVE_FUNCS(syslog)
    case "$DEFS" in
    *-DHAVE_SYSLOG*) break ;;
    *) ;;
    esac
    LIBS="$saved_LIBS"
  done ;;
esac
```

这里是为版本2编写的方式：

```
AC_CHECK_FUNCS(syslog)
if test $ac_cv_func_syslog = no; then
  # syslog is not in the default libraries.  See if it's in some other.
  for lib in bsd socket inet; do
    AC_CHECK_LIB($lib, syslog, [AC_DEFINE(HAVE_SYSLOG)
      LIBS="$LIBS $lib"; break])
  done
fi
```

如果你通过在引号的后边添加反斜线以处理AC_DEFINE_UNQUOTED中的bug，你需要删除它们。 它现在以可以预期的方式工作，并且不需要特别地处理引号（处理反斜线）。 参见设定输出变量。

所有由Autoconf宏设置的布尔shell变量现在用`yes'来表示真值。虽然为了向后兼容，有些宏使用空字符串 表示假，大部分宏使用`no'来表示假。如果你依赖于shell变量用诸如1或者`t'来表示真， 你就需要改变你的测试。

改变了的宏的编写

在定义你自己的宏时，你现在应该使用AC_DEFUN而不是define。 AC_DEFUN自动调用AC_PROVIDE并且确保通过AC_REQUIRE调用该宏不会被其他宏所打断，从而防止在屏幕上出现嵌套的`checking...'消息。继续按照老办法行事没有实际上的伤害， 但它缺乏便利和吸引力。参见宏定义。

你可能把与Autoconf一同发行的宏作为如何解决问题的指南。看看它们的新版本将是一个好主意，因为风格已经有些改进并且 它们利用了一些新的特征。

如果你利用未公开的（undocumented）Autoconf内部元素（宏、变量、变换（diversions））作了微妙的工作，就要检查 你是否需要修改些什么以适应已经发生的变化。可能你甚至能够用版本2中公开（officially）支持的技术来代替你的拼装（kludging）。 但也可能不能。

为了加快你自行编写的特征测试，为它们添加缓存。看看你所有的测试是否足够一般化，从而具有足够的用途以把它们 封装到你可以共享的宏中去。

Autoconf的历史

你可能会困惑，最初为什么要编写Autoconf？它是如何演变到今天的形式的？（为什么它看起来就像大猩猩的吐沫？） 如果你不困惑，那么本章就不包含对你有用的信息，你也可能会跳过它。如果你*困惑*，那就让它明白些…

起源（Genesis）

在1991年六月，我为自由软件基金会维护了许多GNU工具。由于它们被移植到更多的平台并且增加了更多的程序， 用户必须在`Makefile'中选择的`-D'选项的数目（大约20个）变得难以承受。尤其是我-- 我不得不在许多不同的系统上对每个新的发布版本进行测试。所以我编写了一个简单的shell脚本为fileutils包猜测一些 正确的设置，并且把它作为fileutils 2.0的一部分进行发布。这个configure能够胜任工作，因此， 我在下个月中，手工对其进行了修改以用于其他几个GNU工具包，从而创建了相似的configure脚本。 Brian Berliner也修改了我的脚本以用与它的CVS修订控制系统。

那个夏天以后，我得知Richard Stallman和Richard Pixley正在开发用于GNU编译器工具的类似脚本；所以我对我的 configure进行了修改以支持它们的进化的界面：把名为`Makefile.in'的文件当作模板； 添加`+srcdir'，作为许多选项的第一个选项；并创建`config.status'文件。

出发（Exodus）

由于我从用户那里获得了反馈，我组合了许多改进，使用Emacs进行搜索和替换、剪切（cut）和粘贴（paste），在 每个脚本中进行类似的修改。随着我修改更多的GNU工具包以使用configure脚本，完全用手工更新它们 就不可能了。Rich Murphey，GNU图形工具的维护者，在给我发送的邮件中说configure脚本很好，并 问我是否有一个可以生成它们的工具可以发给他。没有，我想，但我将会有！ 所以我开始考虑如何生成它们。这样， 从手工编写configure脚本的苦力向功能强大而易于使用的Autoconf前进的旅程开始了。

