FreeBSD Porter 手册

目录

1. 介绍 . . . 5
2. 自行制作新 port . . . 6
3. 简单的 port . . . 7
   3.1. 编写 Makefile . . . 7
   3.2. 创建描述文件 . . . 7
   3.3. 创建校验和文件 . . . 9
   3.4. 测试 port . . . 9
   3.5. 用 portlint 来检查 port . . . 9
   3.6. 提交新 port . . . 10
4. 复杂的 Porting . . . 11
   4.1. 整个系统是如何运转的? . . . 11
   4.2. 获取源代码 . . . 11
   4.3. 修改 port . . . 12
   4.4. 打补丁 . . . 12
   4.5. 配置 . . . 13
   4.6. 处理用户输入 . . . 13
5. 配置 Makefile . . . 14
   5.1. 作者发布的代码 . . . 14
   5.2. 命名 . . . 14
   5.3. 分类 . . . 18
   5.4. 源码包文件 . . . 22
   5.5. MAINTAINER (监护人) . . . 30
   5.6. COMMENT (一句话说明) . . . 31
   5.7. 依赖关系 . . . 31
   5.8. MASTERDIR (主 port 所在的目录) . . . 35
   5.9. 联机手册 . . . 36
   5.10. Info 文件 . . . 37
   5.11. Makefile 选项 . . . 37
   5.12. 指定工作临时目录 . . . 40
   5.13. 处理冲突 . . . 41
   5.14. 安装文件 . . . 41
6. 特殊情况 . . . 45
   6.1. 共享库 . . . 45
   6.2. Ports 的发行限制 . . . 45
   6.3. 联编机制 . . . 46
   6.4. 利用 GNU autotools . . . 48
   6.5. 使用 GNU gettext . . . 49
   6.6. 使用 perl . . . 51
   6.7. 使用 X11 . . . 51
   6.8. 使用 GNOME . . . 53
   6.9. 使用 Qt . . . 54
   6.10. 使用 KDE . . . 56
   6.11. 使用 Java . . . 57
   6.12. Web 应用， Apache 和 PHP . . . 60
   6.13. 使用 Python . . . 62
   6.14. 使用 Tcl/Tk . . . 63
   6.15. 使用 Emacs . . . 63
   6.16. 使用 Ruby . . . 63
   6.17. 使用 SDL . . . 64

6.18. 使用 wxWidgets . . . 65
6.19. 使用 Lua . . . 69
6.20. 使用 Xfce . . . 73
6.21. 使用 Mozilla . . . 73
6.22. 使用数据库 . . . 74
6.23. 启动和停止服务 (rc 脚本) . . . 74
6.24. 添加用户和用户组 . . . 76
6.25. 依赖内核源代码的 Ports . . . 76
7. 高级 pkg-plist 用法 . . . 78
7.1. 根据 make 变量对 pkg-plist 进行修改 . . . 78
7.2. 空目录 . . . 78
7.3. 配置文件 . . . 79
7.4. 动态装箱单与静态装箱单的对比 . . . 80
7.5. 装箱单 (package list) 的自动化制作 . . . 80
8. pkg-* 文件 . . . 82
8.1. pkg-message (安装预编译包时显示的消息文件) . . . 82
8.2. pkg-install (安装预编译包时执行的脚本文件) . . . 82
8.3. pkg-deinstall (卸载时执行的脚本文件) . . . 82
8.4. pkg-req (安装预编译包时检测是否应执行操作的脚本文件) . . . 82
8.5. 改变 pkg-* 文件的名字 . . . 83
8.6. 使用 SUB_FILES 和 SUB_LIST . . . 83
9. 测试您的 port . . . 84
9.1. 运行 make describe . . . 84
10. 升级一个 port . . . 86
10.1. 使用 CVS 制作补丁 . . . 87
10.2. UPDATING 和 MOVED 文件 . . . 88
11. Ports 的安全 . . . 89
11.1. 安全为何如此重要 . . . 89
11.2. 修复安全漏洞 . . . 89
11.3. 通知整个用户群体 . . . 89
12. 该做什么和不该做什么 . . . 94
12.1. 介绍 . . . 94
12.2. WRKDIR (联编时使用的临时目录) . . . 94
12.3. WRKDIRPREFIX (用于联编的临时目录的父目录名) . . . 94
12.4. 区分不同的操作系统， 以及 OS 的版本 . . . 94
12.5. __FreeBSD_version 值 . . . 95
12.6. 在 bsd.port.mk 之后写一些内容 . . . 124
12.7. 在 wrapper 脚本中使用 exec 语句 . . . 125
12.8. 理性行事 . . . 125
12.9. 遵循 CC 和 CXX 设置 . . . 126
12.10. 遵循 CFLAGS . . . 126
12.11. 线程库 . . . 127
12.12. 反馈 . . . 127
12.13. README.html . . . 127
12.14. 使用 BROKEN、 FORBIDDEN 或 IGNORE 阻止用户安装 port . . . 127
12.15. 使用 DEPRECATED 或 EXPIRATION_DATE 表示某个 port 将被删除 . . . 129
12.16. 避免使用 .error 结构 . . . 129
12.17. 对于 sysctl 的使用 . . . 129
12.18. 重新发布的 distfiles . . . 130
12.19. 杂记 . . . 130
13. 示范的 Makefile . . . 131

14. 保持同步 133
14.1. FreshPorts 133
14.2. 代码库的 Web 访问界面 133
14.3. FreeBSD Ports 邮件列表 133
14.4. 位于 pointyhat.FreeBSD.org 的 FreeBSD Port 联编集群 133
14.5. FreeBSD 的 Ports Distfile 扫描器 133
14.6. FreeBSD 的 Ports 追踪系统 133

Chapter 1. 介绍

几乎每个人都是通过 FreeBSD Ports Collection 在 FreeBSD 上面装应用程序 (“ports”)的。
就像FreeBSD的其它部分一样， 它主要来自于志愿者的努力。
所以在阅读这份文档的时候请务必记住这些。

在 FreeBSD 的世界里， 任何人都能提交新的 port， 或志愿地维护一个已有的 port， 如果那个 port 没人维护的话 - 不需要任何特殊的权限来做这件事情。

Chapter 2. 自行制作新 port

那么， 您有兴趣创建自己的 port 或升级现有的 port？ 太好了。

下面的内容将会提供一些创建FreeBSD port的指导。 如果想升级一个现有的 port，
那么您应该在看完这些内容并阅读 升级一个 port。

因为这份文档不是十分详细， 您还应该再参考一下 /usr/ports/Mk/bsd.port.mk， 所有 port 的 Makefile 文件都会包含它。 即使不是每天都去摆弄 Makefile， 您也会从那个文件里面获得很多知识，
里面的注释非常详细。 还有要补充一下，如果您有其它的问题， 可以给FreeBSD ports 邮件列表 这个 mailing list 发信。



在这份文档里提到的大部分的变量 (VAR) 是不能修改的。 大多 (但不是全部) 都在 /usr/ports/Mk/bsd.port.mk 的开始部分进行了介绍； 其它一些也应该可以在那里找到。
注意这些文件使用了非标准的制表符： Emacs 和 Vim
应该能在打开文件的时候自动识别它， 而 vi(1) 和 ex(1) 则需要在打开文件的时候通过 :set tabstop=4 来修正默认的设置。

想练练手吗？ 请参阅我们的 希望移植的软件列表 来看看您是否有兴趣完成其中的任务。

Chapter 3. 简单的 port

这一章将介绍如何快速创建一个全新的 port。 很多时候， 这点内容是不够的，
您需要阅读这份文档中更深入的内容。

首先， 需要取得包含源代码的 tar包， 并把它放到 DISTDIR变量所指的地方。 默认的情况下， 这应该是 /usr/ports/distfiles。



下面的内容假定您不需要修改软件的源代码就能在 FreeBSD 上编译通过。
如果需要修改代码， 就需要参考下一章的内容了。

3.1. 编写 Makefile

最简单的 Makefile 应该是这个样子的：

```
# New ports collection makefile for:   oneko
# Date created:      5 December 1994
# Whom:              asami
#
# $FreeBSD$
#

PORTNAME=     oneko
PORTVERSION=  1.1b
CATEGORIES=   games
MASTER_SITES= ftp://ftp.cs.columbia.edu/archives/X11R5/contrib/

MAINTAINER=   asami@FreeBSD.org
COMMENT=      A cat chasing a mouse all over the screen

MAN1=         oneko.1
MANCOMPRESSED= yes
USE_IMAKE=    yes

.include <bsd.port.mk>
```

看看您是否能够看懂。 不必担心 $FreeBSD$ 那一行， 当这个 port 被导入到 ports 树里的时候， CVS 会自动填写它。 您可以在 示范的 Makefile那章找到更多的细节。

3.2. 创建描述文件

有 2 个描述文件对于任何一个 port 来说是必须的， 不论它是不是打算成为 package。 它们是 pkg-descr 和 pkg-plist。 这两个文件使用 pkg- 前缀以区别于其它文件。

3.2.1. pkg-descr (关于 port 的冗长描述文件)

这是 port 里一个较长的描述文件。 使用一段或几段文件文字来简明的描述这个 ports 是用来做什么的。



这 不是 手册或者对如何 深入使用/编译这个port的说明！ 要是您从 README 或者联机手册里面中复制文字的话， 请务必小心； 通常， 它们不是对这个 port 简明扼要的描述， 或者用了难以使用的格式 (比如， 联机手册里有迫使两端对齐的空格)。 如果要移植的软件有官方的WWW网页， 您应该在这里列出来。 使用 WWW: 作为前缀来表示 一个网站， 这样其它的自动工具就能正常工作了。

下面是一个简单的 pkg-descr 例子：

```
This is a port of oneko, in which a cat chases a poor mouse all over
the screen.
:
(etc.)

WWW: http://www.oneko.org/
```

3.2.2. pkg-plist (port 的装箱单)

这份文件列出了 port 所要安装的所有文件。 由于 package 也是据此进行打包， 因此它也被称作 "装箱单(packing list)". 这个文件中， 路径是相对于安装的路径的 (通常是 /usr/local 或 /usr/X11R6)。 如果您使用 MANn 变量的话， 请不要在这里列出任何联机手册。 假如 port 在安装过程中会创建一些目录， 请务必增加对应的 @dirrm 行， 以便在 package 被卸载时予以自动删除。

下面是一个简单的例子：

```
bin/oneko
lib/X11/app-defaults/Oneko
lib/X11/oneko/cat1.xpm
lib/X11/oneko/cat2.xpm
lib/X11/oneko/mouse.xpm
@dirrm lib/X11/oneko
```

参考 pkg_create(1) 的联机手册以获得更多有关装箱单的细节



建议您将这个文件里的所有的文件名按字母排序。 这样， 在升级这个port的时候就能够更方便地核实所做的修改。



手工创建这样一份列表可能是一件非常枯燥的事情。 如果您的 port 需要安装大量的文件， 自动创建装箱单 会帮您省下不少时间。

只有一种情况可以不用 pkg-plist文件。 如果这个 port 只安装很少量的一些文件或目录的话， 这些文件和目录就可以分别列在 Makefile 的 PLIST_FILES和PLIST_DIRS 变量里。 举个例子来说， 我们可以在上面那个 oneko port 里面不用 pkg-plist， 而把下面的这几行加到 Makefile 里面：

```
PLIST_FILES=    bin/oneko \
        lib/X11/app-defaults/Oneko \
        lib/X11/oneko/cat1.xpm \
        lib/X11/oneko/cat2.xpm \
        lib/X11/oneko/mouse.xpm
```

```
PLIST_DIRS=    lib/X11/oneko
```

当然， 如果一个 port 不需要给它自己创建目录的话， 就不用设置 PLIST_DIRS 变量了。

不过， 如果用这种方式来列出 port 要安装的文件和目录的话， 也就无法利用在 pkg_create(1) 里介绍的命令来制作 package 了。 因此， 这种方法只适用于那些简单的 port， 使它们更为简化。 同时，这种做法也有助于减少 ports collection 中的文件数量。 在采用 pkg-plist 之前，请考虑一下使用这种方法。

稍后我们将看到 pkg-plist 以及 PLIST_FILES 如何处理 更复杂的任务。

3.3. 创建校验和文件

只要键入 make makesum， port 便会自动创建 distinfo文件。

如果下载的文件的校验和经常变化， 而您又能确保它们的来源可靠 (比如， 来自于CD制造商，或每天联编生成的文档文件)， 就应该在 IGNOREFILES 里面标明这些文件。 这样， 再运行 make makesum 的时候便不会把这些标记 IGNORE 的文件计算在内了。

3.4. 测试 port

应当确定您的 port 确实做了您希望它们做的事情， 包括打包。下面是需要重点检查的一些重要的工作。

- pkg-plist 中没有包括任何不想安装的文件
- pkg-plist 包含了所有应该安装的文件
- 您的 port 能够使用 reinstall 多次安装。
- 您的 port 能在卸载 (deinstall) 时， 自动完成 清理

Procedure: 推荐的测试顺序

1. make install
2. make package
3. make deinstall
4. pkg_add package-name
5. make deinstall
6. make reinstall
7. make package

确信在 package 和 deinstall 阶段没有任何警告。 第三步以后，检查是否所有新建的目录都被正确删除了。 在第四步以后， 试着运行一下所装的软件， 确保当它以 package 方式安装的时候也能正常工作。

自动化这些步骤最简单的方法是通过 ports tinderbox 来进行测试。 它可以维护 jails 并在其中完成全部测试工作， 而不会破坏正在运行的系统的状态。 请参见 ports/ports-mgmt/tinderbox 以了解更多的信息。

3.5. 用 portlint 来检查 port

请使用 portlint 命令来检查您的 port 是否符合我们的规范。 ports-mgmt/portlint 程序是 ports 套件的一部分。 这个程序的主要功能是帮助您检查 Makefile 的样式是否符合规范， 以及 package 的命名是否得体。

3.6. 提交新 port

在提交新 port 之前， 应先阅读 该做什么和不该做什么 一节。

既然已经对所制作的 port 相当满意了， 剩下的工作， 便是将它放进 FreeBSD 的主 ports 树，
以便让更多的人从中受益。 我们并不需要您的 work 目录以及 pkgname.tgz 包，
因此现在可以删除它们了。 假定您的 port 的名字是 oneko， 接下来要做的是 cd 到 oneko 所在的目录，
然后输入命令： shar find oneko > oneko.shar

将这个 oneko.shar 文件作为附件， 使用 send-pr(1) 程序提交 (请参阅 Bug Reports and General Commentary 以了解关于 send-pr(1) 的进一步详情) 将其送出。 请务必将您的 bug 报告分类 (category) 为 ports 并把子分类 (class) 设置为 change-request (不要把报告表及为机密的， 即 confidential！)。 此外，
在 PR 的描述 ("Description") 一栏中的内容应该是 port 的简要介绍 (例如 COMMENT 内容的简化版本)，
而 shar 文件则应填入修正 ("Fix") 栏中。



在问题报告里面使用了一段好的描述， 能使我们的工作变得更容易。 习惯上，
我们会使用类似： "New port: <category>/<portname> <short description of the port>" 这样的标题来说明这是新的 port。 如果您也使用这样的习惯，
那么我们将更容易更方便地阅读您的 PR， 从而加快处理速度。

再次声明， 不要包含原始的distfile， work目录， 或者您用 make package 制作的包； 此外， 对于新的 port 请务必使用 shar(1) 而不是 diff(1)。

在您提交的您的 port 以后请耐心等待。 有时在一个 port 正式加入 FreeBSD 之前需要花费好几个月，
尽管也有可能是几天。 您可以查看 正等待被 commit 到 FreeBSD 的 port PR。

一旦我们看过了您的报告， 有必要的话我们会联系您， 并把它放到 ports 树里。 您的名字也会出现在 Additional FreeBSD Contributors 和其它的文件。 不是很棒吗!? :-)

Chapter 4. 复杂的 Porting

好了， 也许工作没那么简单， port 需要做些修改才能够在 FreeBSD 上跑起来。 在这一章里，
我们将会一步步举例来介绍应该如何修改来使您的 port 能在 FreeBSD 上面运行。

4.1. 整个系统是如何运转的?

首先， 这一系列的动作是由用户在您的 port 目录里敲入 make 后发生的。
您也许会发现在另外的一个窗口里阅读一下 bsd.port.mk 将会有助于您的理解。

要是您不是非常明白 bsd.port.mk 是做什么的话， 也不用太担心， 很多人都不知道的… :→

1. fetch 会首先被执行。 fetch 将检查在本地的 DISTDIR 目录里是否存在 tar 包。 如果 fetch 没有找到就会查找 Makefile 中定义的 MASTER_SITES URL， 还有我们的主 FTP 站点 ftp://ftp.FreeBSD.org/pub/FreeBSD/ports/distfiles/， 在那里我们备份了所有被认可的 distfile。 假设那个 MASTER_SITES 站点是直接连在 Internet 上的， 就会试着用 FETCH 指定的程序取回 distfile。 如果成功的话， 文件会被保存在DISTDIR 所指定的目录以备稍后使用。
2. 接下来会执行 extract。 它会在 DISTDIR 中寻找您的 tar 包 (通常是用 gzip 压缩的 tar 包)，然后解压缩到由 WRKDIR 所指定的临时目录里 (默认为work目录)。
3. 下一步是执行 patch。 首先任何在 PATCHFILES 中定义的补丁都会被打上。 然后， 在由 PATCHDIR 指定的目录 (默认为 files目录) 中发现的patch-*，
它们将会以文件名的字母顺序被先后打上。
4. configure会被执行。 这一步骤可能会有以下几种情形。
   a. 如果存在 scripts/configure， 就会执行它
   b. 如果定义了 HAS_CONFIGURE 或者 GNU_CONFIGURE， 就会执行 WRKSRC/configure。
   c. 如果定义了USE_IMAGE， 就会执行 XMKMF (默认为: xmkmf -a)。
5. build会被执行。 这一步将会进入ports的工作目录 (WRKSRC)
然后进行编译。如果定义了USE_GMAKE， 就会使用 GNU make， 反之， 则会使用系统默认的 make。

以上都是系统默认的步骤。 您也可以定义 pre-something 或者 post-something，
或者把以此命名的脚本放到 scripts 目录， 它们会在默认的动作之前或之后被执行。

举个例子， 如果您在您的 Makefile 里定义了post-extract， 并在 script 目录里放了一个 pre-build 脚本，
那么在 tar 包解开之后 post-extract 将被调用， pre-build 脚本会在默认的编译之前被执行。 我们推荐您在 Makefile 定义所有的动作， 如果不是十分复杂的话， 这样， 别人能更容易明白您的 port
需要执行哪些非默认的动作。

默认的行为都是由 bsd.port.mk 定义的 do-something 来表示的。 例如， port 中用来解压缩的命令是由 do-extract 来定义的。 如果您对默认的设置不满意， 可以通过在 Makefile 重新定义 do-someting
来做些改变。



"主" 动作 (例如 extract、 configure， 等等) 仅仅是用来确定所有相应的阶段都完成了，
以及调用真实的动作或脚本， 它们不应被修改。 如果您想要修改解压缩这个动作，
可以修改 do-extract， 但永远都不要改变 extract 的操作!

我们已经介绍了在用户敲入 make 之后会发生哪些事情了。 接下来我们将进行进一步的学习，
来看一看如何创建一个理想的 port。

4.2. 获取源代码

获取源代码的 tar 包 (通常是 foo.tar.gz 或者 foo.tar.Z) 并把它们放进 DISTDIR。 最好使用 主流 的版本。

您需要设置变量 MASTER_SITES 来指向原始 tar 包的获取位置。 您可以在 bsd.sites.mk

里找到一些速度较快的主流站点。 请使用这些站点 - 和相关的定义 - 如果可能的话，
应尽量避免在同一个源代码树里出现大量重复的信息。 这些站点会随着时间而变化，
如果每个人都随意加入的话会使维护变得非常困难。

如果您找不到一个有很好网络连接的 FTP/HTTP 站点， 或者它们使用了非标准的格式，
您也许就会想在您自己的 FTP 或 HTTP 服务器上放上一份副本。

如果您找不到可靠的地方放置 distfiles， 我们也可以提供给您一些空间来保存它。 我们自己的
ftp.FreeBSD.org； 然而这只是一个折衷的办法。 distfile 必须放进某人在 freefall 上的 ~/public_distfiles/
目录中。 可以要求帮助您 commit port 的人来放这个 distfile， 而这个人也需要把 MASTER_SITES、
MASTER_SITE_LOCAL 以及 MASTER_SITE_SUBDIR 的设置， 改为在 freefall 上的用户名。

如果您的 port 的 distfile 一直在变化， 而作者拒绝改变其版本号， 您可以考虑把 distfiles
放在自己的主页， 并在 MASTER_SITES 里把原作者的列为首选位置。 如果可能， 试着与 port
的作者沟通一下让他不要这么做， 这将有助于建立对源代码的控制。 在您的主页上放置您自己的 distfile
会避免用户得到 checksum mismatch 的错误， 而且能减轻我们 FTP 站点维护人员的工作量。
如果您的port只有一个主站点的话， 我们建议您在自己的网站上做一份备份， 并他列为
MASTER_SITES的第2项。

如果您的 port 需要来自网络上的一些补丁， 请把它们放到 DISTDIR里。
不用担心它们跟源代码不是来自同一站点。 我们有办法处理 (参阅下面的 补丁文件)。

4.3. 修改 port

解开 tar 包， 对源代码做出合理的修改使得这个 port 能在最新版本的 FreeBSD 上面运行。 一定要
仔细记录 您所做的每处改动， 包括删除、添加、修改的文件等等， 这些修改以后会在您的 port
中以脚本或补丁的方式出现， 并且能通过运行它们来自动完成您对 port 的改动要求。

如果您的 port 要求用与用户交互/配置来完成编译或安装的话， 您可以看一下 Larry Wall 的经典的
Configure 脚本， 适当地模仿一下。 Port collection 的目的， 就是使每个 port 占用最少的空间，
并做到软件的 "即插即用"。



除非明确地声明， 否则您提交给 FreeBSD ports collection 的补丁，
脚本和其它的文件都将以标准的 BSD 版权发布。

4.4. 打补丁

在您准备制作 port 的过程中， 增加或修改的文件， 都可以通过 diff(1) 来做成补丁。
希望应用到源代码上的每个补丁， 都应保存为单独的文件， 并命名为 patch-*， 其中 *
表示将要修改的文件的完整路径名， 例如 patch-Imakefile 或 patch-src-config.h。 这些文件，
都应保存在 PATCHDIR (通常是 files/)， 这里的补丁都会自动应用到源代码上。 所有的补丁必须是相对于
WRKSRC 的 (一般而言， 您的 port 会将其 tarball 解压缩在那里， 并完成余下的工作)。
为了让修正和升级更容易， 您应避免使用多个 patch 去修改同一个文件 (例如， patch-file 以及 patch-file2
都修改 WRKSRC/foobar.c) 这种情况。 需要注意的是， 如果修改的文件的路径中包含下划线 (_) 字符，
则在补丁文件名中应使用两个下划线来代替。 例如， 如果需要修改名为 src/freeglut_joystick.c 的文件，
补丁文件的名字应为 patch-src-freeglut__joystick.c。

只有 [-+._a-zA-Z0-9] 这些字符， 可以出现在补丁的文件名中，
请务必不要使用除这些字符以外的其它字符。 不要把您的补丁命名成 patch-aa 或 patch-ab
等这样的名字， 最好能在补丁名中提到路径和文件名。

不要把 RCS 字符串放进补丁。 我们把文件放进 ports 树的时候， CVS 会损坏它们， 当我们再 check out
出来的时候， 它们就会和原来的不一样， 从而导致打补丁失败。 RCS 字符串 是由美元符号 ($) 围绕的，
通常由 $Id 或 $RCS 开头。

使用 diff(1) 的递归选项(-r) 很好， 但是请检查一下最后输出的 patch， 确保没有任何的垃圾信息。
特别地， 有 2 种文件不需要 diff， 并且应该删除： 一种是 Makefile， 当您的port使用了Imake， 或者
GNU configure 等等的话。 如果您不得不编辑configure.in 以使 autoconf 去生成 configure， 不要使用
configure 来做 diff (这常常会有好几千行长!)； 请定义 USE_AUTOTOOLS=autoconf:261 并对应
configure.in 来制作 diff。

另外， 您还应尽量减少补丁中非功能性的空格及空白变动。 在开源世界中，
遵循不同的编码规范的项目共享大量代码是很常见的事情。
如果您从某个项目中提取一部分功能用来修正另一个程序中的问题时， 请务必小心：
补丁中很可能到处都是非功能性的变动行。 这不仅会导致 CVS 库的膨胀， 而且也会让导致问题的故障点，
以及您到底修改了什么变得不甚清晰。

假如需要删除文件， 则应在 post-extract target， 而不是作为补丁的一部分来完成。

除此之外， port 的 Makefile 还可以通过 in-place 模式的 sed(1) 来直接进行简单的替换操作。
如果补丁需要使用变量值， 这就非常有用了。 例如：

```
post-patch:
    @${REINPLACE_CMD} -e 's|for Linux|for FreeBSD|g' ${WRKSRC}/README
    @${REINPLACE_CMD} -e 's|-pthread|${PTHREAD_LIBS}|' ${WRKSRC}/configure
```

往往在移植某些软件的时候会遇到这样一种情况， 特别是这个软件是在 Windows® 上开发的时候，
大多数的源代码都需要进行CR/LF的转换。 这很可能会给以后打补丁带来问题， 还可能触发编译警告，
并给脚本的执行带来麻烦 (/bin/sh^M not found)， 等等。 要迅速将所有文件中的 CR/LF 改为只用 LF，
可以在 port 的 Makefile 中加入 USE_DOS2UNIX=yes。 除此之外，
还可以指定一个需要执行这种转换操作的文件列表：

```
USE_DOS2UNIX=   util.c util.h
```

如果希望转换一系列目录中的一组文件， 也可以使用 DOS2UNIX_REGEX。 它的参数是与 find
兼容的正则表达式。 关于这种格式的说明， 请参阅 re_format(7)。
这个选项对转换所有指定扩展名的文件， 例如只转换源代码文件这样的应用非常有用：

```
USE_DOS2UNIX=   yes
DOS2UNIX_REGEX=  .*\.(c|cpp|h)
```

如果您希望基于现存的文件创建补丁， 可以把文件复制为带 .orig 扩展名的名字， 然后修改原文件。
然后使用 makepatch 目标根据修改在 port 的 files 目录中生成补丁文件。

4.5. 配置

把任何附加的配置命令加进您的 configure 脚本并把它保存到 scripts 子目录。 如前面提到的那样，
您也能在 Makefile 和/或 使用 pre-configure 或 post-configure 的脚本来做同样的事情。

4.6. 处理用户输入

如果您的 port 要求用户的输入以便配置编译、 或安装配置过程， 就必须在 Makefile 里设置
IS_INTERACTIVE 变量。 如果用户设置了 BATCH 的话， 这将让用户能跳过您的 port 来完成 "通宵编译"
(如果用户设置了 INTERACTIVE的话， 那么 只有 那些要求互动的 port 才会被编译) 这将给那些不停编译
ports 的机器省下很多时间。

通常我们还建议， 如果对于那些问题能有合理的缺省答案的话， 应检查一下 PACKAGE_BUILDING 变量，
并根据其设置决定是否执行关闭交互脚本。 这将允许我们为 CDROM 和 FTP 来编译 package。

Chapter 5. 配置 Makefile

配置 Makefile 是相当简单的， 我们在此建议您在开始之前看一下现有的例子。 在这份手册里也有一个 Makefile例子， 照着里面变量的顺序来写能使得您的 port 更容易地被其它人看懂。

现在， 当您开始编写您新的Makefile 的时候， 可以依次思考一下以下的问题：

5.1. 作者发布的代码

放在 DISTDIR 中的是不是标准的用 gzip 压缩的 tar 包， 例如 foozolix-1.2.tar.gz？ 如果是，
可以先略过这一节。 如果不是， 您应当看看是不是要覆盖这些变量： DISTVERSION、 DISTNAME、
EXTRACT_CMD、 EXTRACT_BEFORE_ARGS、 EXTRACT_AFTER_ARGS、 EXTRACT_SUFX，
DISTFILES，取决于您 port 的 distfile 格式有多么怪异。 (最常见的一个例子便是 EXTRACT_SUFX=.tar.Z，
一般这是因为 tar 包是用 compress 而不是 gzip 压缩的时候。)

最糟的情况是， 您需要自己编写 do-extract 来覆盖默认的定义， 尽管这不常见， 但如果遇到了，
还是需要这么做。

5.2. 命名

Makefile 的第一部分便是 port 的名字、 版本号， 以及它所属的分类。

5.2.1. PORTNAME 和 PORTVERSION

您应该把 PORTNAME 设置为您 port 的名字， PORTVERSION 则是 port 的版本号。

5.2.2. PORTREVISION 和 PORTEPOCH

PORTREVISION (port 的修订版本号)

PORTEREVISION 变量是一个单调递增的值， 如果不为 0， 就会被加到包名的后面， 当 PORTVERSION
增加 的时候应被置 0 (也就是当官方有新版本发布的时候)。 PORTREVISION 会被自动化工具 (比如
pkg_version(1)) 用来检测是否存在可用的新版本。

每当 port 发生变化并对生成的 package 的内容或结构有显著影响时， 都应增加 PORTREVISION 值。

下面是一些应当修改 PORTREVISION 的情况：

- 有新的补丁用来修正安全漏洞、 错误， 或给 port 添加了新的功能。
- 修改了 Makefile 里编译时开启或禁用的选项。
- 修改了要安装文件的列表或安装时的行为 (例如， 修改了一个用来给 package 初始化数据的脚本， 如 ssh host keys)。
- 一个port依赖的共享库版本改变 (在这种情况下， 当安装了新版本的共享库，
  后再去安装较早的软件就会出错， 因为它们要依赖老的 libfoo.x 而不是libfoo.(x+1))。
- 原作者修改了 port distfile， 并且 distfile 的新老版本之间用 diff -ru 只能发现一些细微的变化，
  这时我们只需要对 distinfo 做相应的修正， 而不需要修改 PORTVERSION。

不需要修改 PORTREVISION 的例子：

- port 结构风格的改变， 但对于打成的包没有功能的上的变化。
- MASTER_SITES 发生变化， 或进行了对 port 功能的修改， 但不致影响最后打成的包。
- 对 distfiles 诸如修正拼写错误之类的补丁， 对用户而言没有升级上的麻烦。
- 对一个原本编译失败的包的修改， 使其可编译， 而没有加入新功能。 因为 PORTREVISION
  表示包的内容发生了变化， 如果先前没有可编译的包， 也就不需要修改 PORTREVISION 来表示变化。

一个修改并提交 port 的原则是： 使得别人能从中受益 (改进、 修改已有错误， 或使新的 package

能够运行)， 您还要权衡一下这是否应让那些经常更新 ports 树的人升级， 如果回答是 "是" 的话，
PORTREVISION 就应该修改了。

PORTEPOCH (port 的加权版本号)

有时软件商或 FreeBSD 的 porter 会使用比旧版的版本号小的数字做为新版本号的情况。 举例来说， 从
foo-20000801 到 foo-1.0 (从形式上来说这是不对的， 因为 20000801 在数值上比1大很多)。

在这种情况下， PORTEPOCH 应当增加。 如果 PORTEPOCH 非 0， 就应当加到包名字的后面。
PORTEPOCH 永远不能被减少或清零， 因为那样会导致与前一时期的 package
比较版本时产生不正确的结果。 (就是说， 那个 package 就不会被检测到已经过时了。) 新的版本号
(比如前面在前面那个例子中的 1.0,1) 在数值上比前一个版本 (20000801) 小， 但多数自动化的工具会认为
,1 后缀意味着比前一个包的后缀 ,0 大。

错误的去除或重置 PORTEPOCH 会导致很多不幸发生； 如果您还不明白前面的讨论，
请多阅读几次直至明白为止， 或到邮件列表上来提问。

大多数 port 都不会用到 PORTEPOCH， 并且如果某个软件的下一个版本改变了版本号结构的话，
用巧妙的方法来设定 PORTVERSION 也能避免使用 PORTEPOCH。 然而， FreeBSD porter 也需要注意，
当有新版本的软件发布， 但并非正式版本时 - 比如 "snapshot" 版本，
原作者可能会使用当时的日期来命名， 这在新的 "官方" 版本发布的时候， 就很容易引起前面提到的问题。

举个例子， 如果 snapshot 版本的发布日期是 20000917， 这个软件的上一个版本是1.2， 那么这个版本的
PORTVERSIN 应该设为 1.2.20000917 或类似的样子， 而不是20000917， 这样在 1.3 发布以后，
新版本就可以在数值上大于旧的版本了。

关于 PORTREVISION 和 PORTEPOCH 的用例

gtkmumble port，版本号 0.10， 被提交到 ports collection:

```
PORTNAME=    gtkmumble
PORTVERSION=  0.10
```

PKGNAME 变成 gtkmumble-0.10。

然后有人发现了一个安全漏洞， 需要用一个FreeBSD的补丁。 PORTREVISION 就要相应的增加。

```
PORTNAME=    gtkmumble
PORTVERSION=  0.10
PORTREVISION=  1
```

PKGNAME变成了 gtkmumble-0.10_1

软件的作者发布了新的版本， 版本为 0.2 (作者本来的意思是， 用 0.10 表示 0.1.0，"而不是指 0.9
之后的那个版本" - 但是现在太迟了)。 因为现在的次版本号 2 在数值上比上一个版本 10 小， PORTEPOCH
必须增加， 以使新的 package 被认为是 "更新的"。 由于那是作者发布的一个新版本， 因此
PORTREVISION 应被置0 (或者从 Makefile 里面删除它)。

```
PORTNAME=    gtkmumble
PORTVERSION=  0.2
PORTEPOCH=    1
```

PKGNAME 变成了 gtkmumble-0.2,1

下一个版本将会是 0.3。 由于 PORTEPOCH 从不减少， 那么就无须改动：

```
PORTNAME=    gtkmumble
PORTVERSION=  0.3
PORTEPOCH=    1
```

PKGNAME 变成 gtkmumble-0.3,1



如果在这次升级中 PORTEPOCH 被置为了0，那么在装了 gtkmumble-0.10_1 包的机器上就无法检测到 gtkmumble-0.3 包的更新，因为 3 在数值上比 10 小。记住，这是 PORTEPOCH 最重要的地方。

5.2.3. PKGNAMEPREFIX 和 PKGNAMESUFFIX

2 个可选的变量，PKGNAMEPREFIX 和 PKGNAMESUFFIX 可以和 PORTNAME 还有 PORTVERSION 配合使用，形成像这样的 PKGNAME：${PKGNAMEPREFIX}${PORTNAME}${PKGNAMESUFFIX}-${PORTVERSION}。请确定符合我们的 包命名规则。当然，不允许在 PORTVERSION 中使用连字符 (-)。如果包名有 language- 或 -compiled.specifics 部分 (见下文)，请分别用 PKGNAMEPREFIX 和 PKGNAMESUFFIX，不要直接加到 PORTNAME 中。

5.2.4. LATEST_LINK

LATEST_LINK 在编译包的过程中用于确定可以为 pkg_add -r 使用的缩短的名字。举例来说，在安装最新版本的 perl 的时候，只需指定 pkg_add -r perl 而无需知道具体的版本号。这个名字应该是独一无二的，并且对用户而言应该是显而易见的名字。

有时，在 ports 套件中可能会存在同一程序的多个版本。索引和预编译包的联编系统都需要能够将它们视为不同的软件包，尽管其 PORTNAME、PKGNAMEPREFIX，甚至 PKGNAMESUFFIX 可能是一模一样的。遇到这种情况时，就需要将除了 "主" port 之外的其他 port 中的 LATEST_LINK 变量设为不同的值 - 请参见 lang/gcc46 和 lang/gcc port，以及 www/apache* 系列，以了解它的用法。如果设置了 NO_LATEST_LINK，则系统便不会生成对应的连接，对于非 "主" port 来说是一个可行的选择。需要注意的是，如何确定 "主" 版本 - "最流行"、"受支持最好"，"变动最少"，等等 - 已经超过了本书能够给出的建议范围；这里只是向您介绍在选定了一个 "主" port 之后如何指定其他 port 的版本。

5.2.5. 包命名规则

以下是您在命名您的包时应当遵守的规则。这将使得我们放包的目录更利于浏览，因为我们已经有数以万计的包了，如果用户觉得查看包名很困难的话，他们会很快走开的。

一个包的名字应该看起来像这样：language_region-name-compiled.specifics-version.numbers。

要像这样来定义包的名字：${PKGNAMEPREFIX}${PORTNAME}${PKGNAMESUFFIX}-${PORTVERSION}。确保所有的变量符合上面的格式。

1. FreeBSD 会尽力去支持用户当地的语言。如果这个 port 是某种语言专用的，那么 language- 部分应该是 由 ISO-639 定义的自然语言的 2 个字母缩写。比如，ja是表示日本，ru 是表示俄罗斯，vi 表示越南，zh 表示中国，ko 表示韩国，de 表示德国。

   如果是针对某种语言的某一地区的话，再要加上2个字母的国家代码。例如，en_US 表示美国英语，fr_CH 表示瑞士法语。

   language- 部分应该在 PKGNAMEPREFIX 变量里设置。

2. name 部分的首字母应该 小写。(余下的部分可以包含大写字母，所以当您要转换一个包含大写字母软件的名字时，您需要 自己做出判断。) 对于 Perl 5 模块的命名，有个传统的规则是，在前面 加上 p5- 并把两个冒号的部分改为连字号，如：Data::Dumper 模块对应的名字，就应该是 p5-Data-Dumper。
3. 确认 port 的名字和版本之间有清晰的分隔，并放入 PORTNAME 和 PORTVERSION 变量。在 PORTNAME 中包含版本部分的唯一理由是上游软件包真的采用这样的命名方式，类似

textproc/libxml2 或 japanese/kinput2-freewnn port 这样。 否则， 在 PORTNAME 中就不应包含任何版本信息。 许多 port 采用同样的 PORTNAME 名字是很正常的， www/apache* port 便是如此； 在这种情况下， 不同的版本 (以及不同的索引项) 是由 PKGNAMEPREFIX、PKGNAMESUFFIX， 以及 LATEST_LINK 的值的不同而有所区别的。

4. 如果 port 可以使用不同的 硬编码默认配置 进行联编 (通常是一系列 port 的一部分目录名)， 则 -compiled.specifics 部分就应该明示编译进去的默认值 (此处连字号是可选的)。 通常的用例包括纸型和不同的字体尺寸。

   -compiled.specifics 部分应该通过 PKGNAMESUFFIX 变量来设置。

5. 版本号应该紧随在连字号 (-) 后面并由数字和字母组成。 特别指出， 另外的连字号是不允许出现在版本号里的。 唯一例外的是字符串 pl (表示 "patchlevel")， 只能用在软件没有主版本号和次版本号的情况下。 如果软件的版本号里出现了像 "alpha"， "beta"， "rc"， "pre"， 取第一个字母把它放在小数点的后面。 如果在版本号里一直出现那些名字， 那么在数字和字母之间不应有多余的小数点。

   这个方法是为了更容易得凭版本号来排序 port。 特别注意的是， 确保版本号之间的每部分都由小数点来分隔， 如果日期也是版本号的一部分， 就用这样的格式， 0.0.yyyy.mm.dd 这样的格式， 而非 dd.mm.yyyy 甚至 yy.mm.dd 这种不适合表示千年的格式。 在版本号上使用 0.0. 前缀十分重要， 因为当软件发行正式的版本时， 其版本号数字很可能会小于表示年份的 yyyy 数字。

这里是一些真实的例子， 我们藉此说明如何把软件作者对软件的命名， 转换为适合我们包的命名方式：

发行版的名字	PKGNAMEPREFIX	PORTNAME	PKGNAMESUFFIX	PORTVERSION	说明
mule-2.2.2	(空)	mule	(空)	2.2.2	没什么需要修改的
EmiClock-1.0.2	(空)	emiclock	(空)	1.0.2	程序的名字不能使用大写字母
rdist-1.3alpha	(空)	rdist	(空)	1.3.a	像 alpha 这样的字符串是不允许出现的
es-0.9-beta1	(空)	es	(空)	0.9.b1	像 beta 这样的字符串是不允许出现的
mailman-2.0rc3	(空)	mailman	(空)	2.0.r3	像 rc 这样的字符串是不允许出现的
v3.3beta021.src	(空)	tiff	(空)	3.3	那个是啥鬼东西？
tvtwm	(空)	tvtwm	(空)	pl11	总需要有个版本号吧
piewm	(空)	piewm	(空)	1.0	总需要有个版本号吧
xvgr-2.10pl1	(空)	xvgr	(空)	2.10.1	pl 只允许在没有主/次 版本号的情况下才能出现
gawk-2.15.6	ja-	gawk	(空)	2.15.6	日文版
psutils-1.13	(空)	psutils	-letter	1.13	纸张大小已经在编译的时候被硬编码到程序里了
pkfonts	(空)	pkfonts	300	1.0	300dpi 字体的包

如果在原始的代码里没有版本号， 或者原作者并不打算开发另外的版本， 就应把版本号设成 1.0 (就像前面 piewm 的例子那样)。 否则， 要求原始的作者加上版本号或使用日期 (0.0.yyyy.mm.dd) 来作为版本号。

5.3. 分类

5.3.1. CATEGORIES (所属分类)

在包制作完成之后， 它会被放在 /usr/ports/packages/All， 并建立一系列来自 /usr/ports/packages 下子目录的符号连接。 这些子目录的名称是由 CATEGORIES 指定的。 这将方便于那些用户在 FTP 站点或 CDROM 的一大堆包里面寻找自己想要的包。 请查看一下 目前的分类表， 并找出一个适合您 port 的分类。

此列表也会决定您的 port 在 port 目录中的位置。 如果您在这里设定了 1 个以上的分类， 则认为您 port 文件应放到以第一个分类命名的子目录中。 请参阅 后面 关于如何选择正确分类的更多讨论。

5.3.2. 目前的分类表

这是目前 port 中的分类。 那些用星号 (*) 标记的是 虚拟分类 - 它们在ports树里没有相应的子目录， 因而只用来做为次要的分类， 用以方便搜索。



对于非虚拟的分类来说， 您会看到在相对应子目录中的 Makefile 里有写在 COMMENT 里的单行描述。

分类	描述	注意事项
accessibility	帮助残障人士的 port。	
afterstep*	对于 AfterStep 窗口管理器的支持。	
arabic	阿拉伯语言支持。	
archivers	压缩与备份工具。	
astro	有关天文学的 port。	
audio	声音支持。	
benchmarks	测评程序。	
biology	生物学相关的软件。	
cad	计算机辅助设计工具。	
chinese	中文语言支持。	
comms	通讯软件。	大部分是用于串口通讯的。
converters	字符编码转换。	
databases	数据库。	
deskutils	在发明计算机以前就已经在桌面上使用的东西。	
devel	程序开发工具。	不要把开发库放在这里 - 除非您再也找不到更合适的分类，否则就不该放在这个分类里。
dns	DNS 相关的软件。	
docs*	有关 FreeBSD 文档的 Meta-ports。	
editors	通用编辑器。	有特殊用途的编辑器应该被置于相应的分类中 (比如， 数学-方程式编辑器应该放在 math 分类里。
elisp*	Emacs-lisp相关的port。	

分类	描述	注意事项
emulators	其它操作系统的模拟器。	终端模拟器 不应该 属于这个分类 - 基于 X 的应该放在 x11 而基于文本模式的应该放到 comms 或 misc 中去，取决于具体的功能。
finance	货币、 金融以及相关的应用程序。	
french	法语语言支持。	
ftp	FTP 客户端和服务器端的程序。	如果您的 port 同时支持 FTP 和 HTTP 的话， 把它放进 ftp 并把 www 做为第二分类。
games	游戏。	
geography*	与地理学有关的软件。	
german	德语语言支持。	
gnome*	关于 GNOME 项目的支持。	
gnustep*	与 GNUstep 桌面环境有关的软件。	
graphics	图形图象程序。	
hamradio*	业余无线电爱好者使用的软件。	
haskell*	有关 Haskell 编程语言的软件。	
hebrew	希伯来语语言支持。	
hungarian	匈牙利语语言支持。	
ipv6*	IPv6 相关软件。	
irc	IRC 相关程序	
japanese	日语语言支持。	
java	与 Java™ 编程语言有关的软件。	java 分类对 port 而言不应是其唯一的分类。 除了直接与 Java 语言相关的 port 之外， 开发人员应尽量避免使用 java 作为 port 的主分类。
kde*	K 桌面环境 (KDE) 相关的软件。	
kld*	可加载内核模块。	
korean	韩语语言支持。	
lang	编程语言。	
linux*	Linux 相关的应用程序。	
lisp*	和 Lisp 编程语言有关的软件。	
mail	电子邮件软件。	
math	数值计算和其它数学相关的软件。	
mbone*	MBone 应用程序。	
misc	各式各样的实用程序。	通常不属于其它的任何分类， 如果可能的话， 尽量为您的 port 选择 misc 以外的分类， 因为在这里的 port 比较容易被人忽略。
multimedia	多媒体软件。	
net	各种网络相关的软件。	
net-im	即时消息软件。	

分类	描述	注意事项
net-mgmt	网络管理软件。	
net-p2p	对等网 (Peer to peer network) 应用程序。	
news	USENET新闻组相关软件。	
palm	Palm™ 系列相关软件。	
parallel*	并行计算相关软件。	
pear*	Pear PHP 架构相关软件。	
perl5*	Perl5 相关的软件。	
plan9*	Plan9 相关程序。	
polish	波兰语语言语言支持。	
ports-mgmt	用于管理、 安装和开发 FreeBSD ports 和预编译包的 port。	
portuguese	葡萄牙语语言支持。	
print	打印相关的软件。	桌面出版工具 (打印预览工具等等) 也可以放在此分类里。
python*	Python 编程语言相关的软件。	
ruby*	Ruby 编程语言相关的软件。	
rubygems*	移植版本的 RubyGems 软件包。	
russian	俄语语言支持。	
scheme*	与 Scheme 语言有关的 port。	
science	科学相关但不适合放在 astro、biology， 以及 math 分类的 port。	
security	安全相关的实用程序。	
shells	命令行 shell。	
spanish*	西班牙语支持	
sysutils	系统相关的实用程序。	
tcl*	依赖于 Tcl 运行的 port。	
textproc	文本处理的实用程序。	这个分类并不适合于那些应该放到 print 的桌面出版工具。
tk*	依赖于 Tk 运行的 port。	
ukrainian	乌克兰语语言支持。	
vietnamese	越南语语言支持。	
windowmaker*	WindowMaker 窗口管理器的相关支持。	
www	Word Wide Web的相关软件。	HTML语言相关的支持也可以放在这个分类里。
x11	X Window System以及相关软件。	这个分类是给那些直接支持X Window System 的软件的。 不要把常规的 X 应用程序也放进这里； 它们中的大多数都应被归类到 x11-* (参见下文)。 如果您的 port 是 X 应用程序， 应定义 USE_XLIB (使用 USER_IMAKE 隐含包括它)， 然后把它放到合适的分类里。

分类	描述	注意事项
x11-clocks	X11 下的时钟程序。	
x11-drivers	X11 驱动程序。	
x11-fm	X11 下的文件管理器。	
x11-fonts	X11 下的字体以及相关工具。	
x11-servers	X11 服务器。	
x11-themes	X11 主题。	
x11-toolkits	X11 工具包。	
x11-wm	X11 窗口管理器。	
xfce*	与 Xfce 桌面环境有关的 port。	
zope*	Zope 相关的支持。	

5.3.3. 选择正确的分类

由于不少分类是重复的， 您通常在用哪个分类作为您 port 的主分类上做出选择。
下面有几条规则能帮您解决这个问题。 这是一个带优先级的表， 按优先级降序罗列：

- 第一个分类必须是个物理的分类 (参阅 前面)。 这对于制作包是必要的。
  虚拟分类和物理分类可能在包制作完成后混合在一起。
- 对于特定语言的分类通常放在第一位。 例如， 如果您的 port 会安装一些 X11 的日文字体， 那么 CATEGORIES那行 就应该是 japanese x11-fonts。
- 有特定意义的分类应当被列在无特定意义的前面。 例如， HTML 编辑器应该是这样的 www editors，而不是其它的什么。 同样地， 您不应该列出 net， 如果 port 属于 irc、 mail、 news、 security， 或是 www， 因为 net 可以表示它们的超集。
- 只有当主要的分类是一门自然语言的时候， x11 能被做为第二分类。 需要特别指出的是， 您不应把 X 的应用程序也归类为 x11。
- Emacs 模式应当于相应的应用程序放在同一个分类里， 而不是 editors 分类。 举例来说，
  一个用于编辑某种编程语言源代码的 Emacs 模式应该被归为 lang 一类。
- 需要安装可加载内核模块的 port 应在其 CATEGORIES 中归入虚拟分类 kld。
- misc 分类的 port 不能有其它非虚拟的分类。 如果您在您的 CATEGORIES 里设了 misc 和另外的分类，那意味着可以安全地删除 misc 并把 port 放到其它的子目录中了！
- 如果您的 port 确实不属于现有的分类， 才把它放到 misc。