Cygnus configure，它大约也在那个时候被开发，是表驱动的；这意味着用少量的大体上不可猜测 的特征来处理离散数量的系统类型（例如目标文件格式的细节）。Brian Fox为Bash开发的自动配置系统采用了类似 的方法。为了统一用法，我好像必须绝望地试图为每个操作系统的变种的特征维护一个及时更新的数据库。 更容易和更可靠的办法是不检查大多数特征--特别是在那些人们已经在本地深入地研究或者安装了买主提供的补丁 的杂合的系统。

我考虑到使用与Cygnus configure相似的结构，就是提供一个单独的configure脚本， 在运行时读入`configure.in'的片断。但是我不想让每个包都发布所有的特征测试，所以我选择了使用 预处理器从每个`configure.in'中创建不同的configure。这个方法还提供了更多的控制和便利。

我简要地察看了被Larry Wall、Harlan Stenn和Raphael Manfredi采用的Metaconfig包，但我为了几个原因而不采用它。 这种方式生成的Configure脚本是交互式的，我认为太不方便了；我不喜欢它测试某些特征的方式 （例如库函数）；我不知道它是否还有人维护，并且我所见到的Configure脚本在许多现代系统 （例如System V R4和NeXT）中都不能工作；设置在支持某个特征或者不支持该特征时所进行的动作也不是很方便； 我发现它难于学习；并且对于我的需要，它太大、太复杂了（我没有意识到Autoconf最终将变得多么大）。

我考虑过使用Perl来生成我的风格的configure脚本，但显然m4更加适合于简单的 文本替换工作：由于输出是隐含的，它的工作比较少。还有，每个人都已经拥有它了。（一开始，我并不依赖于 GNU对m4的扩展。）我在Maryland大学的一些朋友最近用一些程序，包括tvtwm， 制作了m4的前端，并且我也有兴趣试试一种新语言。

上路（Leviticus）

因为我的configure在没有与用户进行交互的条件下自动地确定了系统的能力，我决定把生成它们 的程序称作Autoconfig。但附加了版本号之后，这个名字对于老式的UNIX文件系统来说就太长了，所以我把它缩短 成Autoconf。

在1991年秋天，我召集了一群期望获得移植性的家伙（alpha测试者）以给我提供反馈从而使我可以压缩（encapsulate） 我用m4宏写的脚本并且继续添加特征、改进检查中采用的技术。测试者中的杰出人物有Pinard，他提出了 创建一个`autoconf'来运行m4并且检查找不到的宏调用的想法；还有Richard Pixley，他建议通过 运行编译器而不是在文件系统中寻找引入文件和符号，以获得更精确的结果；还有Karl Berry，他使得Autoconf可以配置 Tex并且把宏索引添加到文档中；还有Ian Taylor，他增加了对创建C头文件的支持以代替在`Makefile'中添加 `-D'选项的方法，以便他可以把Autoconf用于他的UUCP包。alpha测试者愉快地、一次又一次地随着 Autoconf不同发布版本中的Autoconf名称和宏调用惯例的改变而调整他们的文件。他们都贡献了许多特定的检查、绝妙的 想法，以及对bug的修正。

发展（Numbers）

在1992年七月，在alpha测试之后一个月，我发布了Autoconf 1.0，并且修改了许多GNU包以使用它。我对它带来的正面 作用感到很吃惊。很多人，包括那些编写并不属于GNU工程的软件（例如TCL、FSP和Kerberos V5）的人们，开始使用它， 以至于我无法跟踪他们了。随着很多使用configure脚本的人报告他们所遇到的问题，Autoconf继续快速地得到改进，

Autoconf成为考验m4实现的酷刑般的测试。由于Autoconf定义的宏的长度，UNIX m4开始 失败（dump core），同时也发现了GNU m4中的一些bug。最终，我们意识到我们需要使用一些只有 GNU m4才提供的特征。特别的，4.3BSD m4含有一组增强了的内置宏；System V版本更好一些，但仍然不能提供我们所需要的所有东西。

随着Autoconf得到人们越来越多的重视，对Autoconf进行了更多的开发（并且有了我不能预见的用途）。Karl Berry添加了 对X11的检查。david zuhn贡献了对C++的支持。Pinard使Autoconf能够诊断非法的参数。Jim Blandy勇敢地用它配置 了GNU Emacs，并且为某些未来的改进打下了基础。Roland McGrath用它配置了GNU C库，编写了autoheader 脚本以自动创建C头文件模板，并且为configure添加了一个`--verbose'选项。 Noah Friedman添加了`--macrodir'选项和环境变量AC_MACRODIR。（他还提出了术语 *autoconfiscate*，用来表示“调整软件包以使用Autoconf”。）Roland和Noah改进了AC_DEFINE 中的引用保护并且修正了许多bug，特别是在1993年二月到六月间我们对处理移植性问题感到厌倦的时候。

现状（Deuteronomy）

在积累了一个关于希望添加的主要特征的长长的列表，并且在几年之中各式各样的人们提供的补丁残留了古怪的效果之后。 在1994年四月，处理对Cygnus的支持时，我开始对Autoconf进行一次主要的修订。我添加了大部分Cygnus configure 有，而Autoconf缺少的特征，主要是在david zuhn和Ken Raeburn的帮助下改编Cygnus configure的 相关部分。这些特征包括对使用`config.sub'、`config.guess'、`--host'和`--target'的支持；创建对文件的连接；以及在子目录中运行configure脚本。添加这些特征使得Ken可以放弃GNU as，Rob Savoye可以放弃DejaGNU，而改用Autoconf。

作为对其他人的要求的回应，我添加了更多的特征。许多人要求configure脚本能够 在不同的运行中共享检查的结果，这是因为它们实在太慢了（尤其是像Cygnus那样在配置一个大的源代码树的时候）。 Mike Haertel建议增加与位置有关的初始化脚本。发布必须在MS-DOS中解包（unpack）的软件的人们要求 提供一种覆盖那些诸如`config.h.in'那样的、含有两个点的文件名中的`.in'扩展名的方法。 Jim Avera通过AC_DEFINE和AC_SUBST中的引用扩展了对程序的检测；他的洞察力带来 了重要的改进。Richard Stallman要求把编译器的输出送到`config.log'中，而不是送到`/dev/null'中， 以帮助人们调试Emacs configure脚本。

由于我对程序质量的不满，我进行了一些其他的修改。我减少了用于显示检查结果的消息的二义性，总是打印结果。 我识别宏的名字并且消除编码风格的不一致性。我添加了一些我所开发的附加工具以助于修改源代码包以使用Autoconf。 在Pinard的帮助下，我创建了不会在彼此的消息中导致冲突的宏。（这个特征暴露了他草率地修正的、GNU m4 中的一些性能瓶颈！）我重新组织了人们需要解决的问题的文档。并且我开始了一组测试（testsuite），这是因为 经验已经表明：在我们修改Autoconf的时候，它有明显的回归倾向。

一些alpha测试者再次给出了难以估量的反馈，特别是 Pinard、Jim Meyering、Karl Berry、Rob Savoye、Ken Raeburn和Mark Eichin。

最后，2.0版本准备好了。而且我们也很高兴。（我们又有闲暇时间了。我想。哇，很好。）

陈旧的宏名

在Autoconf的第2版，大部分宏被重新命名以使用更加统一和具有描述性的命名方案。下面是被重新命名了的宏的原来名字， 随后给出了这些宏现在的名字。虽然为了保持向后兼容，旧名字仍然能够被autoconf程序所接受，旧名字都 被看作过时的。关于新的命名方案，参见宏名。

AC_ALLOCA
: AC_FUNC_ALLOCA

AC_ARG_ARRAY
: 因为用途有限而被删除了。

AC_CHAR_UNSIGNED
: AC_C_CHAR_UNSIGNED

AC_CONST
: AC_C_CONST

AC_CROSS_CHECK
: AC_C_CROSS

AC_ERROR
: AC_MSG_ERROR

AC_FIND_X
: AC_PATH_X

AC_FIND_XTRA
: AC_PATH_XTRA

AC_FUNC_CHECK
: AC_CHECK_FUNC

AC_GCC_TRADITIONAL
AC_PROG_GCC_TRADITIONAL
AC_GETGROUPS_T
AC_TYPE_GETGROUPS
AC_GETLOADAVG
AC_FUNC_GETLOADAVG
AC_HAVE_FUNCS
AC_CHECK_FUNCS
AC_HAVE_HEADERS
AC_CHECK_HEADERS
AC_HAVE_POUNDBANG
AC_SYS_INTERPRETER(不同的调用惯例)
AC_HEADER_CHECK
AC_CHECK_HEADER
AC_HEADER_EGREP
AC_EGREP_HEADER
AC_INLINE
AC_C_INLINE
AC_LN_S
AC_PROG_LN_S
AC_LONG_DOUBLE
AC_C_LONG_DOUBLE
AC_LONG_FILE_NAMES
AC_SYS_LONG_FILE_NAMES
AC_MAJOR_HEADER
AC_HEADER_MAJOR
AC_MINUS_C_MINUS_O
AC_PROG_CC_C_O
AC_MMAP
AC_FUNC_MMAP
AC_MODE_T