如果您不能确定使用哪个分类， 请在您提交的 send-pr(1) 里加上一行注释， 这样我们就能在导入进 port 树之前讨论一下。 如果您是 committer， 发一份备忘到 FreeBSD ports 邮件列表 先讨论一下。
很多情况是新的 port 被加到错误的分类里， 然后又立即被移走。这会造成源代码库不必要和不良的膨胀。

5.3.4. 如何提议建立新的分类

由于 Ports Collection 在持续增长， 已经引入了许多新的分类。 新的分类既可以是 虚拟的 分类 - 这些分类在整个 ports 目录中没有属于自己的子目录 - 或 物理的 分类 - 它们有自己的子目录。
接下来我们将讨论与建立新的物理分类有关的事项， 以便帮助您理解如何提议建立新的分类。

我们目前的做法是避免建立新的物理分类， 除非有非常多的 port 应被归入这一分类， 或者 port 属于某一特定的小团体 (例如， 与某种人类语言相关)， 或两者皆是。

这样做的原因是这类修改会让 committer 和用户都不得不进行 许多工作 来在 Ports Collection 进行或追踪修改。 此外， 提议新的分类通常都会引起争论。 (可能这是因为关于某个分类是否 "太大" 一直没有非常一致的意见的缘故， 另一方面， 分类是否能够能够有助于浏览 (以及多少个分类是合适的)，等等， 也都是问题。)

下面是具体的步骤：

1. 在 FreeBSD ports 邮件列表 提议新的分类。 您应提供建立新分类的详细依据，
包括为什么认为现有的分类不够， 以及希望移动位置的一系列 port 的名字。 (如果有尚在 GNATS
而未 commit 的 port， 也应一一列出。) 如果您是相关 port 的监护人或提交者，
说明这一情况可能有助于您的提议得到通过。
2. 参与讨论。
3. 如果有人支持您的建议， 应及时提交一个 PR， 其中包括提议 PR 的理由， 以及需要移动的 port
的列表。 理想情况下， 这个 PR 也应包含针对下列文件的补丁：
   - 进行 repocopy 之后对 Makefile 进行的修改
   - 新分类的 Makefile
   - 旧分类的 Makefile
   - 依赖于旧 port 的 port 的 Makefile
   - (此外， 作为一项加分因素， 您还可以按照 Committer 指南所介绍的流程，
   提供一些其它需要修改的文件。)
4. 由于这是一项影响 ports 基础设施的变动， 它不仅涉及 repo-copy 的使用，
而且也可能会影响联编集群的回归测试操作， 因此这类 PR 应分派给 Ports 管理团队
<portmgr@FreeBSD.org>。
5. 如果这一 PR 得到批准， 某个 committer 将按照在 Committer 指南
中所介绍的步骤来完成余下的工作。

提议新的虚拟分类和上述过程类似， 但会容易许多， 因为不需要实际地移动任何 port。 这种情况下， PR
应附带的补丁， 就只需要修改影响到的 port 的 Makefile， 以便在其中的 CATEGORIES
中加入新的分类了。

5.3.5. 如何提议对分类进行重新组织

有些时候会有一些人提议重新将分类组织为 2-层 或某种基于关键字的结构。 目前为止，
还没有进行任何相关的改变， 因为尽管这些修改比较容易完成， 但修改整个 Ports Collection
所需要进行的工作， 至少也是令人生畏的。 在发表您的观点之前，
请阅读在邮件列表存档中历史上所进行过的提议； 此外， 您也会被要求提供一个可用的原形。

5.4. 源码包文件

在 Makefile 中的第二部分是描述用于联编 port 所必需下载的文件， 以及到什么地方去下载它们。

5.4.1. DISTVERSION/DISTNAME (源码包版本号/名称)

DISTNAME 是作者称呼您所 port 软件的名字。 DISTNAME 的默认值是 ${PORTNAME}-
${PORTVERSION}， 因此只有在需要时才应手工指定。 DISTNAME 只在两个地方用到。
第一处是源码包文件列表 (DISTFILES)， 其默认值是 ${DISTNAME}${EXTRACT_SUFX}。
第二处是源码包应被展开到的目录名， 即 WRKSRC 所指定的目录， 其默认值是 work/${DISTNAME}。

某些软件作者发布源码包的时候并不采取 ${PORTNAME}-${PORTVERSION} 这样的模式， 这可以通过设置
DISTVERSION 来自动处理。 PORTVERSION 和 DISTNAME 会自动地展开， 当然， 也可以改掉它。
下表给出了一些例子：

DISTVERSION	PORTVERSION
0.7.1d	0.7.1.d
10Alpha3	10.a3
3Beta7-pre2	3.b7.p2
8:f_17	8f.17



PKGNAMEPREFIX 和 PKGNAMESUFFIX 并不影响 DISTNAME。 此外还应注意 WRKSRC

> 等于 work/${PORTNAME}-${PORTVERSION}，而源代码的压缩包则可能是 ${PORTNAME}-${PORTVERSION}${EXTRACT_SUFX} 以外的其它名字。
> 一般情况下应该保持 DISTNAME 不变 - 更好的方法是定义 DISTFILES 而不是同时设置 DISTNAME 和 WRKSRC (可能还有 EXTRACT_SUFX)。

5.4.2. MASTER_SITES (主流下载站点)

记录 FTP/HTTP-URL 指向 MASTER_SITES 中原始压缩档的目录部分。 不要忘了结尾的斜线 (/)!

make 宏将尝试使用 FETCH 来抓取所指定的源码包文件，如果无法在本地系统中找到这些文件的话。

建议您指定多个镜像站点，最好是在不同的大洲上的。
这样将有效地防止由于大范围网络问题所导致无法下载的问题。
我们甚至打算增加自动检测距离最近的站点并从那里下载的功能；使用多个站点是这样做的重要一步。

如果原始的源码包可以从比较流行的软件下载站点，例如 SourceForge、GNU 或是 Perl CPAN 等等来获得，您可能会希望使用类似 MASTER_SITE_* 这样的缩写来表示它们 (例如 MASTER_SITE_SOURCEFORGE、MASTER_SITE_GNU 以及 MASTER_SITE_PERL_CPAN)。只需将 MASTER_SITES 设为这些变量，并使用 MASTER_SITE_SUBDIR 来指定路径就可以了。下面是一个例子：

```
MASTER_SITES=     ${MASTER_SITE_GNU}
MASTER_SITE_SUBDIR=  make
```

此外，您还可以用更为简略的格式：

```
MASTER_SITES=  GNU/make
```

这些变量是在 /usr/ports/Mk/bsd.sites.mk 中定义的。新项目会随时增加，因此在您提交 port 之前，应先看一看这个文件的最新版本。

针对常用软件下载站的许多 暗黑魔法 宏，还能够自动判断目录的结构。对于这些站点，只要使用与之对应的缩写，系统便会自动为您生成相关的子目录配置。

```
MASTER_SITES=  SF
```

如果系统猜测的路径不对，则可以使用下面这样的配置来替换。

```
MASTER_SITES=  SF/stardict/WyabdcRealPeopleTTS/${PORTVERSION}
```

表 1. 常用的魔术 MASTER_SITES 宏

宏	自动猜测的子目录
APACHE_JAKARTA	/dist/jakarta/${PORTNAME:S,-,,/,}/source
BERLIOS	/${PORTNAME:L}
CHEESESHOP	/packages/source/source/${DISTNAME:C/(.).*/\1/}/${DISTNAME:C/(.*)-[0-9].*/\1/}
DEBIAN	/debian/pool/main/${PORTNAME:C/^((lib)?.).*$/\1/}/${PORTNAME}
GCC	/pub/gcc/releases/${DISTNAME}
GNOME	/pub/GNOME/sources/${PORTNAME}/${PORTVERSION:C/^(\.[0-9]).*/\1/}

宏	自动猜测的子目录
GNU	/gnu/${PORTNAME}
MOZDEV	/pub/mozdev/${PORTNAME:L}
PERL_CPAN	/pub/CPAN/modules/by-module/${PORTNAME:C/-.*//}
PYTHON	/ftp/python/${PYTHON_PORTVERSION:C/rc[0-9]//}
RUBYFORGE	/${PORTNAME:L}
SAVANNAH	/${PORTNAME:L}
SF	/project/${PORTNAME:L}/${PORTNAME:L}/${PORTVERSION}

5.4.3. EXTRACT_SUFX (压缩包所用的扩展名)

如果您有一个源码包文件， 而它使用了某种怪异的扩展名来表达压缩方法， 应设置 EXTRACT_SUFX。

例如， 如果源码包文件的名字是 foo.tgz 而非更为一般的 foo.tar.gz， 您应写上：

```
DISTNAME=    foo
EXTRACT_SUFX= .tgz
```

USE_BZIP2 和 USE_ZIP 变量会自动根据需要将 EXTRACT_SUFX 设置为 .tar.bz2 或 .zip。
如果这两个都没设置， 则 EXTRACT_SUFX 的 默认值将是 .tar.gz。



任何时候都不需要同时设置 EXTRACT_SUFX 和 DISTFILES.

5.4.4. DISTFILES (全部源代码包)

有些时候所下载的文件名字和 port 的名字没有任何联系。 例如， 可能是 source.tar.gz，
或者与此类似的其它名字。 也有一些其它的应用软件， 它们的源代码可能被存放到了不同的压缩包中，
而且全都需要下载。

如果遇到这种情况， 可以将 DISTFILES 设置为以空格分隔的一组需要下载的文件列表。

```
DISTFILES=    source1.tar.gz source2.tar.gz
```

如果没有予以明确的设置， DISTFILES 的默认值将是 ${DISTNAME}${EXTRACT_SUFX}。

5.4.5. EXTRACT_ONLY (只解压缩部分源文件)

如果只有一部分 DISTFILES 需要解压缩 - 例如， 其中的一个是源代码， 而其它则是未压缩的文档 -
此时应把那些需要解压缩的文件加到 EXTRACT_ONLY 中。

```
DISTFILES=    source.tar.gz manual.html
EXTRACT_ONLY= source.tar.gz
```

如果 DISTFILES 中 没有 需要解压缩的文件， 则应将 EXTRACT_ONLY 设为空串。

```
EXTRACT_ONLY=
```

5.4.6. PATCHFILES (通过下载得到的补丁文件)

如果您的 port 需要来自 FTP 或 HTTP 的一些额外的补丁， 应将 PATCHFILES 设置为这些文件的名字，
并将 PATCH_SITES 指向包含这些文件的目录的 URL (格式与 MASTER_SITES 相同)。

如果这些补丁， 由于包含了其它的目录名， 而导致它们不是相对于源代码目录的顶级目录 (也就是
WRKSRC) 的话， 就需要相应地设置 PATCH_DIST_STRIP 了。 例如，
如果补丁中所有的目录名前面都有一个多余的 foozolix-1.0/， 就应设置 PATCH_DIST_STRIP=-p1。

不需要担心补丁文件本身是否是压缩的； 如果文件名以 .gz or .Z 结尾， 系统会自动解压缩。

如果补丁是同某些其它文件， 例如文档， 一同以 gzip 压缩的 tar 格式发布的， 就不能简单地使用
PATCHFILES 了。 这种情况下， 您应将这些补丁包的文件和位置加入到 DISTFILES 和 MASTER_SITES
中。 然后， 用 EXTRA_PATCHES 变量来指出这些文件， 这样 bsd.port.mk
就会自动地为您应用这些补丁了。 需要特别注意的是， 不要 将补丁文件复制到 PATCHDIR 目录中 -
这个目录可能是不可写的。



压缩包会以同源代码一样的方式解压缩， 因此不需要自行完成解压缩操作，
并复制补丁文件。 如果您一定要这样做， 就要注意，
不要让解压缩出来的文件覆盖先前已经存在的文件。 此外， 这么做还需要手工增加命令，
以便在 pre-clean target 中删除这些复制出来的文件。

5.4.7. 来自不同站点的多个源代码包或补丁文件 (MASTER_SITES:n)

(这一节在某种程度上应被视作 "进阶话题"； 刚开始阅读这份文档的读者可能会希望先跳过这一部分)。

这一节提供了被称作 MASTER_SITES:n 和 MASTER_SITES_NN 的下载控制机制。 这里我们把它们称为
MASTER_SITES:n。

首先给出一些背景。 OpenBSD 在其 DISTFILES 和 PATCHFILES 变量中提供了一个很棒的功能， 即，
允许这些文件和补丁拥有 :n 后缀， 其中 n 可以使用 [0-9]， 来表达组。 例如：

```
DISTFILES=    alpha:0 beta:1
```

在 OpenBSD 中， 源码包文件 alpha 应被关联到变量 MASTER_SITES0 而不是公共的 MASTER_SITES
变量上； 而 beta 则应关联到 MASTER_SITES1 上。

这是一个很有意思的功能， 它可以避免无休止地搜索正确的下载站点的过程。

想象 DISTFILES 中指定了 2 个文件， 而 MASTER_SITES 包含了 20 个站点的情形，
这其中许多站点慢如蜗牛， 而 beta 可以在 MASTER_SITES 的所有站点找到， 而 alpha 只能在第 20
个上面找到。 如果监护人了解这一点， 那么检查所有的站点无疑是在浪费时间， 不是吗?
这显然不是开始一个愉快周末的好办法!

现在您有了一个感性的认识了， 想象一下 DISTFILES 和更多的 MASTER_SITES。 显然， 我们的 "distfiles
调查员先生" 会感谢您减少他浪费在等待下载上所耗费的时间。

下一节中， 将按照 FreeBSD 对上述想法的实现来加以阐释。 我们对 OpenBSD
所提出的概念进行了一些改进。

简化信息

这一节将介绍如何迅速地对从不同的站点以及子目录下载多个源码包和补丁进行精确的控制。 这里，
我们将描述 MASTER_SITES:n 的一种简化用法。 对于多数情况而言这样做是足够的。 然而，
如果您需要更多信息， 还需要参考下面的几节。

一些应用程序需要从多个站点下载不同的源码包。 例如， Ghostscript 包括了程序核心本身，
以及大量的驱动文件， 以及则取决于用户的打印机品牌和型号的驱动程序。
某些驱动文件已经随程序核心附带， 但也有很多需要从其它站点下载。

为了适应这种需要， 每一个 DISTFILES 项应跟随一个冒号， 以及一个 "标签名"。 在 MASTER_SITES
的每个站点也应跟随冒号和标签名， 以便指定从哪个网站下载源码包文件。

例如， 考虑一个将源代码包分为两部分， 即 source1.tar.gz 和 source2.tar.gz 的软件，
它必须从两个不同的站点下载。 port 的 Makefile 应包括类似 简化的 MASTER_SITES:n 用法，
每个文件来自一个站点 的配置。

例 1. 简化的 MASTER_SITES:n 用法， 每个文件来自一个站点

```
MASTER_SITES=  ftp://ftp.example1.com/:source1 \
    ftp://ftp.example2.com/:source2
DISTFILES=    source1.tar.gz:source1 \
    source2.tar.gz:source2
```

多个源码包可以使用同一个标签。 继续前面的例子， 假定增加了第三个源码包， source3.tar.gz， 应从
ftp.example2.com 下载。 Makefile 的这部分应写成 简化的 MASTER_SITES:n 用法，
其中同一个站点上提供了不止一个文件 的样子。

例 2. 简化的 MASTER_SITES:n 用法， 其中同一个站点上提供了不止一个文件

```
MASTER_SITES=  ftp://ftp.example1.com/:source1 \
    ftp://ftp.example2.com/:source2
DISTFILES=    source1.tar.gz:source1 \
    source2.tar.gz:source2 \
    source3.tar.gz:source2
```

深入介绍

前面的例子无法满足您的需求？ 这一节， 我们将详细介绍 MASTER_SITES:n 的精细控制是如何工作的，
以及如何修改您的 port 来使用它们。

1. 元素可以包含 :n 这样的后缀， 其中 n 是 `， 概念上即 _n_ 可以取任意数字或字母，
   但我们目前将其限定为 `[a-zA-Z_][0-9a-zA-Z_]。

   此外， 字符串匹配时对大小写是敏感的； 换言之， n 与 N 不同。

   但是， 由于表达特殊的意义， 下列单词不能用于后缀： default、 all 和 ALL (它们会在 ii
   中介绍的部分用到)。 此外， DEFAULT 是一个有特殊用途的词 (请参见 3)。

2. 后缀为 :n 的项目属于 n 组， 而 :m 属于 m 组， 依此类推。
3. 没有后缀的元素是无组的， 也就是它们都属于那个特殊的 DEFAULT 组。 给元素加入 DEFAULT
   后缀通常是多余的， 除非您有同时属于 DEFAULT 和其它组的元素 (参见 5)。

   下面的例子是等价的， 但通常应适用第一个：

   ```
   MASTER_SITES=  alpha
   ```

```
MASTER_SITES=  alpha:DEFAULT
```

4. 组之间不是互斥的， 同一元素可以同时隶属于多个组， 而组则可以为空或者有任意多个元素。 同一组中的重复元素， 并不会被自动消去。
5. 如果希望同一元素同时属于多个组， 可以用逗号 (,) 分开。

   这种办法可以避免仅为指定不同的组而多次重复同一元素。 例如 :m,n,o 表示这个元素同时属于 m、 n 和 o 这三组。

   下面这些写法都是等价的， 但只推荐使用最后一种：

```
MASTER_SITES=  alpha alpha:SOME_SITE

MASTER_SITES=  alpha:DEFAULT alpha:SOME_SITE

MASTER_SITES=  alpha:SOME_SITE,DEFAULT

MASTER_SITES=  alpha:DEFAULT,SOME_SITE
```

6. 同一组中的所有站点， 会根据 MASTER_SORT_AWK 排序。 在 MASTER_SITES 和 PATCH_SITES 中的组也会进行排序。
7. 在 MASTER_SITES、 PATCH_SITES、 MASTER_SITE_SUBDIR、 PATCH_SITE_SUBDIR、 DISTFILES， 以及 PATCHFILES 中， 都可以使用组， 其语法为：

   a. 所有 MASTER_SITES、 PATCH_SITES、 MASTER_SITE_SUBDIR 以及 PATCH_SITE_SUBDIR 的元素， 都必须以 / 字符结尾。 如果有元素属于某些组， 则组后缀 :n 必须出现在终结符 / 之后。 MASTER_SITES:n 机制依赖于 / 的存在， 以避免在 :n 是元素一部分， 而 :n 同时又表示组 n 时发生混淆。 为了兼容性的考虑， 因为之前 / 终结符在 MASTER_SITE_SUBDIR 和 PATCH_SITE_SUBDIR 元素中都不是必需的， 如果后缀所紧跟的字符不是 /， 则 :n 将被认为是元素的一部分， 而不被当作组后缀， 即使元素拥有 :n 后缀。 请参见 在 MASTER_SITE_SUBDIR 中 MASTER_SITES:n 的详细用法 和 用到逗号分隔符、 多个文件， 多个站点和 不同子目录的 MASTER_SITES:n 详细用法 以了解进一步的细节。

   例 3. 在 MASTER_SITE_SUBDIR 中 MASTER_SITES:n 的详细用法

```
MASTER_SITE_SUBDIR=    old:n new/:NEW
```

   - 组 DEFAULT 中的目录 → old:n
   - 组 NEW 中的目录 → new

   例 4. 用到逗号分隔符、 多个文件， 多个站点和 不同子目录的 MASTER_SITES:n 详细用法

```
MASTER_SITES=  http://site1/%SUBDIR%/ http://site2/:DEFAULT \
    http://site3/:group3 http://site4/:group4 \
    http://site5/:group5 http://site6/:group6 \
    http://site7/:DEFAULT,group6 \
    http://site8/%SUBDIR%/:group6,group7 \
    http://site9/:group8
```

```
DISTFILES=    file1 file2:DEFAULT file3:group3 \
    file4:group4,group5,group6 file5:grouping \
    file6:group7
MASTER_SITE_SUBDIR=    directory-trial:1 directory-n/:groupn \
        directory-one/:group6,DEFAULT \
        directory
```

前述的例子的结果是下述的对于下载行为的精细控制。 站点的列表按照使用的顺序给出。

- file1 将从
  - MASTER_SITE_OVERRIDE
  - http://site1/directory-trial:1/
  - http://site1/directory-one/
  - http://site1/directory/
  - http://site2/
  - http://site7/
  - MASTER_SITE_BACKUP

  下载。
- file2 将和 file1 以同样的方式下载， 因为它们属于同一组
  - MASTER_SITE_OVERRIDE
  - http://site1/directory-trial:1/
  - http://site1/directory-one/
  - http://site1/directory/
  - http://site2/
  - http://site7/
  - MASTER_SITE_BACKUP
- file3 将从
  - MASTER_SITE_OVERRIDE
  - http://site3/
  - MASTER_SITE_BACKUP

  下载。
- file4 将从
  - MASTER_SITE_OVERRIDE
  - http://site4/
  - http://site5/
  - http://site6/
  - http://site7/
  - http://site8/directory-one/
  - MASTER_SITE_BACKUP

  下载。

- file5 将从
  - MASTER_SITE_OVERRIDE
  - MASTER_SITE_BACKUP

  下载。
- file6 将从
  - MASTER_SITE_OVERRIDE
  - http://site8/
  - MASTER_SITE_BACKUP

  下载。

8. 如何对来自 bsd.sites.mk 的特殊变量， 例如 MASTER_SITE_SOURCEFORGE 进行分组?

   参见 MASTER_SITE_SOURCEFORGE 中 MASTER_SITES:n 的详细用法。

   例 5. MASTER_SITE_SOURCEFORGE 中 MASTER_SITES:n 的详细用法

   ```
   MASTER_SITES=  http://site1/
   ${MASTER_SITE_SOURCEFORGE:S/$/:sourceforge,TEST/}
   DISTFILES=    something.tar.gz:sourceforge
   ```

   something.tar.gz 将从所有 MASTER_SITE_SOURCEFORGE 中的站点下载。

9. 如何与 PATCH* 变量连用?