AC_TYPE_MODE_T
AC_OFF_T
AC_TYPE_OFF_T
AC_PID_T
AC_TYPE_PID_T
AC_PREFIX
AC_PREFIX_PROGRAM
AC_PROGRAMS_CHECK
AC_CHECK_PROGS
AC_PROGRAMS_PATH
AC_PATH_PROGS
AC_PROGRAM_CHECK
AC_CHECK_PROG
AC_PROGRAM_EGREP
AC_EGREP_CPP
AC_PROGRAM_PATH
AC_PATH_PROG
AC_REMOTE_TAPE
因为用途有限而被删除了。
AC_RESTARTABLE_SYSCALLS
AC_SYS_RESTARTABLE_SYSCALLS

AC_RETSIGTYPE
AC_TYPE_SIGNAL
AC_RSH
因为用途有限而被删除了。
AC_SETVBUF_REVERSED
AC_FUNC_SETVBUF_REVERSED
AC_SET_MAKE
AC_PROG_MAKE_SET
AC_SIZEOF_TYPE
AC_CHECK_SIZEOF
AC_SIZE_T
AC_TYPE_SIZE_T
AC_STAT_MACROS_BROKEN
AC_HEADER_STAT
AC_STDC_HEADERS
AC_HEADER_STDC
AC_STRCOLL
AC_FUNC_STRCOLL
AC_ST_BLKSIZE
AC_STRUCT_ST_BLKSIZE
AC_ST_BLOCKS
AC_STRUCT_ST_BLOCKS
AC_ST_RDEV
AC_STRUCT_ST_RDEV
AC_SYS_SIGLIST_DECLARED
AC_DECL_SYS_SIGLIST
AC_TEST_CPP
AC_TRY_CPP
AC_TEST_PROGRAM
AC_TRY_RUN
AC_TIMEZONE
AC_STRUCT_TIMEZONE
AC_TIME_WITH_SYS_TIME
AC_HEADER_TIME
AC_UID_T
AC_TYPE_UID_T
AC_UTIME_NULL
AC_FUNC_UTIME_NULL
AC_VFORK
AC_FUNC_VFORK
AC_VPRINTF
AC_FUNC_VPRINTF
AC_WAIT3
AC_FUNC_WAIT3
AC_WARN
AC_MSG_WARN
AC_WORDS_BIGENDIAN
AC_C_BIGENDIAN
AC_YYTEXT_POINTER
AC_DECL_YYTEXT

环境变量索引

这是一个按照字母顺序排序的，由Autoconf检查的环境变量的列表。

Jump to: a - c - s

a

- AC_MACRODIR, AC_MACRODIR, AC_MACRODIR, AC_MACRODIR, AC_MACRODIR, AC_MACRODIR

c

- CONFIG_FILES
- CONFIG_HEADERS
- CONFIG_SHELL
- CONFIG_SITE
- CONFIG_STATUS

s

- SIMPLE_BACKUP_SUFFIX

输出变量索引

这是一个按照字母顺序排序的，Autoconf将在它所创建的文件（通常是一个或更多`Makefile'） 中进行替换的变量的列表。关于这些是如何实现的，请参见设定输出变量。

Jump to: a - b - c - d - e - f - h - i - k - l - m - n - o - p - r - s - t - x - y

a

- ALLOCA
- AWK

b

- bindir
- build
- build_alias
- build_cpu
- build_os
- build_vendor

c

- CC, CC, CC
- CFLAGS, CFLAGS
- configure_input
- CPP
- CPPFLAGS
- CXX
- CXXCPP
- CXXFLAGS, CXXFLAGS

d

- datadir
- DEFS

e

- exec_prefix
- EXEEXT

f

- F77
- FFLAGS, FFLAGS

- FLIBS

h

- host
- host_alias
- host_cpu
- host_os
- host_vendor

i

- includedir
- infodir
- INSTALL
- INSTALL_DATA
- INSTALL_PROGRAM
- INSTALL_SCRIPT

k

- KMEM_GROUP

l

- LDFLAGS
- LEX
- LEX_OUTPUT_ROOT
- LEXLIB
- libdir
- libexecdir
- LIBOBJS, LIBOBJS, LIBOBJS, LIBOBJS, LIBOBJS
- LIBS, LIBS, LIBS
- LN_S
- localstatedir

m

- mandir

n

- NEED_SETGID

o

- OBJEXT
- oldincludedir

p

- prefix
- program_transform_name

r

- RANLIB

s

- sbindir
- SET_MAKE
- sharedstatedir
- srcdir
- subdirs
- sysconfdir

t

- target
- target_alias
- target_cpu
- target_os
- target_vendor
- top_srcdir

X

- X_CFLAGS
- X_EXTRA_LIBS
- X_LIBS
- X_PRE_LIBS

y

- YACC

预处理器符号索引

这是一个按照字母顺序排序的，由Autoconf宏定义的C预处理符号的列表。为了与Autoconf协同工作，C源代码应该 在**#if**指令中使用这些名字。

Jump to: _ - c - d - f - g - h - i - l - m - n - o - p - r - s - t - u - v - w - y

_

- __CHAR_UNSIGNED__
- _ALL_SOURCE
- _MINIX
- _POSIX_1_SOURCE
- _POSIX_SOURCE, _POSIX_SOURCE
- _POSIX_VERSION

c

- C_ALLOCA
- CLOSEDIR_VOID
- const

d

- DGUX
- DIRENT

f

- F77_NO_MINUS_C_MINUS_O

g

- GETGROUPS_T
- GETLODAVG_PRIVILEGED
- GETPGRP_VOID
- gid_t

h

- HAVE_ALLOCA_H
- HAVE_CONFIG_H
- HAVE_DIRENT_H
- HAVE_DOPRNT
- HAVE_*function*
- HAVE_GETMNTENT

- HAVE_*header*
- HAVE_LONG_DOUBLE
- HAVE_LONG_FILE_NAMES
- HAVE_MMAP
- HAVE_NDIR_H
- HAVE_RESTARTABLE_SYSCALLS
- HAVE_ST_BLKSIZE
- HAVE_ST_BLOCKS
- HAVE_ST_RDEV
- HAVE_STRCOLL
- HAVE_STRFTIME
- HAVE_STRINGIZE
- HAVE_SYS_DIR_H
- HAVE_SYS_NDIR_H
- HAVE_SYS_WAIT_H
- HAVE_TM_ZONE
- HAVE_TZNAME
- HAVE_UNISTD_H
- HAVE_UTIME_NULL
- HAVE_VFORK_H
- HAVE_VPRINTF
- HAVE_WAIT3

i

- inline
- INT_16_BITS

l

- LONG_64_BITS

m

- MAJOR_IN_MKDEV
- MAJOR_IN_SYSMACROS
- mode_t

n

- NDIR
- NEED_MEMORY_H
- NEED_SETGID
- NLIST_NAME_UNION
- NLIST_STRUCT
- NO_MINUS_C_MINUS_O

o

- off_t

p

- pid_t

r

- RETSIGTYPE

s

- SELECT_TYPE_ARG1
- SELECT_TYPE_ARG234
- SELECT_TYPE_ARG5
- SETPGRP_VOID
- SETVBUF_REVERSED
- size_t

宏索引

这是按字母排序的Autoconf宏列表。为了使列表易于使用，宏以没有前缀`AC_'的形式列出。

- CHECK_FILE
- CHECK_FILES
- CHECK_FUNC
- CHECK_FUNCS
- CHECK_HEADER
- CHECK_HEADERS
- CHECK_LIB
- CHECK_PROG
- CHECK_PROGS
- CHECK_SIZEOF
- CHECK_TOOL
- CHECK_TYPE
- CHECKING
- COMPILE_CHECK
- CONFIG_AUX_DIR
- CONFIG_HEADER
- CONFIG_SUBDIRS
- CONST
- CROSS_CHECK
- CYGWIN

d

- DECL_SYS_SIGLIST
- DECL_YYTEXT
- DEFINE
- DEFINE_UNQUOTED
- DEFUN
- DIR_HEADER
- DYNIX_SEQ

e

- EGREP_CPP