   前面的例子介绍的都是 MASTER* 变量， 但对于 PATCH* 也是完全一样的， 它们在 简化的 PATCH_SITES 中的 MASTER_SITES:n 用法。 有所介绍。

   例 6. 简化的 PATCH_SITES 中的 MASTER_SITES:n 用法。

   ```
   PATCH_SITES=   http://site1/ http://site2/:test
   PATCHFILES=    patch1:test
   ```

会改变 ports 的哪些行为? 哪些不会?

i. 所有普通的 ports 的行为都会保持不变。 MASTER_SITES:n 功能的代码，
只有在某些元素包含了前述， 特别是 7 中所提及语法的 :n 后缀时， 才会启用。

ii. 不受影响的 port target: checksum、 makesum、 patch、 configure、 build， 等等。 显然， do-fetch、 fetch-list、 master-sites 和 patch-sites 的行为会发生变化。
   - do-fetch: 会按照新的、 带有组后缀的 DISTFILES 和 PATCHFILES 在 MASTER_SITES 和 PATCH_SITES 所匹配的组元素， 以及 MASTER_SITE_SUBDIR 和 PATCH_SITE_SUBDIR 来进行。 请参见 用到逗号分隔符、 多个文件， 多个站点和 不同子目录的 MASTER_SITES:n 详细用法。
   - fetch-list: 和旧式的 fetch-list 类似， 但以同 do-fetch 相似的方式处理组。
   - master-sites 和 patch-sites: (与旧版本不兼容) 仅返回组 DEFAULT 的元素; 事实上，
     它们会执行 master-sites-default 和 patch-sites-default 这两个 target。

     更进一步， 使用 master-sites-all 或 patch-sites-all 这两个 target 之一， 要比直接检查 MASTER_SITES 或 PATCH_SITES 更好。 此外， 未来版本可能不再保证直接检查能够正确工作。

请参见 B 以了解关于这些新 target 的更多技术细节。

iii. port 中的新 target

a. 一系列 master-sites-n 和 patch-sites-n target 可以分别用来列出 MASTER_SITES 和 PATCH_SITES 中的 n 组的内容。 例如， master-sites-DEFAULT 和 patch-sites-DEFAULT 都会返回 DEFAULT 组的内容， 而 master-sites-test 和 patch-sites-test 则返回 test 组的内容， 等等。

b. 新增的 master-sites-all 和 patch-sites-all 这两个 target， 会完成先前 master-sites 和 patch-sites 所做的工作。 它们会返回所有组的元素， 就像这些元素都属于同一组一样， 并且会列出与 MASTER_SITE_BACKUP 或 MASTER_SITE_OVERRIDE 中在 DISTFILES 或 PATCHFILES 中指定的同样多个； 分别对于 master-sites-all 和 patch-sites-all。

5.4.8. DIST_SUBDIR (独立的源码包子目录)

避免让您的 port 使 /usr/ports/distfiles 陷入混乱。 如果您的 port 需要下载很多文件，或者需要下载可能与其它 port 的源文件名冲突的文件 (例如， Makefile)， 则应将 DIST_SUBDIR 设置为 port 的名字 (通常可以用 ${PORTNAME} 或 ${PKGNAMEPREFIX}${PORTNAME})。 这将把 DISTDIR 从默认的 /usr/ports/distfiles 改为 /usr/ports/distfiles/DIST_SUBDIR， 并将与您的 port 有关的文件放到那个目录中。

此外， 它也会在备份文件主服务器 ftp.FreeBSD.org 上查找同一子目录下的文件 (直接在您的 Makefile 中设置 DISTDIR 则不会有这样的效果， 因此您应使用 DIST_SUBDIR。)



这一设置并不影响您在 Makefile 中定义的 MASTER_SITES。

5.4.9. ALWAYS_KEEP_DISTFILES (一直保存源码包)

如果您的 port 采用的是预编译的包， 但却采用了某种要求源代码必须与预编译版本一同提供的授权， 例如 GPL， 则应使用 ALWAYS_KEEP_DISTFILES 来告诉 FreeBSD 联编集群保留一份在 DISTFILES 中文件的副本。 一般来说这些 port 的用户并不需要这些文件， 因此， 只在定义了 PACKAGE_BUILDING 符的时候， 才将源代码包文件加入 DISTFILES 是个好主意。

例 7. 如何使用 ALWAYS_KEEP_DISTFILES。

```
.if defined(PACKAGE_BUILDING)
DISTFILES+=         foo.tar.gz
ALWAYS_KEEP_DISTFILES= yes
.endif
```

当您在 DISTFILES 加入其它文件时， 请务必确保这些文件也出现在了 distinfo 中。 此外，这些额外的文件通常也会展开到 WRKDIR 中， 对于某些 ports， 这可能导致一些不希望的副作用，因而需要进行特别的处理。

5.5. MAINTAINER (监护人)

请在此处写上您的电子邮件地址。 :-)

需要注意一点， MAINTAINER 变量的值只能是一个不包括注释部分的电子邮件地址， 其格式应为 user@hostname.domain。 请不要在此处写任何说明性的文字， 例如您的真实姓名 - 这会给 bsd.port.mk 带来麻烦。

监护人有责任保持 port 随时更新， 并确保其能够正确地运行。 详细的 port 监护人职责说明， 请参见 port 监护人面临的挑战 一节。

对于 port 的修改， 应被发给 port 的监护人进行复审， 且在 commit 之前需要获得其监护人的同意。

假如某一 port 的监护人没有在两周之内 (不包括主要的公共假日) 响应来自用户的更新请求，
则可视为监护人超时， 在这种情况下可以在没有监护人明确同意的情形下进行更新。
如果监护人在多达三个月的时间内没有进行任何响应， 则可以认为该监护人不辞而别，
允许对出现此类问题的 port 进行监护人变更。 尽管如此， 监护人为 Ports 管理团队
<portmgr@FreeBSD.org> 或者 Security Officer 团队 <security-officer@FreeBSD.org> 的 port
不受此限。 对监护人为这些小组的 port 进行未经许可的 commit 是不允许的。

我们保留对监护人所提交修正案进行改动的权力， 以便使其更符合现行的 Ports Collection 规范，
而无需提交补丁的人明确批准。 此外， 大规模的基础性修改， 也可能使 port
在没有得到监护人同意的情形下进行修改。 但这类修改都不应影响 port 本身的功能。

Ports 管理团队 <portmgr@FreeBSD.org> 保留以任何原因收回或绕过任何人监护权的权力， 而 Security Officer 团队 <security-officer@FreeBSD.org> 则保留以安全原因收回或绕过监护权的权力。

5.6. COMMENT (一句话说明)

这一变量用于指定 port 的一句话说明。 请 勿将 package 的名字 (或软件的版本) 放在说明中。
这一说明的第一个字母应大写， 结尾不用句点。 下面是一个例子：

```
COMMENT=    A cat chasing a mouse all over the screen
```

Makefile 中的 COMMENT 变量应该紧接着 MAINTAINER 变量出现。

请务必将 COMMENT 这行限制在不超过 70 个字符之内， 因为这行内容会成为 pkg_info(1) 呈现给用户的 port 的一句话简介。

5.7. 依赖关系

许多 ports 会依赖其它 port。 这是包括 FreeBSD 在内的多数 类-Unix 系统的很方便的功能。 这项功能，
可以避免在每个 port 或预编译包中都带上重复的依赖的代码， 而可以以依赖关系的方式去共享它们。
有七个变量用于帮助您确保所需的文件都存在于用户的机器上。 此外，
也提供了用于支持常见情形的依赖关系变量， 以及对依赖关系行为的更多控制。

5.7.1. LIB_DEPENDS (依赖的函数库/共享库)

这个变量用于指定 port 所依赖的共享库。 其内容是由一系列 lib:dir:target 元组构成的表， 其中 lib
是共享库的名字， 而 dir 则是在找不到时应该从哪里联编和安装， 最后， target
用于指定在那个目录中调用的 target。 例如，

```
LIB_DEPENDS=  jpeg.9:${PORTSDIR}/graphics/jpeg
```

会检测主版本号为 9 的 jpeg 共享库， 如果它不存在， 则会进入到您的 ports 目录中的 graphics/jpeg
子目录， 并联编和安装它。 如果您指定的 target 就是 DEPENDS_TARGET (默认是 install)，
则可以略去不写。



lib 部分是一个正则表达式， 用于在 ldconfig -r 的输出中进行查找。 可以使用类似 intl.[5-7] 和 intl 这样的值。 前一种模式， 即 intl.[5-7]， 能够匹配 intl.5、 intl.6 和 intl.7 中的任意一个。 第二种模式， 即 intl 则可以匹配任意版本的 intl 库。

依赖关系会被检测两次， 一次是在 extract target 中， 而另一次则是在 install target。 另外，
依赖关系的名字会放到 package 中， 以便让 pkg_add(1)
能够自动地在用户系统上安装所需的未安装的其它 package。

5.7.2. RUN_DEPENDS (依赖的运行环境)

这个变量可以用来指定 port 在运行时所需要的可执行文件， 以及资源文件。 它是一系列 path:dir:target 元组的列表， 这里， path 时所需的可执行， 或者资源文件的名字， dir 是在无法找到这些文件或目录时，去什么地方完成联编和安装以便获得这些文件； 而 target 则用来指定在这个目录中所调用的 target 的名字。 假如 path 以斜线 (/) 开始， 则会当作普通文件， 使用 test -e 来测试； 反之，则系统会假定这是一个可执行文件， 并且用 which -s 来检测程序是否存在于搜索路径中。

例如，

```
RUN_DEPENDS=  ${LOCALBASE}/etc/innd:${PORTSDIR}/news/inn \
      xmlcatmgr:${PORTSDIR}/textproc/xmlcatmgr
```

将检查文件， 或者目录 /usr/local/etc/innd 是否存在， 如果找不到， 则将从 port 目录的 news/inn 子目录加以安装。 系统也会检查是否能够在搜索路径中找到名为 xmlcatmgr 的文件， 如果找不到的话，则会进入 ports 目录中的 textproc/xmlcatmgr 子目录， 并进行联编和安装的操作。



这种情况下， innd 实际上是一个可执行文件； 如果可执行文件不会出现在搜索路径中，您就需要指定完整路径了。



ports 联编集群上官方的搜索 PATH 是

```
/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:/usr/X11R6/bin
```

这个依赖关系会在 install target 的过程中进行检查。 此外， 依赖关系的名字会被放到 package 中， 以便 pkg_add(1) 能够在用户的系统中尚未安装相关软件时自动地安装那些 package。 如果您希望指定一个的 target 和默认的 DEPENDS_TARGET 相同， 则可以略去不写。

一种比较常见的情形是 RUN_DEPENDS 和 BUILD_DEPENDS 完全一样，
这种情况在移植的软件是采用脚本语言书写， 或联编环境与运行环境需求相同时尤其普遍。
这种情况可以用下面简单明了的方式直接将其中一个变量赋值给另一个变量：

```
RUN_DEPENDS= ${BUILD_DEPENDS}
```

不过， 这种赋值有可能会令运行环境被某些没有在 port 原本的 BUILD_DEPENDS
明确定义的依赖关系污染。 导致这种情况的原因是 make(1) 计算变量赋值时默认采用的是延后计算 (lazy evaluation)。 例如， 如果在 Makefile 中使用了 USE_* 变量， 这些变量就会由 ports/Mk/bsd.*.mk 处理， 并填写与之对应的联编依赖关系。 例如， USE_GMAKE=yes 会把 devel/gmake 加入到 BUILD_DEPENDS。 如果希望避免这些附加的依赖关系污染 RUN_DEPENDS，
在使用赋值的时候就需要小心考虑这类扩展的情况， 例如， 可以在赋值展开之前复制变量的值：

```
RUN_DEPENDS:=  ${BUILD_DEPENDS}
```

5.7.3. BUILD_DEPENDS (依赖的联编环境)

此变量用于指定用来联编 port 的可执行文件或资源文件。 与 RUN_DEPENDS 类似， 它是一个 path:dir:target 元组的列表。 例如，

```
BUILD_DEPENDS=
     unzip:${PORTSDIR}/archivers/unzip
```

将检测名为 unzip 的可执行文件是否存在， 如果不存在， 则会进入到您的 ports 目录中的 archivers/unzip 并完成联编和安装工作。



这里的 "build" 表示从解压缩到编译的全部过程。 依赖关系是在 extract target 的过程中检测的。 假如您要指定的 target 和 DEPENDS_TARGET 相同， 则可以略去不写。

5.7.4. FETCH_DEPENDS (依赖的下载环境)

这一变量用于指定 port 在下载时所需的可执行文件或资源文件。 和前两个类似， 它是一组 path:dir:target 元组。 例如，

```
FETCH_DEPENDS=
    ncftp2:${PORTSDIR}/net/ncftp2
```

将检测名为 ncftp2 的可执行文件是否存在， 如果找不到， 则将进入到您 ports 目录中的 net/ncftp2 子目录并加以联编和安装。

这个依赖关系是在 fetch target 过程中检查的。 如果与 DEPENDS_TARGET 相同， 则可以省略 target 部分。

5.7.5. EXTRACT_DEPENDS (依赖的解压缩环境)

此变量用于指定 port 在解压缩时所需的可执行文件或其它资源文件。 和前一个变量类似， 它是一系列 path:dir:target 元组的列表。 例如，

```
EXTRACT_DEPENDS=
    unzip:${PORTSDIR}/archivers/unzip
```

将检查名为 unzip 的可执行文件是否存在， 如果不存在， 则会进入到您的 ports 目录中的 archivers/unzip 子目录， 予以联编和安装。

这个依赖关系是在 extract target 的过程中检查的。 如果与 DEPENDS_TARGET 相同， 则可以略去 target 部分。



只有在其它方式都不可用 (默认是 gzip) 而且无法通过 USE_* 所介绍的 USE_ZIP 或 USE_BZIP2 都不能达到需要时， 才应使用这个变量。

5.7.6. PATCH_DEPENDS (依赖的打补丁环境)

这个变量用于指定 port 在进行 patch 操作时所需的可执行文件或其它资源文件。 和前一个变量类似， 它是一组 path:dir:target 元组的表。 例如，

```
PATCH_DEPENDS=
    ${NONEXISTENT}:${PORTSDIR}/java/jfc:extract
```

表示进入到您的 ports 目录中的 java/jfc 子目录， 并将其解压缩。

这个依赖关系是在 patch target 的过程中检查的。 target 部分如果和 DEPENDS_TARGET 相同， 就可略去不写。

5.7.7. USE_*

提供了一系列变量，用以封装大量 port 都用到的依赖关系。虽然使用这些变量是可选的，但它们能显著减少 port 的 Makefile 复杂性。这些变量的共同特征在于，它们的名字都是 USE_* 这样的形式。这些变量的使用，应严格限制于 port 的 Makefile 以及 ports/Mk/bsd.*.mk，而绝不应用于表达用户能够设置的选项 - 这种情况下应采用 WITH* 和 WITHOUT* 这样的变量。



在任何情况下，都不应在 /etc/make.conf 中配置任何 USE_*。例如，设置

```
USE_GCC=3.4
```

将导致每个 port 都依赖 gcc34，甚至包括 gcc34 本身！

表 2. 常用的 USE_* 变量

变量	含义
USE_BZIP2	此 port 的源码包是使用 bzip2 压缩的。
USE_ZIP	此 port 的源码包是用 zip 压缩的。
USE_BISON	此 port 在联编时使用 bison。
USE_CDRTOOLS	此 port 需要使用 cdrecord，根据用户的喜好，可能是 sysutils/cdrtools 或 sysutils/cdrtools-cjk。
USE_GCC	此 port 需要使用某一特定版本的 gcc 才能完成编译。可以使用类似 3.4 这样的值来精确指定版本。 如果希望使用不低于某一版本的编译器，则可以用 3.4+ 这样的形式。如果与所希望的版本吻合，则将使用基本系统中所提供的 gcc，反之，系统会从 ports 中安装所希望版本的 gcc，并调整 CC 以及 CXX 变量的设置。

与 gmake 和 configure 脚本有关的变量在 联编机制 中进行了介绍，而 autoconf、automake 以及 libtool 的介绍则可以在 利用 GNU autotools 找到。使用 perl 介绍了与 Perl 有关的的变量。使用 X11 中列出了关于 X11 的变量。关于 GNOME 的变量在 使用 GNOME，而关于 KDE 的则在 使用 KDE。使用 Java 讲述了和 Java 有关的变量，而 Web 应用，Apache 和 PHP 则包含了关于 Apache、PHP 以及 PEAR 的介绍性信息。关于 Python，在 使用 Python 进行了讨论，而关于 Ruby 的介绍，则可以在 使用 Ruby 中找到。使用 SDL 提供了用于 SDL 应用程序的变量介绍，最后，使用 Xfce 包含了关于 Xfce 的信息。

5.7.8. 在依赖关系中指定最低版本

在依赖某个其他 port 时，可以采用下面的句法，通过除 LIB_DEPENDS 之外的 *_DEPENDS 变量来指定最低版本：

```
p5-Spiffy>=0.26:${PORTSDIR}/devel/p5-Spiffy
```

第一个字段指明了所依赖 package 的名字，用以与 package 数据库中的某项匹配，然后是比较算符，以及 package 的版本号。前面的例子中，如果系统中安装了 p5-Spiffy-0.26 则认为满足了依赖条件。

5.7.9. 关于依赖关系的补充说明

如前面所提到的那样，在需要某一依赖的 port 时，将调用 DEPENDS_TARGET 所指定的 target。这一变量的默认值是 install。这不是一个用户变量，它不应在 port 的 Makefile 中予以定义。如果您的 port 需要使用特殊的 target 来处理依赖关系，应使用 *_DEPENDS 的 :target 部分，而不是重定义 DEPENDS_TARGET 来完成。

当您输入 make clean 时， 其依赖的 port 也会自动进行清理。 如果您不希望如此， 应定义环境变量 NOCLEANDEPENDS。 如果 port 依赖一些重新联编需要花费很长时间的 port 时， 例如 KDE， GNOME 或 Mozilla 时， 这一方法会非常有用。

要无条件地依赖某个 port， 可以使用 ${NONEXISTENT} 作为 BUILD_DEPENDS 或 RUN_DEPENDS 的第一部分。 只有在您需要使用其它 port 提供的源代码时才应这样做。
通常也可以通过这样指定来缩短编译所需的时间。 例如

```
BUILD_DEPENDS=  ${NONEXISTENT}:${PORTSDIR}/graphics/jpeg:extract
```

表示依赖 jpeg port 并将其解压缩。

5.7.10. 循环的依赖关系是致命的



不要在 ports tree 中引入任何循环依赖关系!

ports 联编技术不能够容忍循环依赖关系。 如果您引入了这样的关系， 就一定会有人安装的 FreeBSD 会因此而损坏， 而且这种现象会越来越多。 这些情形很难检测； 如果有疑虑， 在进行这样的修改之前， 务必执行： cd /usr/ports; make index。 这个过程在旧的机器上会很慢， 但能够让大量的用户 - 也包括您自己 - 拯救于由这种问题所造成的困惑之中。

5.8. MASTERDIR (主 port 所在的目录)

如果 port 需要依某些变量的设置 (举例来说， 分辨率或纸型) 来联编略有不同的预编译包， 则可以为每一个这样的包建立不同的目录， 这样可以让用户更容易地看到他们想要安装的版本， 但又能在这些 port 之间共用尽可能多的文件。 一般情况下， 如果运用得当， 除主目录之外都只需要很短的 Makefile。 这些 Makefile 中， 可以用 MASTERDIR 来指定其它文件所在的目录。 另外， 还应使用一个变量作为 PKGNAMESUFFIX 的一部分， 以便为不同的包给出不同的命名。

用例子来阐述这些会更为明晰。 以下是 japanese/xdvi300/Makefile 的部分代码：

```
PORTNAME=     xdvi
PORTVERSION=  17
PKGNAMEPREFIX= ja-
PKGNAMESUFFIX= ${RESOLUTION}
:
# default
RESOLUTION?=  300
.if ${RESOLUTION} != 118 && ${RESOLUTION} != 240 && \
    ${RESOLUTION} != 300 && ${RESOLUTION} != 400
    @${ECHO_MSG} "Error: invalid value for RESOLUTION: \"${RESOLUTION}\""
    @${ECHO_MSG} "Possible values are: 118, 240, 300 (default) and 400."
    @${FALSE}
.endif
```

japanese/xdvi300 也提供了全部常规的补丁， 以及打包用到的文件等等内容。 如果您在那里输入 make， 它将使用默认的分辨率值 (300) 并正常地联编 port。

对于其它分辨率而言， 以下是 完整的 xdvi118/Makefile：

```
RESOLUTION=    118
MASTERDIR=     ${.CURDIR}/../xdvi300

.include "${MASTERDIR}/Makefile"
```

(xdvi240/Makefile 和 xdvi400/Makefile 是相似的)。 MASTERDIR 定义会告诉 bsd.port.mk 常规的目录，例如 FILESDIR 以及 SCRIPTDIR 应在 xdvi300 中查找。 RESOLUTION=118 这行将覆盖在 xdvi300/Makefile 中所作的 RESOLUTION=300 设置， 从而 port 将以分辨率为 118 的设置来联编。

5.9. 联机手册

MAN[1-9LN] 这些变量， 会自动地将联机手册加到 pkg-plist (这也意味着 不能 在 pkg-plist 中列出联机手册 - 参见 PLIST 的生成 来了解更多细节)。 此外， 这也会让安装阶段自动地根据在 /etc/make.conf 中所作的 NO_MANCOMPRESS 设置来自动对联机手册文件执行压缩或解压缩操作。

如果 port 尝试通过使用符号连接或硬连接将联机手册安装为多个名字， 就必须使用 MLINKS 变量来予以明示。 由 port 创建的连接， 将由 bsd.port.mk 删除和重建， 以确认它们指向了正确的文件。 任何在 MLINKS 中列出的文件都不应在 pkg-plist 中再出现。

要指定是否在安装时对联机手册进行压缩， 可以使用 MANCOMPRESSED 变量。 这一变量可以取三种值， yes、 no 和 maybe 之一。 yes 表示联机手册已经以压缩的形式安装， no 表示还没有， 而 maybe 则表示所安装的软件会尊重 NO_MANCOMPRESS 的设置值， 因此 bsd.port.mk 不需要特别做什么事情。

如果设置了 USE_IMAKE 而未定义 NO_INSTALL_MANPAGES， MANCOMPRESSED 会自动设为 yes， 反之则是 no。 除非默认值不合适， 否则就不需要在 port 中明确地加以改变。

如果 port 将联机手册放到了 PREFIX 之外的其它目录， 则应使用 MANPREFIX 来加以设置。 此外， 如果只有某些部分的联机手册会安装到不标准的位置， 例如某些 perl 模块的 port， 还可以使用 MAN_sect_PREFIX (此处 sect 是 1-9、 L 或 N 之一) 来指定。

如果您的联机手册需要装入专用于某一语言专用的子目录， 需要将 MANLANG 设为那种语言的名字。 此变量的默认值是 "" (也就是只有英语)。

下面是一个综合的例子。

```
MAN1=       foo.1
MAN3=       bar.3
MAN4=       baz.4
MLINKS=     foo.1 alt-name.8
MANLANG=    "" ja
MAN3PREFIX=  ${PREFIX}/shared/foobar
MANCOMPRESSED= yes
```

这表示 port 会安装六个文件；

```
${MANPREFIX}/man/man1/foo.1.gz
${MANPREFIX}/man/ja/man1/foo.1.gz
${PREFIX}/shared/foobar/man/man3/bar.3.gz
${PREFIX}/shared/foobar/man/ja/man3/bar.3.gz
${MANPREFIX}/man/man4/baz.4.gz
```

```
${MANPREFIX}/man/ja/man4/baz.4.gz
```

此外， ${MANPREFIX}/man/man8/alt-name.8.gz 可能会通过您的 port 安装， 也可能不会。 无论如何，都会创建一个符号连接， 把 foo(1) 和 alt-name(8) 联机手册连起来。

假如只有部分联机手册是翻译过的， 则可以使用一些根据 MANLANG 内容动态生成的变量：

```
MANLANG=    "" de ja
MAN1=       foo.1
MAN1_EN=    bar.1
MAN3_DE=    baz.3
```

这相当于下列文件：

```
${MANPREFIX}/man/man1/foo.1.gz
${MANPREFIX}/man/de/man1/foo.1.gz
${MANPREFIX}/man/ja/man1/foo.1.gz
${MANPREFIX}/man/man1/bar.1.gz
${MANPREFIX}/man/de/man3/baz.3.gz
```

5.10. Info 文件

如果软件包需要安装 GNU info 文件， 则需要在 INFO 变量中一一列出 (不需要指定 .info 后缀)。系统假定这些文件均会安装到 PREFIX/INFO_PATH 目录中。 如果软件包有需要， 也可以通过修改 INFO_PATH 来指定不同的位置。 不过， 并不推荐这样做。 所有列出的项目均是相对于 PREFIX/INFO_PATH 的文件路径。 例如， lang/gcc34 表示将 info 文件安装到 PREFIX/INFO_PATH/gcc34， 因此 INFO 应写成类似这样：

```
INFO= gcc34/cpp gcc34/cppinternals gcc34/g77 ...
```

这样安装/卸载代码就会自动地在注册包之前将它们加入到临时的 pkg-plist 中了。

5.11. Makefile 选项

某些大型应用程序可以在联编时使用一系列配置选项，用以在系统中已经安装了某些库或应用程序时增加一些功能。 例如， 选择某种自然 (人类的) 语言， GUI 或命令行界面， 由于并不是所有的用户都希望使用这些库或者应用程序， port 系统提供了一组方便的机制， 来让 port 的作者控制联编时的配置。 支持这些特性可以让用户体验更好，并达到事半功倍的效果。

5.11.1. 开关 (Knobs)

WITH_* 和 WITHOUT_*

这些变量是为系统管理员准备的。 许多这样的变量被标准化并置于 ports/KNOBS 文件。

在创建一个 port 的时候，不要使用某个应用程序专有的 knob 名称，比如对于 Avahi 这个 port，应该用 WITHOUT_MDNS 而不是 WITHOUT_AVAHI_MDNS。



您不应假定每一个 WITH_* 都会有对应的 WITHOUT_* 变量， 反之亦然。 一般而言，

会使用默认值。



除非另有说明，这些变量都是测试是否定义，而不是它们设置了 YES 或 NO。

表 3. 常见的 WITH_* 和 WITHOUT_* 变量

变量	意义
WITHOUT_NLS	表示不需要国际化支持， 这可以节省编译所消耗的时间。 默认情况下， 会启用国际化支持。
WITH_OPENSSL_BASE	使用基本系统中的 OpenSSL 版本。
WITH_OPENSSL_PORT	从 security/openssl 安装 OpenSSL，即使基本系统中的版本是最新的。
WITHOUT_X11	如果 port 能够在是否包含 X 支持的情况下分别联编，则一般情况应该默认以包含 X 支持的配置来联编。 如果定义了这一变量， 则应联编不包含 X 支持的版本。

开关 (knob) 的命名

我们建议 port 的开发人员使用相似的开关，以便最终用户使用，并减少开关名称的总数。
最为常用的开关名字可以在 KNOBS 文件中找到。

开关的名字应反映其功能。 如果 port 的 PORTNAME 包括 lib- 前缀，则开关名中应删去 lib- 前缀。

5.11.2. OPTIONS (菜单式可选项)

背景

OPTIONS 将为正在安装 port 的用户提供一个包含可用选项的对话框，并将用户的选择保存到
/var/db/ports/portname/options 中。 下次重新联编 port 时，这些选项将被再次使用。 这样一来，
就不需要劳神去记忆您之前联编 port 时的那几十个 WITH_* 和 WITHOUT_*选项了！