- EGREP_HEADER
- ENABLE
- ERROR
- EXEEXT

f

- F77_LIBRARY_LDFLAGS
- FIND_X
- FIND_XTRA
- FUNC_ALLOCA
- FUNC_CHECK
- FUNC_CLOSEDIR_VOID
- FUNC_FNMATCH
- FUNC_GETLOADAVG
- FUNC_GETMNTENT
- FUNC_GETPGRP
- FUNC_MEMCMP
- FUNC_MMAP
- FUNC_SELECT_ARGTYPES
- FUNC_SETPGRP
- FUNC_SETVBUF_REVERSED
- FUNC_STRCOLL
- FUNC_STRFTIME
- FUNC_UTIME_NULL
- FUNC_VFORK
- FUNC_VPRINTF
- FUNC_WAIT3

g

- GCC_TRADITIONAL
- GETGROUPS_T
- GETLOADAVG

h

- HAVE_FUNCS
- HAVE_HEADERS
- HAVE_LIBRARY
- HAVE_POUNDBANG
- HEADER_CHECK
- HEADER_DIRENT
- HEADER_EGREP
- HEADER_MAJOR
- HEADER_STAT
- HEADER_STDC
- HEADER_SYS_WAIT
- HEADER_TIME

i

- INIT
- INLINE
- INT_16_BITS
- IRIX_SUN
- ISC_POSIX

l

- LANG_C
- LANG_CPLUSPLUS
- LANG_FORTRAN77
- LANG_RESTORE
- LANG_SAVE
- LINK_FILES
- LN_S
- LONG_64_BITS
- LONG_DOUBLE
- LONG_FILE_NAMES

m

- MAJOR_HEADER
- MEMORY_H
- MINGW32
- MINIX
- MINUS_C_MINUS_O
- MMAP
- MODE_T
- MSG_CHECKING
- MSG_ERROR
- MSG_RESULT
- MSG_WARN

o

- OBJEXT
- OBSOLETE
- OFF_T
- OUTPUT

p

- PATH_PROG
- PATH_PROGS
- PATH_X
- PATH_XTRA
- PID_T
- PREFIX
- PREFIX_PROGRAM
- PREREQ

- PROG_AWK
- PROG_CC
- PROG_CC_C_O
- PROG_CPP
- PROG_CXX
- PROG_CXXCPP
- PROG_F77_C_O
- PROG_FORTRAN
- PROG_GCC_TRADITIONAL
- PROG_INSTALL
- PROG_LEX
- PROG_LN_S
- PROG_MAKE_SET
- PROG_RANLIB
- PROG_YACC
- PROGRAM_CHECK
- PROGRAM_EGREP
- PROGRAM_PATH
- PROGRAMS_CHECK
- PROGRAMS_PATH
- PROVIDE

r

- REMOTE_TAPE
- REPLACE_FUNCS
- REQUIRE
- REQUIRE_CPP
- RESTARTABLE_SYSCALLS
- RETSIGTYPE
- REVISION
- RSH

s

- SCO_INTL
- SEARCH_LIBS, SEARCH_LIBS
- SET_MAKE
- SETVBUF_REVERSED
- SIZE_T
- SIZEOF_TYPE
- ST_BLKSIZE
- ST_BLOCKS
- ST_RDEV
- STAT_MACROS_BROKEN, STAT_MACROS_BROKEN
- STDC_HEADERS
- STRCOLL
- STRUCT_ST_BLKSIZE
- STRUCT_ST_BLOCKS
- STRUCT_ST_RDEV
- STRUCT_TIMEZONE
- STRUCT_TM
- SUBST
- SUBST_FILE
- SYS_INTERPRETER
- SYS_LONG_FILE_NAMES
- SYS_RESTARTABLE_SYSCALLS
- SYS_SIGLIST_DECLARED

t

- TEST_CPP
- TEST_PROGRAM
- TIME_WITH_SYS_TIME
- TIMEZONE
- TRY_COMPILE
- TRY_CPP
- TRY_LINK

This document was generated on 20 June 1999 using the texi2html translator version 1.52.