当用户运行 make config (或首次运行 make build) 时，框架会首先检查
/var/db/ports/portname/options。 如果这个文件不存在，则它会使用 OPTIONS
的值来生成一个可以启用或禁用各个选项的对话框。 随后，用户的选择将保存到 options 文件中，
并被用于联编 port。

如果新版本的 port 新增了 OPTIONS，则系统会再次给出对话框，并根据先前的 OPTIONS
配置预设先前存在的配置。

使用 make showconfig 可以查看保存的配置。 此外， make rmconfig 可以删除已经保存的配置。

语法

OPTIONS 变量的语法是：

```
OPTIONS=  OPTION "说明性文字" 默认值 ...
```

默认值必须是 ON 和 OFF 之一。 这种三元组可以使用多次。

定义 OPTIONS 变量的值，必须在引入 bsd.port.options.mk 之前进行。 而 WITH_* 和 WITHOUT_*
变量则只能在引入了 bsd.port.options.mk 之后才可以进行检测。 使用 bsd.port.pre.mk
也可以达到同样的目的，在系统开始提供 bsd.port.options.mk 之前的许多 port 都在使用这种用法。
不过，请注意 bsd.port.pre.mk 会要求某些变量已经进行过定义，如 USE_* 等。

例 8. 简单的 OPTIONS 用法

```
OPTIONS=    FOO "启用 foo 选项" On \
            BAR "支持 bar 功能" Off

.include <bsd.port.options.mk>

.if defined(WITHOUT_FOO)
CONFIGURE_ARGS+=   --without-foo
.else
CONFIGURE_ARGS+=   --with-foo
.endif

.if defined(WITH_BAR)
RUN_DEPENDS+=  bar:${PORTSDIR}/bar/bar
.endif

.include <bsd.port.mk>
```

例 9. Old style use of OPTIONS

```
OPTIONS=    FOO "Enable option foo" On

.include <bsd.port.pre.mk>

.if defined(WITHOUT_FOO)
CONFIGURE_ARGS+=   --without-foo
.else
CONFIGURE_ARGS+=   --with-foo
.endif

.include <bsd.port.post.mk>
```

5.11.3. 自动激活的特性

在使用 GNU configure 脚本时， 一定要小心有些特性会由其自动检测而激活。 您应通过明确地指定相应的 --without-xxx 或 --disable-xxx 参数到 CONFIGURE_ARGS 来禁用不希望的特性。

例 10. 处理选项时的错误做法

```
.if defined(WITH_FOO)
LIB_DEPENDS+=     foo.0:${PORTSDIR}/devel/foo
```

```
CONFIGURE_ARGS+=   --enable-foo
.endif
```

在前面的例子中， 假设系统中已经安装了 libfoo 库。 用户可能并不希望应用程序使用 libfoo， 因此他在 make config 对话框中关掉了这个选项。 但是， 应用程序的 configure 脚本检测到了系统中存在这个库， 并将其加入到了最终可执行文件支持的功能中。 现在， 如果用户决定从系统中卸载 libfoo 时， ports 系统就无法保护这个应用程序免遭破坏了 (因为没有记录 libfoo 的依赖关系)。

例 11. 处理选项时的正确做法

```
.if defined(WITH_FOO)
LIB_DEPENDS+=     foo.0:${PORTSDIR}/devel/foo
CONFIGURE_ARGS+=   --enable-foo
.else
CONFIGURE_ARGS+=   --disable-foo
.endif
```

在第二个例子中， libfoo 库被明确禁用。 即使系统中已经安装了这个库， configure 脚本也不会启用相应的功能了。

5.12. 指定工作临时目录

每个 port 都会被解压缩到一个工作临时目录中， 这个目录必须是可写的。 ports 系统默认情况下会将 DISTFILES 解压缩到一个叫做 ${DISTNAME} 的目录中。 换言之， 如果设了：

```
PORTNAME=    foo
PORTVERSION=  1.0
```

则 port 的源码包文件的顶级目录将是 foo-1.0。

如果这不是所希望的情形， 您可以修改一系列变量的设置。

5.12.1. WRKSRC (开始联编操作的目录名)

这个变量给出了在应用程序的源代码包解压缩之后所生成的目录的名字。
如果我们之前的例子解压缩生成一个叫做 foo (而不是 foo-1.0) 的目录， 您应：

```
WRKSRC=    ${WRKDIR}/foo
```

或者， 也可能是

```
WRKSRC=    ${WRKDIR}/${PORTNAME}
```

5.12.2. NO_WRKSUBDIR (不需要临时的联编目录)

如果 port 完全不需要写入到某个子目录中， 您应设置 NO_WRKSUBDIR 以明示这一点。

```
NO_WRKSUBDIR= yes
```

5.13. 处理冲突

针对不同的 package 或 port 之间的冲突情形， 系统提供了不同的变量来协助开发人员进行表达: CONFLICTS、 CONFLICTS_INSTALL 和 CONFLICTS_BUILD。



这些用于描述冲突的变量会自动地设置 IGNORE， 后者的完整介绍， 可以在 使用 BROKEN、 FORBIDDEN 或 IGNORE 阻止用户安装 port 找到。

在删去相互冲突的 port 时， 建议将 CONFLICTS 保留几个月，
以便让那些不经常更新系统的用户能够看到。

5.13.1. CONFLICTS_INSTALL

如果您的软件包不能与某些软件包同时安装 (例如由于安装同样的文件到相同的位置、 运行时不兼容等等)，
则应把其它软件包的名字列在 CONFLICTS_INSTALL 变量中。 此处可以使用 shell 通配符， 如 * 和 ?。
列出其它软件包的名字时需要遵循它们在 /var/db/pkg 中出现的样子。 请确保 CONFLICTS_INSTALL
不会匹配到您正制作的这个预编译包的名字， 否则， 使用 FORCE_PKG_REGISTER
来强制安装就没有办法进行了。 对于 CONFLICTS_INSTALL 的检查是在联编过程之后、
安装开始之前进行的。

5.13.2. CONFLICTS_BUILD

如果您的软件包在系统中存在某些其它软件包时不能完成联编， 则应把其它软件包的名字列在
CONFLICTS_BUILD 变量中。 此处可以使用 shell 通配符， 如 * 和 ?。
列出其它软件包的名字时需要遵循它们在 /var/db/pkg 中出现的样子。 对于 CONFLICTS_BUILD
的检查是在联编过程开始之前进行的。 联编时的冲突不会在编译好的包中予以记录。

5.13.3. CONFLICTS

如果您的 port 在某些其它 port 已经存在的情况下既不能联编， 也不能安装， 则应把其它软件包的名字列在
CONFLICTS 变量中。 此处可以使用 shell 通配符， 如 * 和 ?。 列出其它软件包的名字时需要遵循它们在
/var/db/pkg 中出现的样子。 请确保 CONFLICTS 不会匹配到您正制作的这个预编译包的名字， 否则，
使用 FORCE_PKG_REGISTER 来强制安装就没有办法进行了。 对于 CONFLICTS
的检查是在联编过程之后、 安装开始之前进行的。

5.14. 安装文件

5.14.1. INSTALL_* 宏

一定要使用由 bsd.port.mk 提供的宏， 以确保在您自己的 *-install target
中能够以正确的属主和权限模式安装文件。

- INSTALL_PROGRAM 是安装可执行二进制文件的命令。
- INSTALL_SCRIPT 是安装可执行脚本文件的命令。
- INSTALL_LIB 是安装动态连接库的命令。
- INSTALL_KLD 是用于安装可加载式内核模块的命令。 在某些平台上， 当对内核模块进行 strip
  之后会导致一些问题， 因此您应使用这个宏而不是 INSTALL_PROGRAM 来安装内核模块。
- INSTALL_DATA 是安装可共享数据的命令。
- INSTALL_MAN 是安装联机手册和其他文档的命令 (注意它并不会执行压缩操作)。

这些宏展开后基本上都是包含适当参数的 install 命令。

5.14.2. 对可执行文件和动态连接库做脱模 (strip) 操作

除非不得不进行， 否则不要手工对可执行文件作脱模操作。 所有文件在安装时都应脱模， 但 INSTALL_PROGRAM 宏会在安装的同时对其进行脱模 (参见下一节的内容)。 INSTALL_LIB 宏

如果您需要对某一文件进行脱模， 但不希望使用 INSTALL_PROGRAM 及 INSTALL_LIB 宏， 则应使用 ${STRIP_CMD} 来处理程序。 一般而言这应该在 post-install target 中进行。 例如:

```
post-install:
  ${STRIP_CMD} ${PREFIX}/bin/xdl
```

可以使用 file(1) 命令来检查所安装的可执行文件是否进行过脱模。 如果它没有给出 not stripped 的提示， 则表示已经做过脱模了。 另外， strip(1) 不会对已经脱模过的文件重新脱模， 它会直接退出的。

5.14.3. 安装一个目录下的全部文件

有时， 会有需要安装大量的文件， 并保持其层次结构， 例如， 将整个目录结构从 WRKSRC 复制到 PREFIX 的目标目录。

针对这种情况， 系统提供了两个宏。 使用这些宏， 而不是直接使用 cp 的优势是它们能够确保目标文件的属主和权限正确。 第一个宏， COPYTREE_BIN 将所有安装的文件视为可执行文件， 因而适合安装文件到 PREFIX/bin。 第二个宏， COPYTREE_SHARE， 则不会设置可执行权限， 因此适合于将文件安装到 PREFIX/share 下。

```
post-install:
  ${MKDIR} ${EXAMPLESDIR}
  (cd ${WRKSRC}/examples/ && ${COPYTREE_SHARE} \* ${EXAMPLESDIR})
```

这个例子将原作者提供的整个 examples 目录复制到您 port 指定的安装示范文件的位置。

```
post-install:
  ${MKDIR} ${DATADIR}/summer
  (cd ${WRKSRC}/temperatures/ && ${COPYTREE_SHARE} "June July August"
${DATADIR}/summer/)
```

这个例子将把夏季的三个月的数据， 复制到 DATADIR 中的 summer 子目录。

经由设置 COPYTREE_* 宏的第三个参数， 您还可以为 find 指定额外的参数。 例如， 如果希望安装除了 Makefile 之外的其他所有文件， 可以使用下述命令。

```
post-install:
  ${MKDIR} ${EXAMPLESDIR}
  (cd ${WRKSRC}/examples/ && \
    ${COPYTREE_SHARE} \* ${EXAMPLESDIR} "! -name Makefile")
```

需要注意的是， 这些宏并不能自动将所安装的文件加到 pkg-plist 中， 您还是需要自行列出它们。

5.14.4. 安装附加的文档

如果您的软件包含了标准的联机手册和 info 手册以外的文档， 而且您认为它们对用户会有用， 请把这些文档安装到 PREFIX/shared/doc 下。 和前面类似， 这也可以在 post-install target 中完成。

为您的 port 建立一个新的目录。 这个目录的名字应该反映它是属于哪个 port 的。 通常建议使用 PORTNAME。 不过， 如果您认为不同版本的 port 可能会同时安装， 也可以用完整的 PKGNAME。

另外， 应该让是否安装取决于变量 NOPORTDOCS 的设置， 这样用户就能够在 /etc/make.conf 中禁止安装它。 例如：

```
post-install:
.if !defined(NOPORTDOCS)
  ${MKDIR} ${DOCSDIR}
  ${INSTALL_MAN} ${WRKSRC}/docs/xvdocs.ps ${DOCSDIR}
.endif
```

这里是一些便于使用的变量， 以及它们在 Makefile 中默认的展开方式：

- DATADIR 会展开成 PREFIX/shared/PORTNAME。
- DATADIR_REL 会展开成 share/PORTNAME。
- DOCSDIR 会展开成 PREFIX/shared/doc/PORTNAME。
- DOCSDIR_REL 会展开成 share/doc/PORTNAME。
- EXAMPLESDIR 会展开成 PREFIX/shared/examples/PORTNAME。
- EXAMPLESDIR_REL 会展开成 share/examples/PORTNAME。



NOPORTDOCS 只控制将要安装到 DOCSDIR 的那些文档， 而不应影响标准的联机手册以及 info 手册的安装。 安装到 DATADIR 和 EXAMPLESDIR 的文件则相应地受 NOPORTDATA 和 NOPORTEXAMPLES 控制。

这些变量也会被导出到 PLIST_SUB 中。 只要可能， 它们的值就将在那里以相对于 PREFIX 的路径形式出现。 也就是说， share/doc/PORTNAME 在装箱单中默认情况下会替换掉 %%DOCSDIR%%， 等等。 (更多的 pkg-plist 代换可以在 这里 找到。)

所有非无条件安装的文档文件和目录， 都应在 pkg-plist 出现， 并且使用 %%PORTDOCS%% 前缀， 例如：

```
%%PORTDOCS%%%%DOCSDIR%%/AUTHORS
%%PORTDOCS%%%%DOCSDIR%%/CONTACT
%%PORTDOCS%%@dirrm %%DOCSDIR%%
```

如果不希望在 pkg-plist 中逐个列举文档文件， port 也可以将 PORTDOCS 设置为一组文件及其 shell glob 模式， 通过这种方式来加入到最终的装箱单中。 这些名字应是相对于 DOCSDIR 的。 因此， 使用了 PORTDOCS， 并将文档安装到非标准位置的 port， 应相应地设置 DOCSDIR。 如果有在 PORTDOCS 中列出目录， 或者这一变量中的 glob 模式匹配到了目录， 则整个子树中的文件和目录， 都将被注册到最终的装箱单中。 如果定义了 NOPORTDOCS， 则 PORTDOCS 中定义的文件和目录将不被安装或加入装箱单。 是否安装文档到前面所说的 PORTDOCS 仍取决于 port 本身。 下面是一个典型的使用 PORTDOCS 的例子：

```
PORTDOCS=     README.* ChangeLog docs/*
```



与 PORTDOCS 类似， 对应于 DATADIR 和 EXAMPLESDIR 的变量分别是 PORTDATA 和 PORTEXAMPLES。

您也可以使用 pkg-message 这个文件， 来在安装时显示一些信息。 参见 关于使用 pkg-message 的这一节 以了解进一步的详情。 需要说明的是， 并不需要把 pkg-message

加到 pkg-plist 中。

5.14.5. 子目录

尽可能让 port 将它创建的文件， 放置到 PREFIX 中正确的位置。 一些 port 会把各式各样的东西混在一起，
并放到一个同名的目录中， 这是不对的。 另外， 许多 port 会把除了可执行文件、
头文件和联机手册之外的所有文件， 全都一股脑地放到 lib 中， 这在和 BSD 配合使用时会有问题。
多数文件， 应被放到下列位置之一： etc (安装/配置文件)、 libexec (由系统内部调用的可执行文件)、 sbin
(为超级用户/管理员提供的可执行文件)、 info (用于 info 浏览器的文档) 或 share (平台无关的其它文件)。
请参见 hier(7) 以了解进一步的详情； 针对 /usr 的那些规则， 同样也适用于 /usr/local。
例外情况是那些需要和 USENET "news" 打交道的 port， 它们可以选择采用 PREFIX/news
作为文件的目的地。

Chapter 6. 特殊情况

有一些您在创建port时的特殊情况，我们在这里提一下。

6.1. 共享库

如果您的port安装了一个或多个共享库,那么请定义一个 USE_LDCONFIG make 变量， 在post-install标记把它注册进共享库 缓冲时会调用bsd.port.mk去运行 ${LDCONFIG} -m来指向新库的安装目录。(通常是 PREFIX/lib) 同样，您也可以适当的在您的 pke-plist文件 中定义一组@exec /sbin/ldconfig -m和@unexec /sbin/ldconfig -R， 那么用户可以在安装后马上
就能使用，并且在卸载软件包后系统也不会认为这些共享库仍然存在。

```
USE_LDCONFIG= yes
```

如果您需要把共享库安装在缺省的位置之外， 可以通过定义 make 变量 USE_LDCONFIG
来改变默认的安装路径， 它包含安装共享库的目录列表 例如： 如果您的共享库安装到 PREFIX/lib/foo 和 PREFIX/lib/bar directories目录，您可以在您的 Makefile中这样设置:

```
USE_LDCONFIG= ${PREFIX}/lib/foo ${PREFIX}/lib/bar
```

请务必仔细检查， 通常这是完全不必要的， 或者可以通过 -rpath 或在连接时设置 LD_RUN_PATH 来避免(参见 lang/moscow_ml 给出的例子)， 或者用一个 shell 封装程序来在执行可执行文件之前设置 LD_LIBRARY_PATH， 类似 www/seamonkey 那样。

当在 64-位系统上安装 32-位 的函数库时， 请使用 USE_LDCONFIG32。

尽量将共享库版本号保持为 libfoo.so.0 这样的格式。 我们的运行环境连接器只会检查主 (第一个)
版本数字。

如果在更新 port 时升级了其库的主版本号， 则其它所有连接了受影响的库的 port 的 PORTREVISION
都应递增， 以强制它们采用新版本的库重新编译。

6.2. Ports 的发行限制

众多协议，并且其中的一些致力于 限制怎样的应用程序能被打包， 是否能用于销售赢利等等。



做为一名porter您有义务去阅读软件的协议 并且确保FreeBSD 项目不必为通过FTP/HTTP或CD-ROM重新发布源码或编译的二进制而解释 什么。如果有任何疑问， 请联系 FreeBSD ports 邮件列表。

处于这种情况，就可以设置以下描述 的变量。

6.2.1. NO_PACKAGE (禁止编译结果打包)

这个变量表示我们可能不能生成这个应用 程序的二进制文件。例如，他的协议不允许
二进制文件的再次发行，或者他可能禁止从 补丁过的源代码打包的发行。

不管怎么样，port的 DISTFILES 可以 随意的镜像到FTP/HTTP。除非NO_CDROM
变量也被设置，软件包也可以发行在一张CD-ROM （或类似的媒介上）。

NO_PACKAGE也能用在当二进制包 不是非常有用，并且这个应用软件经常要
从源代码编译。例如：当这个应用软件在 编译的时候要在配置信息中指定特定的硬件
代码时，可以设置NO_PACKAGE。

NO_PACKAGE应该设置成字符串 来描述为什么这个软件 不能打包。

6.2.2. NO_CDROM (禁止以 CDROM 发行预编译包)

这个变量仅仅指出虽然我们允许 生成二进制包，但也许我们既不能把这个 软件包也不能把port的DISTFILES 放在光盘（或类似的媒介）上销售。但不管怎么样， 二进制包和port的DISTFILES 可以从FTP/HTTP上获得。

如果这个变量和 NO_PACKAGE一起被设置， 那么这个port的DISTFIELS 将只能从FTP/HTTP上获得。

NO_CDROM 应该被设置成一个字符串 来描述为什么这个port不能重新发布在CD-ROM上。
例如：如果这个port的协议仅仅是用于"非商业活动" ,那么这个变量就能设置了。

6.2.3. NOFETCHFILES (不自动抓取指定的文件)

在 NOFETCHFILES 变量中定义的文件， 不会自动从 MASTER_SITES 抓取。 一种典型的用例是，
使用来自某个软件供应商提供的 CD-ROM 上的文件。

用于检查在 MASTER_SITES 上是否包含了所需文件的工具， 应忽略这些文件， 而不是报告它们不存在。

6.2.4. RESTRICTED (禁止任何形式的再分发)

如果应用程序既不允许镜像其 DISTFILES， 也不允许发布其预编译版本的包， 设置它就可以了。

NO_CDROM 或 NO_PACKAGE 不应与 RESTRICTED 同时设置， 因为它包含了这些情形。

RESTRICTED 应设置为一个说明 port 为何不能发布的串。 典型情况可能是由于 port 包含了专有的软件，
因而用户需要自行下载 DISTFILES， 可能是注册或者同意某一 EULA 的条款。

6.2.5. RESTRICTED_FILES (禁止某些文件的再分发)

当设置了 RESTRICTED 或 NO_CDROM 时， 这个变量会默认设置为 ${DISTFILES} ${PATCHFILES}，
否则它会为空。 如果只有某些源码包文件是受限的， 则可以用这个变量来指明它们。

注意， port committer 应该在 /usr/ports/LEGAL 中为每一个源码包文件撰写对应的项目，
并介绍这些限制的原因。

6.3. 联编机制

6.3.1. Ports 的并行联编

FreeBSD ports 框架支持使用多个 make 子进程来进行并行编译， 在 SMP 上这可以全面地利用系统的 CPU 计算能力， 令 port 的联编过程更快、 更有效率。

目前这是通过向原作者的代码传递 make(1) 参数 -jX 来实现的。 遗憾的是， 并不是所有的 port
都能够很好地处理这个选项。 因此， 必须通过明确地在 Makefile 中指定 MAKE_JOBS_SAFE=yes
来启用这一功能。

从 port 监护人的角度还有一个控制的方法是设置 MAKE_JOBS_UNSAFE=yes 变量。
这个变量主要是用于已知不能与 -jX 配合使用的 port， 即使用户在 /etc/make.conf 中定义了
FORCE_MAKE_JOBS=yes 变量， 系统也不会使用并行编译。

6.3.2. make、 gmake， 以及 imake

如果 port 用到了 GNU make， 应设置 USE_GMAKE=yes。

表 4. 与 gmake 有关的 port 变量

变量	意义
USE_GMAKE	此 port 需要使用 gmake 来完成联编过程。
GMAKE	不在 PATH 中时， gmake 的完整路径。

对于 X 应用程序的 port， 如果它使用 imake 根据 Imakefile 文件来生成 Makefile， 则应设置 USE_IMAKE=yes。 这会使联编过程中的配置 (configure) 阶段自动执行 xmkmf -a。 如果 -a 标志会给您的 port 带来麻烦， 则需设置 XMKMF=xmkmf。 如果 port 用到了 imake 但并不使用 install.man target， 则应设置 NO_INSTALL_MANPAGES=yes。

如果 port 源文件的 Makefile 的主联编 target 是 all 以外的名字， 应对应地设置 ALL_TARGET。 对于 install 而言， 对应的变量是 INSTALL_TARGET。

6.3.3. configure 脚本

假如 port 使用 configure 脚本来从 Makefile.in 生成 Makefile 文件， 需要设置 GNU_CONFIGURE=yes。 如果希望传额外的参数给 configure 脚本 (默认参数为 --prefix=${PREFIX} --infodir=${PREFIX}/${INFO_PATH} --mandir=${MANPREFIX}/man --build=${CONFIGURE_TARGET})， 应通过 CONFIGURE_ARGS 来指定这些参数。 类似地， 可以通过 CONFIGURE_ENV 变量来传递一些环境变量。

如果您的软件包使用 GNU configure， 而生成的可执行文件命名方式 "怪异" 如 i386-portbld-freebsd4.7- 应用程序名， 则需要更进一步地通过改变 CONFIGURE_TARGET 变量来按照较新版本的 autoconf 生成的脚本所希望的方式指定 target。 其方法是， 紧随 Makefile 中 GNU_CONFIGURE=yes 一行之后加入：

```
CONFIGURE_TARGET=--build=${MACHINE_ARCH}-portbld-freebsd${OSREL}
```

表 5. 用于用到了 configure 脚本的 port 的变量

变量	意义
GNU_CONFIGURE	此 port 需要用 configure 脚本来准备联编。
HAS_CONFIGURE	与 GNU_CONFIGURE 类似， 但默认的 configure target 并不加入 CONFIGURE_ARGS。
CONFIGURE_ARGS	希望传给 configure 脚本的额外参数。
CONFIGURE_ENV	希望在执行 configure 脚本时设置的环境变量。
CONFIGURE_TARGET	替换默认的 configure target。 其默认值是 ${MACHINE_ARCH}-portbld-freebsd${OSREL}。

6.3.4. 使用 scons

如果您的 port 使用 SCons， 就需要定义 USE_SCONS=yes 了。

表 6. 使用 scons 的 port 会用到的变量

变量	含义
SCONS_ARGS	当前 port 希望传给 SCons 环境的参数。
SCONS_BUILDENV	希望在系统环境中设置的变量。
SCONS_ENV	希望在 SCons 环境中设置的变量。
SCONS_TARGET	传递给 SCons 的最后一个参数， 类似于 MAKE_TARGET。

如果希望让第三方的 SConstruct 尊重通过 SCONS_ENV (其中最重要的是 CC/CXX/CFLAGS/CXXFLAGS 配置) 传给 Scons 的配置， 则需要对 SConstruct 进行修改， 使联编的 Environment 按下列方式建立：

```
env = Environment(**ARGUMENTS)
```

其后， 可以通过 env.Append 和 env.Replace 来对它进行修改。

6.4. 利用 GNU autotools

6.4.1. 入门

众多 GNU autotools 提供了一种在多重操作系统和机器架构之上联编软件的抽象机制。 在 Ports Collection 中， port 可以通过简单的方法来使用这些工具：

```
USE_AUTOTOOLS= 工具:版本[:操作] ...
```

撰写本书时， 工具 可以设置为 libtool、 libltdl、 autoconf、 autoheader、 automake 或 aclocal 之一。

版本 用来指定希望使用的工具的特定版本 (参见 devel/{automake,autoconf,libtool}[0-9]+ 以了解有效的版本号)。

操作 是一个可选的扩展选项， 用于修改如何使用工具。

可以同时指定多个不同的工具， 可以在一行中指定， 也可以用 Makefile 的 += 结构。

最后， 可以使用一个特殊的名为 autotools 的工具， 它会安装全部可用的 autotools 版本， 以适应跨平台开发的需要。 您可以通过安装 devel/autotools port 来达到这一目的。

6.4.2. libtool

使用 GNU 联编框架的共享库通常会使用 libtool 来调整共享库的编译和安装， 以便与所运行的操作系统相匹配。 通常的做法是使用应用程序所附带的 libtool 副本。 如果需要使用外部的 libtool， 则可以使用 Ports 套件提供的版本：

```
USE_AUTOTOOLS= libtool:版本[:env]
```

如果不使用额外的操作符， libtool:版本 表示希望联编框架使用 configure 脚本来对系统所安装的 libtool 进行修补。 这会暗含地定义 GNU_CONFIGURE。 更进一步， 联编框架还会设置一系列 make 和 shell 变量用于 port 后续的操作。 请参见 bsd.autotools.mk 了解进一步的详情。

如果指定了 :env 操作符， 则表示只设置环境， 而跳过其他的操作。

最后， LIBTOOLFLAGS 和 LIBTOOLFILES 可以用来替换最常修改的参数， 以及将被 libtool 修补的文件。 多数 port 不需要这样做。 请参见 bsd.autotools.mk 以了解进一步的细节。

6.4.3. libltdl

一些 ports 会使用 libltdl 库， 后者是 libtool 软件包的一部分。 使用这个库并不意味着必须使用 libtool 本身， 因此提供了另一组结构。

```
USE_AUTOTOOLS= libltdl:版本
```

目前， 这一设置所做的全部工作是将 LIB_DEPENDS 设置为适当的 libltdl port， 并作为一项方便的功能， 协助开发人员消除在 USE_AUTOTOOLS 框架以外的， 对于 autotools port 的依赖。 这个工具并不提供其它的操作符。

6.4.4. autoconf 和 autoheader

某些 port 并没有直接提供 configure 脚本， 但包含了作为 autoconf 模板的 configure.ac 文件。 可以用下列设置来要求 autoconf 创建 configure 脚本， 并使用 autoheader 来为 configure 脚本创建模板头文件。

```
USE_AUTOTOOLS= autoconf:版本[:env]
```

以及

```
USE_AUTOTOOLS= autoheader:版本
```

上述设置会暗含使用 autoconf:版本。

对于 libtool， 设置与前面类似。 如果指定可选的 :env 操作符， 则表示只设置用于后续工作的环境。 如果不指定， 则会对 port 进行相应的修补和重新配置。

其它的可选变量， 如 AUTOCONF_ARGS 和 AUTOHEADER_ARGS 可以通过 port 的 Makefile 来显式地指定替换。 类似 libtool， 多数 port 并不需要这样做。

6.4.5. automake 和 aclocal

某些软件包只提供了 Makefile.am 文件。 这些文件必须首先用 automake 转换为 Makefile.in 并使用 configure 来生成实际的 Makefile。

类似地， 偶尔会有一些软件包不提供联编所需的 aclocal.m4 文件。 这些文件可以通过使用 aclocal 来扫描 configure.ac 或 configure.in 自动生成。

aclocal 与 automake 有和 autoheader 与 autoconf 在前面一节中所介绍的相类似的关系。 aclocal 会暗含使用 automake， 因此：

```
USE_AUTOTOOLS= automake:版本[:env]
```

和

```
USE_AUTOTOOLS= aclocal:版本
```

也自动暗含使用 automake:版本。

与 libtool 类似， autoconf 如果使用了可选的 :env 操作符表示仅仅设置用于后续使用的环境， 如果不设置， 则会对 port 进行重新配置。

对于 autoconf 和 autoheader 而言， automake 和 aclocal 提供了对应的可选参数变量 AUTOMAKE_ARGS 和 ACLOCAL_ARGS， 如果需要的话， 可以在 port 的 Makefile 中指定。

6.5. 使用 GNU gettext

6.5.1. 基本用法

如果您的 port 需要使用 gettext， 只要将 USE_GETTEXT 设置为 yes， 您的 port 就会增加对 devel/gettext 的依赖。 USE_GETTEXT 也可以指定为所需的 libintl 库的版本， 它是 gettext 的基本组成部分， 尽管如此， 强烈建议您不要 使用这个功能： 您的 port 应能与目前版本的 devel/gettext 配合工作。

在 port 中相当常见的情况下， 会需要同时使用 gettext 和 configure。 一般而言， GNU configure 能够自动定位到 gettext。 如果它没有成功地完成这项工作， 则可以通过类似下面这样的 CPPFLAGS 和 LDFLAGS 将 gettext 的位置告诉它：

```
USE_GETTEXT=   yes
CPPFLAGS+=    -I${LOCALBASE}/include
LDFLAGS+=     -L${LOCALBASE}/lib

GNU_CONFIGURE= yes
CONFIGURE_ENV= CPPFLAGS="${CPPFLAGS}" \
        LDFLAGS="${LDFLAGS}"
```

当然， 不需要传参数给 configure 时， 代码可以更为简练：

```
USE_GETTEXT=   yes
GNU_CONFIGURE= yes
CONFIGURE_ENV= CPPFLAGS="-I${LOCALBASE}/include" \
        LDFLAGS="-L${LOCALBASE}/lib"
```

6.5.2. 可选用法

一些软件产品提供了禁用 NLS 的能力， 例如， 在 configure 时， 指定 --disable-nls 参数。 如果您 port 的软件支持这种配置， 则应根据 WITHOUT_NLS 的设置来有条件地使用 gettext。
对于比较简单和不太复杂的 port， 您可以使用下列结构：

```
GNU_CONFIGURE=      yes

.if !defined(WITHOUT_NLS)
USE_GETTEXT=        yes
PLIST_SUB+=         NLS=""
.else
CONFIGURE_ARGS+=    --disable-nls
PLIST_SUB+=         NLS="@comment "
.endif
```

您要做的下一件事是合理地安排装箱单文件， 使其根据用户配置来决定是否将消息编录 (message catalog) 文件放入最终的装箱单。 前面已经介绍了在 Makefile 中所需的写法， 这种做法在 高级 pkg-plist 用法 这节中进行了介绍。 简单地说， 在 pkg-plist 中出现的 %%NLS%% 均会在禁用 NLS 时自动替换为 “`@comment `”， 反之则替换为空串。 这样， 在最终的装箱单中带 %%NLS%% 的行， 在 NLS 关闭的情况下就会变为注释， 反之， 这些前缀就会自动删掉。 现在需要做的事情就是把 %%NLS%% 插到 pkg-plist 中的消息编录文件的那些行开头， 例如：

```
%%NLS%%share/locale/fr/LC_MESSAGES/foobar.mo
%%NLS%%share/locale/no/LC_MESSAGES/foobar.mo
```

在比较复杂的情形中， 您可能需要使用更高级的技术， 例如 动态生成装箱单 等。

6.5.3. 处理消息编录目录

在安装消息编录文件时还有一个需要注意的地方。 这些文件会放到 LOCALBASE/shared/locale

下与语言对应的目录中，这些目录一般您的 port 不需要创建和删除。 最常用的语言的目录已经在 /etc/mtree/BSD.local.dist 中列出； 也就是说， 它们是基本系统的一部分。 其他一些语言的目录， 则由 devel/gettext 控制。 您最好查看一下 pkg-plist， 以确定是否正在安装某种不常用语言的文件。

6.6. 使用 perl

如果 MASTER_SITES 设为 MASTER_SITE_PERL_CPAN， 则应尽量把 MASTER_SITE_SUBDIR 设置为顶级目录的名字。 例如， 对 p5-Module-Name 而言推荐的名字是 Module。 您可以在 cpan.org 找到顶级目录的名字。 这可以确保在模块的作者发生变化时， 保持 port 继续可用。

以上规则有一个例外， 即对应目录不存在或源码包不在那个目录中时， 允许使用作者的 id 作为 MASTER_SITE_SUBDIR。

所有这些选项均同时接受 YES 和版本串， 类似 5.8.0+ 这样的写法。 使用 YES 表示 port 能够配合所有受支持的 Perl 版本来使用。 如果 port 只能配合特定版本的 Perl 来使用， 则可以用版本串来表示， 例如最低版本 (如 5.7.3+)、 最高版本 (如 5.8.0-) 或某个具体的版本 (如 5.8.3)。

表 7. 用于用到 perl 的 port 的变量

变量	意义
USE_PERL5	表示 port 将 perl 5 用于联编和运行。
USE_PERL5_BUILD	表示 port 将 perl 5 用于联编。
USE_PERL5_RUN	表示 port 将 perl 5 用于运行。
PERL	perl 5 的完整路径， 可能是系统自带的， 或者从 port 安装， 但没有版本号。 如果您需要在脚本中替换 “#!” 行， 则应使用这个变量。
PERL_CONFIGURE	采用 Perl 的 MakeMaker 进行配置。 这一变量隐含设置 USE_PERL5。
PERL_MODBUILD	使用 Module::Build 进行配置、 联编并安装。 这一变量隐含设置 PERL_CONFIGURE。



Perl 模块通常并没有官方网站， 这些 port 应将 cpan.org 作为其 pkg-descr WWW 行的内容。 推荐的 URL 格式为 http://search.cpan.org/dist/Module-Name/ (保留最后的斜线)。

6.7. 使用 X11

6.7.1. X.Org 组件

在 Ports 套件中提供的 X11 实现是 X.Org。 如果您的应用程序用到了 X 组件， 则应将 USE_XORG 设为所需要的那些组件。 目前可用的组件包括：

bigreqsproto compositeproto damageproto dmx dmxproto evieproto fixesproto fontcacheproto fontenc fontsproto fontutil glproto ice inputproto kbproto libfs oldx printproto randrproto recordproto renderproto resourceproto scrnsaverproto sm trapproto videoproto x11 xau xaw xaw6 xaw7 xaw8 xbitmaps xcmiscproto xcomposite xcursor xdamage xdmcp xevie xext xextproto xf86bigfontproto xf86dgaproto xf86driproto xf86miscproto xf86rushproto xf86vidmodeproto xfixes xfont xfontcache xft xi xinerama xineramaproto xkbfile xkbui xmu xmuu xorg-server xp xpm xprintapputil xprintutil xpr oto xproxymngproto xrandr xrender xres xscrnsaver xt xtrans xtrap xtst xv xvmc xxf86dga xxf86misc xxf86vm.

最新的列表， 可以在 /usr/ports/Mk/bsd.xorg.mk 中找到。

The Mesa Project 是一个致力于自由的 OpenGL 实现的计划。 您可以使用 USE_GL 变量来让 port 依赖其不同的组件。 可用的选项包括： glut, glu, glw, glew, gl 和 linux。 为了实现向前兼容， 当使用 yes 时系统会自动将其映射为 glu。

例 12. 使用 USE_XORG 的例子

```
USE_XORG=  xrender xft xkbfile xt xaw
USE_GL=    glu
```

许多 ports 会定义 USE_XLIB， 这会导致 port 依赖 50 多个动态连接库。 由于它出现于 X.org 模块化之前，因此这个变量仅为向前兼容的原因提供， 新的 port 不应再使用它。

表 8. 用到 X 的 port 可以使用的变量

USE_XLIB	此 port 用到了 X 库。 已过时 - 您应使用 USE_XORG 变量列出用到的 X.Org 组件，而不是使用这个变量。
USE_IMAKE	此 port 用到了 imake。
USE_X_PREFIX	已过时。 目前其作用与 USE_XLIB 相同，并可以直接用后者替换。
XMKMF	设置为 xmkmf 的完整路径名， 如果它不在 PATH 中的话。 默认值是 xmkmf -a。

表 9. 用于表示对 X11 某些组件的依赖关系的变量

X_IMAKE_PORT	用以提供 imake 以及许多其它用于联编 X11 的工具的 port。
X_LIBRARIES_PORT	用以提供 X11 库的 port。
X_CLIENTS_PORT	用以提供 X 客户的 port。
X_SERVER_PORT	用以提供 X 服务的 port。
X_FONTSERVER_PORT	用以提供字体服务的 port。
X_PRINTSERVER_PORT	用以提供打印服务的 port。
X_VFBSERVER_PORT	用以提供在虚拟帧缓存服务(virtual framebuffer server) 的 port。
X_NESTSERVER_PORT	用以提供嵌套 X 服务的 port。
X_FONTS_ENCODINGS_PORT	用以为字体提供编码的 port。
X_FONTS_MISC_PORT	用以提供多种位图字体的 port。
X_FONTS_100DPI_PORT	用以提供 100dpi 位图字体的 port。
X_FONTS_75DPI_PORT	用以提供 75dpi 位图字体的 port。
X_FONTS_CYRILLIC_PORT	用以提供西里尔位图字体的 port。
X_FONTS_TTF_PORT	用以提供 TrueType® 字体的 port。
X_FONTS_TYPE1_PORT	用以提供 Type1 字体的 port。
X_MANUALS_PORT	用以提供面向开发人员的联机手册的 port。

例 13. 在变量中使用与 X11 有关的变量

```
# 使用某些 X11 库并依赖字体服务和西里尔字体。
RUN_DEPENDS=  ${LOCALBASE}/bin/xfs:${X_FONTSERVER_PORT} \
        ${LOCALASE}/lib/X11/fonts/cyrillic/crox1c.pcf.gz:${X_FONTS_CYRILLIC_PORT}

USE_XORG=    x11 xpm
```

6.7.2. 需要使用 Motif 的 port

如果您的 port 需要 Motif 库， 则应在 Makefile 中定义 USE_MOTIF。 默认的 Motif 实现是 x11-toolkits/open-motif。 用户可以通过设置 WANT_LESSTIF 变量来选择 x11-toolkits/lesstif 代替它。

bsd.port.mk 会将 MOTIFLIB 变量设置为到合适的 Motif 库的引用。 请使用补丁将您 port 中 Makefile 或 Imakefile 提到 Motif 库的地方改为 ${MOTIFLIB}。

有两种比较常见的情况：

- 如果 port 中将 Motif 在其 Makefile 或 Imakefile 表达为 -lXm， 则简单地将其替换为 ${MOTIFLIB}。
- 如果 port 在其 Imakefile 中使用 XmClientLibs， 则将其改为 ${MOTIFLIB} ${XTOOLLIB} ${XLIB}.

注意 MOTIFLIB (通常) 会展开为 -L/usr/X11R6/lib -lXm 或 /usr/X11R6/lib/libXm.a，
所以不需要在其前加入 -L 或 -l。

6.7.3. X11 字体

如果 port 将为 X Window 系统安装字体， 将这些字体放到 LOCALBASE/lib/X11/fonts/local。

6.7.4. 通过 Xvfb 来获得虚拟的 DISPLAY

某些应用程序必须在有可用的 X11 显示的时候才能成功编译。 当编译的机器没有控制台时，
这会带来问题。 为了解决这个问题， 如果定义了适当的变量， 联编基础设施会启动采用虚拟帧缓存的 X server。 此时， 编译过程中将会传出可用的 DISPLAY。

```
USE_DISPLAY= yes
```

6.7.5. 桌面项

通过利用 DESKTOP_ENTRIES 变量， 可以很容易地在您的 port 中创建桌面项 (Freedesktop 标准)。
这些项会在类似 GNOME 或 KDE 这样的符合这一标准的桌面环境中显示在应用程序菜单中。
这样做会自动创建、 安装 .desktop 文件， 并将其加入 pkg-plist。 其语法为：

```
DESKTOP_ENTRIES= "NAME" "COMMENT" "ICON" "COMMAND" "CATEGORY"
StartupNotify
```

您可以在 Freedesktop 网站上 找到可用的分类名称。 StartupNotify
表示应用程序在支持启动通知的环境中清除状态信息。

例子：

```
DESKTOP_ENTRIES= "ToME" "Roguelike game based on JRR Tolkien's work" \
        "${DATADIR}/xtra/graf/tome-128.png" \
        "tome -v -g" "Application;Game;RolePlaying;" \
        false
```

6.8. 使用 GNOME

FreeBSD/GNOME 项目组使用一组自己的变量来定义 port 所使用的 GNOME 组件。 这些变量的详细列表
可以在 FreeBSD/GNOME 项目的主页找到。

6.9. 使用 Qt

6.9.1. 在 port 中使用 Qt

表 10. 用于使用 Qt 的 port 的变量

USE_QT_VER	表示 port 用到了 Qt 工具套件。 可用的值包括 3 和 4； 用于指定使用的 Qt 的主版本。 此外，系统会自动为 configure 脚本和 make 命令提供必要的参数。
QT_PREFIX	这个变量会自动设为 Qt 的安装路径 (只读变量)。
MOC	这个变量会自动设为 moc 的路径 (只读变量)。 默认值与 USE_QT_VER 变量的值有关。
QTCPPFLAGS	通过 CONFIGURE_ENV 传给 Qt 工具套件的编译参数。 默认配置与 USE_QT_VER 有关。
QTCFGLIBS	通过 CONFIGURE_ENV 传给 Qt 工具套件的连接库。 默认配置与 USE_QT_VER 有关。
QTNONSTANDARD	禁止系统自动修改 CONFIGURE_ENV、CONFIGURE_ARGS 和 MAKE_ENV。

表 11. 其他用于使用 Qt 4.x 的变量

QT_COMPONENTS	用于指定 Qt4 工具和函数库的依赖。 详情见后。
UIC	这个变量会自动设为 uic 的路径 (只读变量)。 默认值与 USE_QT_VER 有关。
QMAKE	这个变量会自动设为 qmake 的路径 (只读变量)。 其默认值与 USE_QT_VER 有关。
QMAKESPEC	这个变量会自动设为 qmake 配置文件的路径 (只读变量)。 其默认值与 USE_QT_VER 有关。

当设置了 USE_QT_VER 时， 系统自动会给 configure 脚本传一系列有用的参数：

```
CONFIGURE_ARGS+= --with-qt-includes=${QT_PREFIX}/include \
        --with-qt-libraries=${QT_PREFIX}/lib \
        --with-extra-libs=${LOCALBASE}/lib \
        --with-extra-includes=${LOCALBASE}/include
CONFIGURE_ENV+=  MOC="${MOC}" CPPFLAGS="${CPPFLAGS} ${QTCPPFLAGS}"
LIBS="${QTCFGLIBS}" \
        QTDIR="${QT_PREFIX}" KDEDIR="${KDE_PREFIX}"
```

如果将 USE_QT_VER 设为 4， 则还会进行下列配置：

```
CONFIGURE_ENV+= UIC="${UIC}" QMAKE="${QMAKE}" QMAKESPEC="${QMAKESPEC}"
MAKE_ENV+=    QMAKESPEC="${QMAKESPEC}"
```

6.9.2. 组件的选择 (仅限 Qt 4.x)

当把 USE_QT_VER 设为 4 时， 就可以通过 QT_COMPONENTS 变量来指定对 Qt4 工具和函数库的依赖了。 通过在组件的名称后面添加 _build 或 _run 这样的后缀， 则可相应地将这依赖关系限于联编或运行时刻。 在没有指定后缀时， 系统默认在联编和运行时刻均依赖该组件。

通常情况下在指明函数库一类的组件时应不使用后缀， 联编工具类组件应使用 _build 后缀，
而插件类组件， 则应使用 _run 后缀。 下表中列出了一些最常用的组件 (全部可用的组件， 则在
/usr/ports/Mk/bsd.qt.mk 中的 _QT_COMPONENTS_ALL 列出)：

表 12. 可用的 Qt4 函数库组件

名字	描述
corelib	核心库 (在 port 只使用 corelib 而没有用到其他库时可以省略)
gui	图形用户界面库
network	网络函数库
opengl	OpenGL 函数库
qt3support	Qt3 兼容支持函数库
qtestlib	单元测试函数库
script	脚本函数库
sql	SQL 函数库
xml	XML 函数库

您可以通过在成功编译之后， 通过在主可执行文件上运行 ldd 来确定所需的库。

表 13. 可用的 Qt4 工具组件

名字	描述
moc	元对象编译器 (几乎所有的 Qt 应用程序在联编过程中都需要它)
qmake	Makefile 生成器 / 联编工具
rcc	资源编译器 (如果应用程序中包含 .rc 或.qrc 文件，就需要它)
uic	用户界面编译器 (如果应用程序中包含使用 Qt Designer 创建的 *.ui 文件时就需要它 - 一般说来 Qt 应用程序都会使用 GUI 的)

表 14. 可用的 Qt4 插件组件

名字	描述
iconengines	SVG 图标引擎插件 (如果应用程序使用 SVG 图标)
imageformats	用于 GIF、 JPEG、 MNG 和 SVG 的 imageformat 插件 (如果应用程序使用图片文件)

例 14. 选择 Qt4 组件

在这个例子中， 我们将要移植的应用程序用到了 Qt4 图形用户界面函数库、 Qt4 核心 (core) 函数库、
所有 Qt4 代码生成工具以及 Qt4 的 Makefile 生成器。 由于 gui
函数库会自动附带对核心函数库的依赖， 因此并不需要明确指出需要 corelib 的依赖关系。 Qt4
代码生成工具 moc、 uic 和 rcc 以及 Makefile 生成器 qmake 只在联编过程中才会用到，
因此可以指定 _build 后缀：

```
USE_QT_VER=  4
QT_COMPONENTS= gui moc_build qmake_build rcc_build uic_build
```

6.9.3. 其他考虑

如果应用程序没有提供 configure 文件， 而是给了一个 .pro 文件， 则应这样：

```
HAS_CONFIGURE= yes

do-configure:
    @cd ${WRKSRC} && ${SETENV} ${CONFIGURE_ENV} \
        ${QMAKE} -unix PREFIX=${PREFIX} texmaker.pro
```

请注意， 这与系统提供的 BUILD.sh 中的 qmake 类似。 传递 CONFIGURE_ENV 能够确保 qmake 可以看到 QMAKESPEC 变量， 否则它可能无法正常工作。 qmake 会生成标准的 Makefile， 因此无需自行编写 build target。

Qt 应用程序通常会编写为能够跨平台使用， 通常 X11/Unix 并不是开发它的平台，
有时这会导致一些边边角角的问题， 例如：

- 缺少必要的 includepaths。 许多应用程序会使用托盘图标支持， 但忽略了这些头或库文件需要在 X11 目录中查找。 您可以通过命令行告诉 qmake 将这些头文件和函数库加入到搜索路径中， 例如：

  ```
  ${QMAKE} -unix PREFIX=${PREFIX} INCLUDEPATH+=${LOCALBASE}/include \
    LIBS+=-L${LOCALBASE}/lib sillyapp.pro
  ```

- 有问题的安装路径。 有时， 类似图标或 .desktop 文件这样的一些数据， 默认情况下没有安装到 XDG-兼容的程序会扫描的路径中。 editors/texmaker 就是一个这样的例子 - 请参考这个 port 的 files 目录中的 patch-texmaker.pro， 以了解如何在 Qmake 工程文件中修正这个问题。

6.10. 使用 KDE

6.10.1. 变量定义 (只用于 KDE 3.x)

表 15. 用于使用 KDE 3.x 的 port 的变量

USE_KDELIBS_VER	表示 port 用到了 KDE 库。 这个变量可以指定希望使用的 KDE 主版本号， 如果设置了这个变量， 则系统也会将 USE_QT_VER 设为适当的版本。 该变量目前唯一有效的值是 3。
USE_KDEBASE_VER	表示 port 用到了 KDE 的基本系统。 这个变量可以指定希望使用的 KDE 主版本号， 如果设置了这个变量， 则系统也会将 USE_QT_VER 设为适当的版本。 该变量目前唯一有效的值是 3。

6.10.2. 用于 KDE 4 的变量定义

如果您的应用程序需要使用 KDE 4.x， 则应将 USE_KDE4 设为所需组件的列表。
下面列出一些最常用到的组件 (最新的组件列表位于 /usr/ports/Mk/bsd.kde4.mk 中的 _USE_KDE4_ALL)：

表 16. 可用的 KDE4 组件

名称	说明
akonadi	个人信息管理 (PIM)存储服务
automoc4	令 port 使用 automoc4 联编工具集

名称	说明
kdebase	基本的 KDE 应用程序 (Konqueror、 Dolphin、 Konsole)
kdeexp	试验性的 KDE 库 (包含尚未完全确定不变的 API)
kdehier	常用的 KDE 目录层次结构
kdelibs	基本 KDE 库
kdeprefix	如果设置了这个选项， 则 port 将安装到 ${KDE4_PREFIX} 而不是 ${LOCALBASE}
pimlibs	PIM 函数库
workspace	用于组成桌面的应用程序和函数库 (Plasma、 KWin)

KDE 4.x port 会安装到 ${KDE4_PREFIX}， 目前是 /usr/local/kde4， 以避免与 KDE 3.x ports 冲突。 这是通过指定 kdeprefix 组件来实现的， 它表示替换默认的 PREFIX。 不过， port 仍会遵循通过 MAKEFLAGS 环境变量设置的 PREFIX 以及其它 make 参数。

KDE 4.x ports 有可能和 KDE 3.x ports 冲突， 因此如果启用了 kdeprefix 组件， 它们会安装到 ${KDE4_PREFIX}。 目前 KDE4_PREFIX 的默认值是 /usr/local/kde4。 也可以将 KDE 4.x ports 安装到自定义的 PREFIX。 当 PREFIX 是通过 MAKEFLAGS 环境变量， 或直接在 make 命令行指定时， 它会替换 kdeprefix 提供的配置。

例 15. USE_KDE4 示例

下面是一个简单的 KDE 4 port。 USE_CMAKE 指定 port 使用 CMake - 许多 KDE 4 项目所使用的配置工具。 USE_KDE4 则引入 KDE 函数库， 并令 port 在联编阶段使用 automoc4。 需要的 KDE 组件， 以及其他依赖的组件可以从 configure 的日志中获知。 USE_KDE4 并不会自动设置 USE_QT_VER。 如果 port 需要使用某些 Qt4 组件， 则需要设置 USE_QT_VER 并指定所需要的组件。

```
USE_CMAKE=    yes
USE_KDE4=     automoc4 kdelibs kdeprefix
USE_QT_VER=   4
QT_COMPONENTS= qmake_build moc_build rcc_build uic_build
```

6.11. 使用 Java

6.11.1. 变量定义

如果您的 port 需要 Java™ 开发包 (JDK™) 来完成联编、 支持运行， 甚至完成解开源代码包这样的工作， 就应该定义 USE_JAVA。

在 Ports Collection 中有许多不同的 JDK， 它们的版本各不相同， 或是来自不同的供应商。 如果您的 port 必须使用其中的某个特定的版本， 也可以予以定义。 最新的稳定版本是 java/jdk16。

表 17. 用到 Java 的 port 可以使用的变量

变量	意义
USE_JAVA	只有定义它才能使其它变量生效。
JAVA_VERSION	用空格分开的适合 port 使用的 Java 版本。 可选的 ""`` 可以用于指定某个范围的版本 (可以用： `1.5[] 1.6[] 1.7[])。
JAVA_OS	用空格分开的适应 port 的 JDK port 操作系统类型 (可以用： native linux)。

变量	意义
JAVA_VENDOR	用空格分开的适应 port 的 JDK port 供应商 (可以用： freebsd bsdjava sun openjdk)。
JAVA_BUILD	设置这个变量表示所选的 JDK port 应被列入 port 的联编依赖关系。
JAVA_RUN	设置这个变量表示所选的 JDK port 应被列入 port 的运行环境依赖关系。
JAVA_EXTRACT	设置这个变量表示所选的 JDK port 应被列入 port 的解压缩支持依赖关系。

下面是在设置了 USE_JAVA 之后， port 能够从系统中获得的配置：

表 18. 向使用了 Java 的 port 提供的变量

变量	值
JAVA_PORT	JDK port 的名字 (例如 'java/diablo-jdk16')。
JAVA_PORT_VERSION	JDK port 的完整版本 (例如 '1.6.0')。 如果您只需要版本号的前两位， 则可用 ${JAVA_PORT_VERSION:C/^([0-9])\.([0-9])(.*)$/\1.\2/}。
JAVA_PORT_OS	所用 JDK port 的操作系统 (例如 'native')。
JAVA_PORT_VENDOR	所用 JDK port 的供应商 (例如 'freebsd')。
JAVA_PORT_OS_DESCRIPTION	所用 JDK port 操作系统的描述 (例如 'Native')。
JAVA_PORT_VENDOR_DESCRIPTION	所用 JDK port 供应商的描述 (例如 'FreeBSD Foundation')。
JAVA_HOME	JDK 的安装目录 (例如 '/usr/local/diablo-jdk1.6.0')。
JAVAC	所用 Java 编译器的完整路径 (例如 '/usr/local/diablo-jdk1.6.0/bin/javac')。
JAR	所用 jar 工具的完整路径 (例如 '/usr/local/diablo-jdk1.6.0/bin/jar' 或 '/usr/local/bin/fastjar')。
APPLETVIEWER	所用 appletviewer 工具的完整路径 (例如 '/usr/local/diablo-jdk1.6.0/bin/appletviewer')。
JAVA	所用 java 执行文件的完整路径。 您应使用它来执行 Java 程序 (例如 '/usr/local/diablo-jdk1.6.0/bin/java')。
JAVADOC	所用 javadoc 工具的完整路径。
JAVAH	所用 javah 程序的完整路径。
JAVAP	所用 javap 程序的完整路径。
JAVA_KEYTOOL	所用 keytool 工具的完整路径。
JAVA_N2A	所用 native2ascii 工具的完整路径。
JAVA_POLICYTOOL	所用 policytool 程序的完整路径。
JAVA_SERIALVER	所用 serialver 程序的完整路径。
RMIC	所用 RMI 桩/架 生成器， rmic 的完整路径。
RMIREGISTRY	所用 RMI 注册表程序， rmiregistry 的完整路径。
RMID	所用 RMI 服务程序 rmid 的完整路径。
JAVA_CLASSES	所用 JDK 类文件目录的完整路径。 ${JAVA_HOME}/jre/lib/rt.jar。

您可以使用 java-debug make target 以获取用于调试 port 的信息。 大多数前述变量的值皆会予以呈现。

此外， 还会定义下述常量， 以确保所有的 Java port 均以一致之方式安装：

表 19. 为使用 Java 的 port 定义的常量

常量	值
JAVASHAREDIR	所有 Java 相关资料的安装根目录。 默认值：${PREFIX}/shared/java.
JAVAJARDIR	用以安装 JAR 文件的目录。 默认值：${JAVASHAREDIR}/classes。
JAVALIBDIR	其它 port 安装的 JAR 文件所在的目录。 默认值：${LOCALBASE}/shared/java/classes。

相关的项也会定义在 PLIST_SUB (在 根据 make 变量对 pkg-plist 进行修改 中进行介绍) 和 SUB_LIST 中。

6.11.2. 采用 Ant 进行联编

如果 port 采用 Apache Ant 进行联编， 则需要定义 USE_ANT。 如是， 则 Ant 将作为 子-make 命令来使用。 如果 port 未定义 do-build target， 则将默认依 MAKE_ENV、 MAKE_ARGS 和 ALL_TARGET。 的设置执行 Ant。 这类似于 联编机制 中介绍的关于 USE_GMAKE 的机制。

6.11.3. 最佳实践

如果您正移植某个 Java 库， 您的 port 应把 JAR 文件安装到 ${JAVAJARDIR}， 而其它文件则应放在 ${JAVASHAREDIR}/${PORTNAME} 下 (除了文档， 参见下文)。 要减少打包文件的尺寸， 您可以直接在 Makefile 中引用这些 JAR 文件， 具体做法是使用下面的语句 (此处的 myport.jar 是作为 port 一部分安装的 JAR 文件的名字)：

```
PLIST_FILES+= %%JAVAJARDIR%%/myport.jar
```

移植 Java 应用程序时， port 通常会希望将所有文件安装到同一目录 (包括其依赖的 JAR)。 这时强烈建议使用 ${JAVASHAREDIR}/${PORTNAME}。 移植软件的开发人员， 可以自行决定是否将所依赖的其它 JAR 安装到此目录， 或直接使用已经装好的那些 (来自 ${JAVAJARDIR})。

无论您正制作哪一类的 port (库或者应用程序)， 附加的文档都应安装到和其它 port 同样的位置。 已经知道， JavaDoc 会根据 JDK 版本的不同而产生不同的文件。 对于那些不打算强制使用某一特定版本 JDK 的 port 而言， 这无疑提高了制作装箱单 (pkg-plist) 的难度。 这是为什么强烈建议使用 PORTDOCS 宏的原因。 更进一步， 即使您能够预测 javadoc 将要生成的文件， 所需的 pkg-plist 的尺寸， 也是鼓吹使用 PORTDOCS 的一大理由。

DATADIR 的默认值是 ${PREFIX}/shared/${PORTNAME}。 对 Java port 而言将 DATADIR 改为 ${JAVASHAREDIR}/${PORTNAME} 是一个好主意。 当然， DATADIR 会自动加到 PLIST_SUB 中 (在 根据 make 变量对 pkg-plist 进行修改 有所介绍) 因此您可以在 pkg-plist 中直接使用 %%DATADIR%%。

撰写本文时， 对是应该从源代码联编， 还是直接安装预编译的 Java ports 安装包并没有明确的规定。 尽管如此， FreeBSD Java Project 的开发人员仍鼓励移植软件的开发者在不麻烦的情况下尽可能从源代码完成联编。

本节中所介绍的全部特性， 均是在 bsd.java.mk 中实现的。 如果您感觉自己的 port 需要更为复杂的 Java 支持， 请首先参阅 bsd.java.mk CVS 日志， 因为通常撰文介绍最新特性需要一些时间。 此外， 如果您认为所缺少的支持对许多其它 Java port 亦属有益， 请在 freebsd-java 对其进行讨论。

在 PR 中的 java 类别， 主要是用于 FreeBSD Java project 移植 JDK 本身之用。 因而， 提交您的 Java port 时， 应归入 ports 类别， 除非您正尝试解决的问题是 JDK 实现本身或 bsd.java.mk 的。

类似地， 您应参考 分类 中所详述的关于 CATEGORIES 在 Java port 中的使用规则。

6.12. Web 应用， Apache 和 PHP

6.12.1. Apache

表 20. 用到 Apache 的 port 可以使用的变量

USE_APACHE	此 port 需要 Apache。 可用的值： yes (任意可用版本)、 1.3、 2.0、 2.2、 2.0+、 等等。 默认依赖的版本是 1.3。
WITH_APACHE2	此 port 需要 Apache 2.0。 如果没有这个变量， 则 port 将依赖 Apache 1.3。 这一变量目前已经过时， 因而不应继续使用。
APXS	到 apxs 可执行文件的完整路径。 您可以在 port 中替代该值。
HTTPD	到 httpd 可执行文件的完整路径。 您可以在 port 中替代该值。
APACHE_VERSION	目前系统中安装的 Apache 版本 (只读变量)。 这一变量只有在引用了 bsd.port.pre.mk 之后才能使用， 其可能的值为： 13、 20、 22。
APACHEMODDIR	Apache 模块所在的文件夹。 在 pkg-plist 中， 这一变量会自动展开。
APACHEINCLUDEDIR	Apache 头文件所在的文件夹。 在 pkg-plist 中， 这一变量会自动展开。
APACHEETCDIR	Apache 配置文件所在的文件夹。 在 pkg-plist 中， 这一变量会自动展开。

表 21. 在移植 Apache 模块时比较有用的变量

MODULENAME	模块的名称。 默认值为 PORTNAME。 例如： mod_hello
SHORTMODNAME	模块的简略名字。 默认情况下会自动根据 MODULENAME 计算， 但您也可以自行设置值来替代它。 例如： hello
AP_FAST_BUILD	使用 apxs 来编译和安装模块。
AP_GENPLIST	同时自动创建 pkg-plist。
AP_INC	在编译过程中， 将指定的目录加入到搜索头文件的目录中。
AP_LIB	在编译过程中， 将指定的目录加入到搜索函数库的目录中。
AP_EXTRAS	传递给 apxs 的额外参数。

6.12.2. Web 应用

Web 应用程序应安装到 PREFIX/www/应用程序的名字。 为方便起见， 这个路径在 Makefile 和 pkg-plist 均以 WWWDIR 变量的形式提供。 在 Makefile 中可以使用 WWWDIR_REL 来表示包含了 PREFIX 的该变量值。

web 服务器进程所用的用户和用户组， 分别以 WWWOWN 和 WWWGRP 变量的形式提供， 如果您需要修改某些文件的属主的话。 这两个变量的默认值均为 www。 如果您的 port 希望使用其他值， 请使用 WWWOWN?= myuser 这种写法， 以便让用户能够更容易地修改它。

除非您的 port 必需使用 Apache， 否则不要将其写入依赖关系。 请尊重运行您的应用程序的用户选择 Apache 以外的其他 web 服务器的需求。

6.12.3. PHP

表 22. 用到 PHP 的 port 中可以使用的变量

USE_PHP	此 port 需要 PHP。 取值为 yes 将把 PHP 加入依赖关系。 此外， 还可以在此指定将所需要的 PHP 扩展模块。 例如： pcre xml gettext
DEFAULT_PHP_VER	选择在没有安装 PHP 时自动安装的 PHP 主版本。 默认是 4。 可选 4、 5 之一。
IGNORE_WITH_PHP	此 port 无法与给定版本的 PHP 一同工作。 可选值为 4、 5 之一。
USE_PHPIZE	此 port 将作为 PHP 扩展模块进行联编。
USE_PHPEXT	此 port 将作为 PHP 扩展， 且需要作为扩展模块注册。
USE_PHP_BUILD	联编依赖于 PHP。
WANT_PHP_CLI	希望使用 CLI (命令行) 版本的 PHP。
WANT_PHP_CGI	希望使用 CGI 版本的 PHP。
WANT_PHP_MOD	希望使用 Apache 模块版本的 PHP。
WANT_PHP_SCR	希望使用 CLI 或 CGI 版本的 PHP。
WANT_PHP_WEB	希望使用 Apache 模块或 CGI 版本的 PHP。

6.12.4. PEAR 模块

移植 PEAR 模块的过程非常简单。

使用 FILES、 TESTS、 DATA、 SQLS、 SCRIPTFILES、 DOCS 以及 EXAMPLES 这些变量来指明您希望安装的文件。 所有这里列出的文件都会自动安装到合适的位置， 并加入 pkg-plist。

在 Makefile 文件的最后一行引入 ${PORTSDIR}/devel/pear/bsd.pear.mk。

例 16. 用于 PEAR 类的 Makefile 例子

```
PORTNAME=      Date
PORTVERSION=   1.4.3
CATEGORIES=    devel www pear

MAINTAINER=    example@domain.com
COMMENT=       PEAR Date and Time Zone Classes

BUILD_DEPENDS= ${PEARDIR}/PEAR.php:${PORTSDIR}/devel/pear-PEAR
RUN_DEPENDS=   ${BUILD_DEPENDS}

FILES=         Date.php Date/Calc.php Date/Human.php Date/Span.php    \
               Date/TimeZone.php
TESTS=         test_calc.php test_date_methods_span.php testunit.php  \
               testunit_date.php testunit_date_span.php wknotest.txt  \
               bug674.php bug727_1.php bug727_2.php bug727_3.php      \
               bug727_4.php bug967.php weeksinmonth_4_monday.txt      \
```

```
        weeksinmonth_4_sunday.txt weeksinmonth_rdm_monday.txt  \
        weeksinmonth_rdm_sunday.txt
DOCS=       TODO
_DOCSDIR=    .

.include <bsd.port.pre.mk>
.include "${PORTSDIR}/devel/pear/bsd.pear.mk"
.include <bsd.port.post.mk>
```

6.13. 使用 Python

Ports 套件支持同时并行安装多个不同的 Python 版本。 Ports 应确保能够根据用户配置的 PYTHON_VERSION 变量使用正确的 python 解释器。 一般说来， 这是通过将脚本中的 python 路径名替换为 PYTHON_CMD 变量的值来实现的。

在 PYTHON_SITELIBDIR 下安装文件的 ports 应在包名上使用 pyXY- 前缀， 以便明示它们将会配合哪个 Python 版本使用。

```
PKGNAMEPREFIX= ${PYTHON_PKGNAMEPREFIX}
```

表 23. 对用到 Python 的 port 最有用的一些变量

USE_PYTHON	此 port 需要 Python。 可以用 2.3+ 这样的形式来指定所希望的版本。 除此之外， 也可以用横线来分隔两个版本号， 以表示某个范围的版本， 例如： 2.1-2.3
USE_PYDISTUTILS	使用 Python distutils 来完成配置、 编译和安装。 对包含 setup.py 的 port 而言这是必需的。 它会自动覆盖默认的 do-build 以及 do-install 这两个 target。 如未定义 GNU_CONFIGURE， 它还会改变 do-configure。
PYTHON_PKGNAMEPREFIX	作为 PKGNAMEPREFIX 来区分不同 Python 版本的 package。 例如： py24-
PYTHON_SITELIBDIR	全站 package 所在的目录， 它包括了 Python 的安装目录 (通常是 LOCALBASE)。 在安装 Python 模块时， PYTHON_SITELIBDIR 变量会非常有用。
PYTHONPREFIX_SITELIBDIR	去掉了 PREFIX 部分的 PYTHON_SITELIBDIR。 应尽可能在 pkg-plist 中使用 %%PYTHON_SITELIBDIR%%。 %%PYTHON_SITELIBDIR%% 的默认值是 lib/python%%PYTHON_VERSION%%/site-packages
PYTHON_CMD	Python 解释器的命令行， 包括版本号。
PYNUMERIC	将数值处理扩展模块加入依赖关系。
PYNUMPY	对新增的数值计算扩展, numpy的依赖。 (PYNUMERIC 目前已被作者淘汰)。
PYXML	将 XML 扩展模块加入依赖关系。 (对于 Python 2.0 和更高版本不再需要， 因为它已经成为了标准组件)。

USE_TWISTED	将 twistedCore 加入依赖关系。 也可以用这个变量指定所需的组件， 例如： web lore pair flow
USE_ZOPE	加入对 Zope， 一种 web 应用平台的依赖。 这会把 Python 依赖改为 Python 2.3。 此外 ZOPEBASEDIR 也会自动设为 Zope 安装目录的位置。

完整的可用变量列表， 可以在 /usr/ports/Mk/bsd.python.mk 中找到。

6.14. 使用 Tcl/Tk

Ports 套件支持同时安装多个 Tcl/Tk 版本。 Ports 应至少支持默认的 Tcl/Tk 版本， 以及通过 USE_TCL 和 USE_TK 变量指定的更高版本。 希望使用的 tcl 版本， 则可以通过 WITH_TCL_VER 变量来使用。

表 24. 用到 Tcl/Tk 的 port 可以使用的变量

USE_TCL	表示 port 依赖于 Tcl 函数库 (不是 shell)。 可以指定需要的最低版本， 例如 84+。 不支持的版本， 可以在 INVALID_TCL_VER 变量中逐个指定。
USE_TCL_BUILD	表示 port 在联编过程中需要使用 Tcl。
USE_TCL_WRAPPER	需要使用 Tcl shell 而不需要特定版本的 tclsh 的 port 可以使用这个新变量。 系统中会安装 tclsh wrapper， 用户则可以指定所希望的 tcl shell。
WITH_TCL_VER	由用户定义的、 希望使用的 Tcl 版本。
UNIQUENAME_WITH_TCL_VER	和 WITH_TCL_VER 类似， 但是针对 port 指定的。
USE_TCL_THREADS	需要包含线程支持的 Tcl/Tk。
USE_TK	表示 port 依赖于 Tk 库 (不是 wish shell)。 它同时会隐含将 USE_TCL 设置为相同的值。 更多的描述， 请参考 USE_TCL 变量。
USE_TK_BUILD	与 USE_TCL_BUILD 变量表达类似的含义。
USE_TK_WRAPPER	与 USE_TCL_WRAPPER 变量表达类似的含义。
WITH_TK_VER	表达与 WITH_TCL_VER 变量类似的含义， 它同时会隐含将 WITH_TCL_VER 设置为相同的值。

可用的变量的完整列表， 可以在 /usr/ports/Mk/bsd.tcl.mk 中找到。

6.15. 使用 Emacs

本节尚有待撰写。

6.16. 使用 Ruby

表 25. 使用 Ruby 的 port 可以使用的变量

变量	说明
USE_RUBY	此 port 需要 Ruby。
USE_RUBY_EXTCONF	此 port 使用 extconf.rb 来完成配置。
USE_RUBY_SETUP	此 port 使用 setup.rb 来完成配置。
RUBY_SETUP	将此变量名设置为所用的 setup.rb 的文件名。 通常会是 install.rb。

下表展示了 ports 系统提供给 port 作者的一些变量。 您应使用这些变量， 以便把文件装到合适的位置。

请尽可能多地在 pkg-plist 中使用它们。 这些变量不应在 port 中重新定义。

表 26. 使用 Ruby 的 port 中的一些可用的只读变量

变量	说明	示范值
RUBY_PKGNAMEPREFIX	作为 PKGNAMEPREFIX 以区分用于不同 Ruby 版本的 package。	ruby18-
RUBY_VERSION	x.y.z 形式的完整 ruby 版本。	1.8.2
RUBY_SITELIBDIR	平台无关库的安装路径。	/usr/local/lib/ruby/site_ruby/1.8
RUBY_SITEARCHLIBDIR	平台相关的库的安装路径。	/usr/local/lib/ruby/site_ruby/1.8/amd64-freebsd6
RUBY_MODDOCDIR	模块文档的安装路径。	/usr/local/shared/doc/ruby18/patsy
RUBY_MODEXAMPLESDIR	模块用例的安装路径。	/usr/local/shared/examples/ruby18/patsy

可用变量的完整列表， 可以在 /usr/ports/Mk/bsd.ruby.mk 中找到。

6.17. 使用 SDL

变量 USE_SDL 可以用于自动配置 port 的依赖关系， 以适应使用类似 devel/sdl12 和 x11-toolkits/sdl_gui 这些依赖 SDL 的库的情形。

目前系统能够识别下列 SDL 库：

- sdl: devel/sdl12
- gfx: graphics/sdl_gfx
- gui: x11-toolkits/sdl_gui
- image: graphics/sdl_image
- ldbad: devel/sdl_ldbad
- mixer: audio/sdl_mixer
- mm: devel/sdlmm
- net: net/sdl_net
- sound: audio/sdl_sound
- ttf: graphics/sdl_ttf

因此， 如果 port 需要依赖 net/sdl_net 和 audio/sdl_mixer， 则对应的写法将是：

```
USE_SDL=    net mixer
```

同时， net/sdl_net 和 audio/sdl_mixer 所依赖的 devel/sdl12 也会被自动地加入。

加入您使用 USE_SDL， 它将自动地：

- 将对于 sdl12-config 的依赖关系加入到 BUILD_DEPENDS
- 将变量 SDL_CONFIG 加入到 CONFIGURE_ENV
- 将对所选的库的依赖， 加入到 LIB_DEPENDS

要检查某个特定的 SDL 库是否可用， 可以通过 WANT_SDL 变量来达到目的：

```
WANT_SDL=yes

.include <bsd.port.pre.mk>

.if ${HAVE_SDL:Mmixer}!=""
USE_SDL+=  mixer
.endif

.include <bsd.port.post.mk>
```

6.18. 使用 wxWidgets

这一节介绍了在 ports tree 中的 wxWidgets 库的现状， 以及它与 ports 系统的集成。

6.18.1. 介绍

许多不同版本的 wxWidgets 库之间是存在相互冲突的 (它们会安装同名的文件) 在 ports 系统中，
这一问题是通过将不同的版本以包含版本号后缀的名字安装来解决的。

这样做的一个最明显的缺点是， 应用程序必须进行修改， 才能找到所希望的版本。 幸运的是，
多数应用程序会调用 wx-config 脚本来确定需要的编译器和连接器选项。
这个脚本会随可用的版本不同而有不同的名字。 主要的应用程序都会尊重环境变量的配置， 或提供一个 configure 参数， 用以指定调用哪个 wx-config。 如果不是这样的话， 就需要对应用程序打补丁了。

6.18.2. 版本的选择

为了让您的 port 使用指定版本的 wxWidgets， 可以定义两个变量 (如果只定义了一个，
则另一个会取默认值)：

表 27. 用于选择 wxWidgets 版本的变量

变量	说明	默认值
USE_WX	列出这个 port 能使用的版本	全部版本
USE_WX_NOT	列出这个 port 不能使用的版本	无

下面是可用的 wxWidgets 版本， 以及对应的 ports：

表 28. 可用的 wxWidgets versions

版本	Port
2.4	x11-toolkits/wxgtk24
2.6	x11-toolkits/wxgtk26
2.8	x11-toolkits/wxgtk28



从 2.5 版开始， 也提供了对应的 Unicode 版本， 这种版本可以通过 slave port 安装，
与普通版本相比， 它会多一个 -unicode 后缀， 不过这可以通过使用变量来处理 (请参见 Unicode)。

在 用于选择 wxWidgets 版本的变量 中的变量， 可以设为下列值或由空格分隔的组合：

表 29. wxWidgets 版本

说明	例子
单个版本	2.4
某版本以上版本	2.4+
某版本以下版本	2.6-
某段版本 (版本号较小的必须在前)	2.4-2.6

除此之外， 还有一些用以从可用的本那本中选择所希望的版本的变量。 这种变量也可以设为一组版本，而靠前的版本的优先级更高。

表 30. 用于选择希望的版本的 wxWidgets versions

变量名	用于
WANT_WX_VER	port
WITH_WX_VER	用户

6.18.3. 选择组件

也有一些其他应用， 尽管它们本身并不是 wxWidgets 库， 但却与之相关。 这些应用程序可以在 WX_COMPS 变量中使用， 以下是可用的组件：

表 31. 可用的 wxWidgets 组件

名称	说明	版本限制
wx	主库	无
contrib	第三方库	无
python	wxPython (Python 绑定)	2.4-2.6
mozilla	wxMozilla	2.4
svg	wxSVG	2.6

您可以为每个依赖的组件， 通过冒号分隔的后缀指定其类型。 如果没有指定， 则会使用默认的依赖类型 (参见 默认的 wxWidgets 依赖关系类型)。 下面是可用的类型：

表 32. 可用的 wxWidgets 依赖类型

名称	说明
build	联编时需要该组件， 相当于 BUILD_DEPENDS
run	运行时需要该组件， 相当于 RUN_DEPENDS
lib	联编和运行时均需要该组件， 相当于 LIB_DEPENDS

组件的默认依赖关系类型， 如下表所示：

表 33. 默认的 wxWidgets 依赖关系类型

组件	依赖关系类型
wx	lib
contrib	lib
python	run
mozilla	lib
svg	lib

例 17. 选择 wxWidgets 组件

下面的片段展示了使用 wxWidgets 版本 2.4 及第三方库的方法。

```
USE_WX=    2.4
WX_COMPS=    wx contrib
```

6.18.4. Unicode

wxWidgets 库从其 2.5 版开始支持 Unicode 了。 在 ports 系统中， 这两种版本均有提供，并可以通过下列变量来选择：

表 34. 用以在 Unicode 版本的 wxWidgets 的变量

变量	说明	作用
WX_UNICODE	该 port 只能 配合 Unicode 版本使用	port
WANT_UNICODE	port 能够与两种版本配合使用，但希望使用 Unicode 版本	port
WITH_UNICODE	令 port 使用 Unicode 版本	用户
WITHOUT_UNICODE	令 port 使用普通版本，如果支持的话 (即未定义 WX_UNICODE)	用户



如果 port 同时支持 Unicode 和普通版本， 请不要使用 WX_UNICODE。 如果希望默认启用 Unicode， 应定义 WANT_UNICODE。

6.18.5. 检测已安装的版本

要检测系统中安装的版本， 就需要定义 WANT_WX。 如果没有将其设置为特定的版本，则组件将包含版本后缀。 HAVE_WX 变量在检测完成后会自动填入内容。

例 18. 检测已安装的 wxWidgets 版本和组件

下面的片段可以在安装 wxWidgets 的系统中令 port 使用它， 反之则作为一项选项提供。

```
WANT_WX=    yes

.include <bsd.port.pre.mk>

.if defined(WITH_WX) || ${HAVE_WX:Mwx-2.4} != ""
USE_WX=    2.4
CONFIGURE_ARGS+=--enable-wx
.endif
```

下面的片段在系统中有安装过时启用 wxPython 支持， 或在没有安装时作为选项提供； 对 wxWidgets 也是如此办理， 版本皆为 2.6。

```
USE_WX=    2.6
WX_COMPS=    wx
WANT_WX=    2.6
```

```
.include <bsd.port.pre.mk>

.if defined(WITH_WXPYTHON) || ${HAVE_WX:Mpython} != ""
WX_COMPS+=     python
CONFIGURE_ARGS+=--enable-wxpython
.endif
```

6.18.6. 定义的变量

以下是一些可以在 port 中使用的变量 (这之前需要定义 用于选择 wxWidgets 版本的变量 中的至少一个变量)。

表 35. 为使用 wxWidgets 的 port 定义的变量

变量名	说明
WX_CONFIG	到 wxWidgets wx-config 脚本的路径 (名字会随版本不同而不同)
WXRC_CMD	到 wxWidgets wxrc 程序的路径 (名字会随版本不同而不同)
WX_VERSION	将要用到的 wxWidgets 版本 (例如， 2.6)
WX_UNICODE	如果没有定义， 而将会使用 Unicode 时， 系统将自动定义此变量。

6.18.7. 在 bsd.port.pre.mk 中进行处理

如果您需要在引用了 bsd.port.pre.mk 之后立即对一些变量进行处理， 则需要定义 WX_PREMK。



如果定义了 WX_PREMK， 则在此 之后 定义的依赖关系、 组件和变量将不会生效， 您在引用 bsd.port.pre.mk 之前的 wxWidgets port 变量将直接起作用。

例 19. 在命令中使用 wxWidgets 变量

下面的片段以执行 wx-config 脚本来得到完整的版本号， 将其赋值到变量中， 并传递给一个程序举例说明了 WX_PREMK 的用法。

```
USE_WX=      2.4
WX_PREMK=    yes

.include <bsd.port.pre.mk>

.if exists(${WX_CONFIG})
VER_STR!=    ${WX_CONFIG} --release

PLIST_SUB+=  VERSION="${VER_STR}"
.endif
```



在 target 中的 wxWidgets 变量可以直接使用， 而无需 WX_PREMK 的参与。

6.18.8. 额外的 configure 参数

某些 GNU configure 脚本在只设置了 WX_CONFIG 环境变量时， 无法自动找到 wxWidgets，
而需要使用额外的参数来加以指定。 您可以使用 WX_CONF_ARGS 变量来给出这些参数。

表 36. 可用于 WX_CONF_ARGS 的值

可用值	结果
absolute	--with-wx-config=${WX_CONFIG}
relative	--with-wx=${LOCALBASE} --with-wx-config=${WX_CONFIG:T}

6.19. 使用 Lua

这一节描述了在 ports 系统中的 Lua 库的现状， 以及它与 ports 系统的集成。

6.19.1. 介绍

许多不同版本的 Lua 库和相关的解释器之间是相互冲突的 (它们会安装同名的文件)。 在 ports 系统中，
这一问题是通过将不同版本的文件以不同的版本号作为后缀名解决的。

这样做最大的一个问题是， 每个程序都需要进行修改才能找到它所需要的版本。 不过，
通过将适当的参数传给编译器和连接器很容易解决这个问题。

6.19.2. 选择版本

要让您的 port 使用指定版本的 Lua， 可以定义两个变量的值 (如果只定义了其中的一个，
则另一个会使用默认值)：

表 37. 用于选择 Lua 版本的变量

变量	说明	默认值
USE_LUA	port 能够使用的 Lua 版本列表	全部可用版本
USE_LUA_NOT	与 port 不兼容的版本列表	无

下面是目前 ports 系统提供的可用 Lua 版本和对应的目录：

表 38. 可用的 Lua 版本

版本	Port
4.0	lang/lua4
5.0	lang/lua50
5.1	lang/lua

在 用于选择 Lua 版本的变量 中的变量， 可以设置为下面的版本之一， 或用空格分隔的若干版本：

表 39. 指定 Lua 版本

说明	例子
一个版本	4.0
某个版本或更高版本	5.0+
不高于某个版本	5.0-
版本范围 (低版本必须在前)	5.0-5.1

除此之外， 也有一些用来从可用版本中选择推荐版本的其它变量。 这些变量也可以设置为一组版本，
而前面的版本优先级较高。

表 40. 用于选择推荐 Lua 版本的变量

变量名	用于
WANT_LUA_VER	port
WITH_LUA_VER	用户

例 20. 选择 Lua 版本

下面是一个用到 Lua 版本 5.0 或 5.1， 并默认使用 5.0 的 port 的片段。 这个默认值可以通过 WITH_LUA_VER 来另外指定。

```
USE_LUA=    5.0-5.1
WANT_LUA_VER= 5.0
```

6.19.3. 组件的选择

也有一些其它的应用， 尽管本身并不是 Lua 库， 但却与它们相关。 这些应用可以通过 LUA_COMPS 变量来指定。 可用的组件如下：

表 41. 可用的 Lua 组件

名字	说明	版本限制
lua	主库	无
tolua	用于访问 C/C++ 代码的库	4.0-5.0
ruby	Ruby 绑定	4.0-5.0



还有一些其它的组件， 但这些组件是由解释器， 而不是由应用程序使用的 (也就是不被其它模块使用)。

每个组件的依赖关系类型可以通过手工添加分隔符为冒号的后缀来指定。 如果不指定， 则会采用默认类型 (请参见 默认的 Lua 依赖关系类型)。 以下是可用的依赖关系类型：

表 42. 可用的 Lua 依赖关系类型

名字	说明
build	这个组件是联编过程所必需的， 相当于 BUILD_DEPENDS
run	在运行时需要这个组件， 相当于 RUN_DEPENDS
lib	这个组件在联编和运行时都需要， 相当于 LIB_DEPENDS

组件的默认依赖关系类型如下：

表 43. 默认的 Lua 依赖关系类型

组件	依赖关系类型
lua	对于 4.0-5.0 是 lib (动态连接) 而对于 5.1 则是 build (静态连接)
tolua	build (静态连接)
ruby	lib (动态连接)

例 21. 选择 Lua 组件

下面是一个使用了 Lua 版本 4.0 及其 Ruby 绑定的 port 片段。

```
USE_LUA=    4.0
LUA_COMPS=  lua ruby
```

6.19.4. 检测系统中已安装的版本

要检测系统中已安装的版本， 您必须定义 WANT_LUA。 如果没有将其设定为具体的版本，则组件会包含版本后缀。 在检测之后， HAVE_LUA 变量将设为检测到的版本。

例 22. 检测已安装的 Lua 版本和组件

下面是一个如果系统中有安装 Lua 或选择了选项时使用它的 port 片段。

```
WANT_LUA=     yes

.include <bsd.port.pre.mk>

.if defined(WITH_LUA5) || ${HAVE_LUA:Mlua-5.[01]} != ""
USE_LUA=      5.0-5.1
CONFIGURE_ARGS+=--enable-lua5
.endif
```

下面的这段 port 在系统中已经有安装， 或用户选择了 tolua 和 Lua 支持时加以安装， 版本均选择 4.0。

```
USE_LUA=      4.0
LUA_COMPS=    lua
WANT_LUA=     4.0

.include <bsd.port.pre.mk>

.if defined(WITH_TOLUA) || ${HAVE_LUA:Mtolua} != ""
LUA_COMPS+=   tolua
CONFIGURE_ARGS+=--enable-tolua
.endif
```

6.19.5. 定义的变量

在 port 中可以使用下列变量 (在定义了 用于选择 Lua 版本的变量 中至少一个变量之后)。

表 44. 为用到 Lua 的 port 定义的变量

变量名	说明
LUA_VER	将要使用的 Lua 版本。 (例如， 5.1)
LUA_VER_SH	Lua 动态连接库的主版本 (例如， 1)
LUA_VER_STR	不带点的 Lua 版本 (例如， 51)
LUA_PREFIX	安装 Lua (及其组件) 使用的后缀
LUA_SUBDIR	在 ${PREFIX}/bin、 ${PREFIX}/share 和 ${PREFIX}/lib 中用于安装 Lua 的子目录
LUA_INCDIR	用以安装 Lua 和 tolua 头文件的目录
LUA_LIBDIR	用以安装 Lua 和 tolua 库文件的目录
LUA_MODLIBDIR	用以安装 Lua 模块动态连接库 (.so) 的目录
LUA_MODSHAREDIR	用以安装 Lua 模块 (.lua) 的目录
LUA_PKGNAMEPREFIX	Lua 模块包的后缀名
LUA_CMD	到 Lua 解释器的路径
LUAC_CMD	到 Lua 编译器的路径
TOLUA_CMD	到 tolua 程序的路径

例 23. 告诉 port 到什么地方去找 Lua

下面的 port 片段展示了如何告诉使用的 configure 脚本去什么地方查找 Lua 的头文件和库文件。

```
USE_LUA=      4.0
GNU_CONFIGURE= yes
CONFIGURE_ENV= CPPFLAGS="-I${LUA_INCDIR}" LDFLAGS="-L${LUA_LIBDIR}"
```

6.19.6. 在 bsd.port.pre.mk 时进行处理

如果您需要在使用引用 bsd.port.pre.mk 之后就得到变量， 以便将其用于执行一些命令， 需要定义 LUA_PREMK。



如果您定义了 LUA_PREMK， 则在您引用 bsd.port.pre.mk之后， 即使修改了 Lua port 变量， 版本和依赖关系也都不会随之发生变化了。

例 24. 在命令中使用 Lua 变量

下面的片段展示了如何利用 LUA_PREMK， 并运行 Lua 解释器得到完整的版本串， 将其赋值给一个变量， 并传递给程序。

```
USE_LUA=      5.0
LUA_PREMK=    yes

.include <bsd.port.pre.mk>

.if exists(${LUA_CMD})
VER_STR!=     ${LUA_CMD} -v
```

```
CFLAGS+=    -DLUA_VERSION_STRING="${VER_STR}"
.endif
```



在 target 中的 Lua 变量可以在命令中安全的使用， 而无需使用 LUA_PREMK。

6.20. 使用 Xfce

USE_XFCE 变量可以用来自动配置使用基于 Xfce 库或应用程序， 如 x11-toolkits/libxfce4gui 和 x11-wm/xfce4-panel 的 port 的依赖关系。

目前， 系统能够识别下列 Xfce 库和应用程序：

- libexo：x11/libexo
- libgui：x11-toolkits/libxfce4gui
- libutil：x11/libxfce4util
- libmcs：x11/libxfce4mcs
- mcsmanager：sysutils/xfce4-mcs-manager
- panel：x11-wm/xfce4-panel
- thunar：x11-fm/thunar
- wm：x11-wm/xfce4-wm
- xfdev：dev/xfce4-dev-tools

除此之外， 还能够使用下列参数：

- configenv： 如果您的 port 需要使用特殊的 CONFIGURE_ENV 来查找所需的库。

```
-I${LOCALBASE}/include -L${LOCALBASE}/lib
```

会加到 CONFIGURE_ENV 的 CPPFLAGS。

因此， 如果 port 有到 sysutils/xfce4-mcs-manager 的依赖关系， 并需要在 configure 的环境中指定特殊的饿 CPPFLAGS， 则所用的语法为：

```
USE_XFCE=    mcsmanager configenv
```

6.21. 使用 Mozilla

表 45. 用到 Mozilla 的 port 使用的变量

USE_GECKO	port 支持的 Gecko 后端。 可选值： libxul (libxul.so)、 seamonkey (libgtkembedmoz.so，过时， 新 port 应避免使用)。
USE_FIREFOX	port 需要使用 Firefox 作为运行环境依赖。 可选值： yes (使用默认版本)、 40、 36、 35。 默认的依赖采用的是版本 40。
USE_FIREFOX_BUILD	port 需要使用 Firefox 作为联编环境依赖。 可选值： 参见 USE_FIREFOX。 这个变量会自动设置 USE_FIREFOX 使用相同的值。

USE_SEAMONKEY	port 需要使用 SeaMonkey 作为运行环境依赖。 可选值： yes (使用默认版本)、 20、 11 (过时， 新 port 应避免使用)。 默认的依赖采用的是版本 20。
USE_SEAMONKEY_BUILD	port 需要使用 SeaMonkey 作为联编环境依赖。 可选值： 参见 USE_SEAMONKEY。 这个变量会自动设置 USE_SEAMONKEY 使用相同的值。
USE_THUNDERBIRD	port 需要使用 Thunderbird 作为运行环境依赖。 可选值： yes (使用默认版本)、 31、 30 (过时， 新 port 应避免使用)。 默认的依赖采用的是版本 31。
USE_THUNDERBIRD_BUILD	port 需要使用 Thunderbird 作为联编环境依赖。 可选值： 参见 USE_THUNDERBIRD。 这个变量会自动设置 USE_THUNDERBIRD 使用相同的值。

可用变量的完整列表， 请参阅 /usr/ports/Mk/bsd.gecko.mk。

6.22. 使用数据库

表 46. ports 中有关数据库的变量

Variable	Means
USE_BDB	如果这个变量为 yes， 则把 databases/db41 列为依赖关系。 这个变量还可以被设置成的值有： 40，41， 42、 43、 44、 46、 47、 48 或 51。 您可以声明可接受值的范围， USE_BDB=42+ 将寻找已安装的最高版本， 如果没有找到则退回到 42。
USE_MYSQL	如果这个变量为 yes， 则把 databases/mysql55-server 列为依赖关系。还有一个相关的变量，WANT_MYSQL_VER, 可以设置的值有 323，40，41，50，51，52， 55 或者 60。
USE_PGSQL	如果设置成 yes，则把 databases/postgresql82 列为依赖关系。还有一个相关的变量，WANT_PGSQL_VER， 可以设置的值有 73，74，80，81，82，83 或 90。

更多详情请参阅 bsd.database.mk。

6.23. 启动和停止服务 (rc 脚本)

rc.d 脚本在系统启动时用于启动服务， 并为管理员提供停止、 启动和重新启动某个服务的标准方法。 Ports 安装的脚本会集成到系统的 rc.d 框架中。 关于如何使用它的说明， 可以在 使用手册的 rc.d 章节 找到。 关于可用命令的详细解释， 则可以在 rc(8) 和 rc.subr(8) 找到。 最后， 您可以参阅 这篇文章 了解撰写 rc.d 脚本的最佳实践。

可以安装一或多个 rc.d 脚本：

```
USE_RC_SUBR=  doormand
```

这些脚本必须放到 files 目录， 并附加 .in。 这个文件中可以使用标准的 SUB_LIST 替换展开。 除此之外， 我们还强烈推荐使用 %%PREFIX%% 和 %%LOCALBASE%% 替换展开。 关于 SUB_LIST 的介绍可以在 本书的相关章节 找到。

在 FreeBSD 6.1-RELEASE 之前， 与 rcorder(8) 的集成是通过 USE_RCORDER 而不是 USE_RC_SUBR

来完成的。 不过， 除非 port 需要提供安装进基本系统这样的选项， 或者服务需要在 rc.d 脚本 FILESYSTEMS 之前运行这类特殊情况， 一般来说是不需要使用这个功能的。

从 FreeBSD 6.1-RELEASE 开始， 本地安装的 rc.d 脚本 (包括由 port 安装的脚本) 会纳入基本系统的 rcorder(8)。

以下是一个简单的 rc.d 脚本：

```
#!/bin/sh

# $FreeBSD$
#
# PROVIDE: doormand
# REQUIRE: LOGIN
# KEYWORD: shutdown
#
# 在 /etc/rc.conf.local 或 /etc/rc.conf 中增加下述设置可以启用这一服务：
#
# doormand_enable (bool):  默认设为 NO。
#           设为 YES 可以启用 doormand。
# doormand_config (path):  默认设为 %%PREFIX%%/etc/doormand/doormand.cf。
#

. /etc/rc.subr

name="doormand"
rcvar=${name}_enable

command=%%PREFIX%%/sbin/${name}
pidfile=/var/run/${name}.pid

load_rc_config $name

: ${doormand_enable="NO"}
: ${doormand_config="%%PREFIX%%/etc/doormand/doormand.cf"}

command_args="-p $pidfile -f $doormand_config"

run_rc_command "$1"
```

除非有很站得住脚的理由提前启动服务，所有的 ports 脚本应使用

```
REQUIRE: LOGIN
```

。 如果服务需要以特定用户 (除 root 之外) 身份启动， 则必须这样做。 在前面的例子中， 我们还使用了

```
KEYWORD: shutdown
```

以便让 mythical port 在系统停机的过程中以正常的方式终止，因为它需要在系统引导过程中启动服务。
如果脚本没有启动任何服务，则并不需要这样做。

这里，对变量的默认赋值方法应采用 "="，而非 ":=" 这样的形式。这是因为，
前一种赋值方法只有在变量未被设置时才设置默认值，而后一种方法则会在变量没有设置，或者
其值为空时都设置默认值。用户非常可能在其 rc.conf.local 中使用类似

```
doormand_flags=""
```

这样的设置，而采用 ":=" 来进行赋值，则会在不经意间覆盖用户所希望的设置。



新增的脚本均不应使用 .sh 后缀。未来，仍然包含这一后缀的脚本将被批量改名。

6.23.1. 卸载时停止服务

可以在卸载的过程中自动地停止服务。我们建议只有在绝对必要，
例如必须在删除文件之前停止服务这类的情况下才使用这一功能。通常来说，
决定是否在卸载时停止服务是系统管理员需要考虑的事情。另外要注意，这个功能也会影响升级过程。

需要时可以在 pkg-plist 中加入：

```
@stopdaemon doormand
```

这里的参数必须与 USE_RC_SUBR 变量的内容匹配。

6.24. 添加用户和用户组

一些 port 需要在安装的系统中创建特定的用户或用户组。如果有这种情况，请从 50 到 999
之间选择一个尚未使用的 UID，并在 ports/UIDs (针对用户) 或 ports/GIDs (针对组) 中予以记录。
请务必确保您没有使用系统中已经在其他 ports 中使用的 UID。

如果您的 port 需要创建新用户或用户组，请在提交补丁的时候一并提交这两个文件的补丁。

接下来，可以在您的 Makefile 中使用 USERS 和 GROUPS 这两个变量，
系统会在安装时自动创建用户或组。

```
USERS=  pulse
GROUPS= pulse pulse-access pulse-rt
```

现有的保留 UID 和 GID 列表，可以在 ports/UIDs 和 ports/GIDs 找到。

6.25. 依赖内核源代码的 Ports

某些 ports (例如可加载式内核模块) 需要内核的源文件才能编译。
下面是检测用户是否安装了源代码的例子：

```
.if !exists(${SRC_BASE}/sys/Makefile)
IGNORE=       requires kernel sources to be installed
```

```
.endif
```

Chapter 7. 高级 pkg-plist 用法

7.1. 根据 make 变量对 pkg-plist 进行修改

某些 port， 特别是 p5- port， 会需要根据配置选项 (或对于 p5- port 而言， perl 的版本) 来修改它们的 pkg-plist。 为简化这一工作， 在 pkg-plist 中的 %%OSREL%%、 %%PERL_VER%%， 以及 %%PERL_VERSION%% 将自动进行相应的替换。 其中， %%OSREL%% 的值是操作系统以数值表示的版本 (例如 4.9)。 %%PERL_VERSION%% 和 %%PERL_VER%% 是 perl 的完整版本号 (例如 5.8.9)。许多其它与 port 文档文件有关的 %%变量%% 在 相应章节 中进行了介绍。

如果您还需要进行其它的替换， 可以通过将 PLIST_SUB 变量设置为一组 变量=值 对来实现。 其中， %%VAR%% 表示在 pkg-plist 中将被 值 替换的那些文字。

举例来说， 如果 port 需要把很多文件放到和版本有关的目录中， 可以在 Makefile 中按照类似下面的例子：

```
OCTAVE_VERSION= 2.0.13
PLIST_SUB=    OCTAVE_VERSION=${OCTAVE_VERSION}
```

并在 pkg-plist 中将具体的版本替换为 %%OCTAVE_VERSION%%。 这样， 在升级 port 时， 就不需要再到 pkg-plist 中修改那几十 (或者， 有时甚至是上百) 行的内容了。

如果您的 port 需要根据一定的配置来有条件地安装一些文件， 通常的做法是在 pkg-plist 中列出这些文件时， 在对应行的开头加上 %%TAG%%， 并将 TAG 写到 Makefile 中的 PLIST_SUB 变量中， 根据需要替换掉， 或替换为 @comment， 后者表示让打包工具忽略这行：

```
.if defined(WITH_X11)
PLIST_SUB+= X11=""
.else
PLIST_SUB+= X11="@comment "
.endif
```

与之对应， 在 pkg-plist 中：

```
%%X11%%bin/foo-gui
```

这一替换过程 (以及加入 联机手册 的过程)， 会在 pre-install 和 do-install 两个 target 之间， 通过读取 PLIST 并写入 TMPPLIST (默认情况下， 是： WRKDIR/.PLIST.mktmp) 来完成。 因此， 如果您的 port 动态生成 PLIST， 就需要在 pre-install 之前完成。 另外， 如果您的 port 需要编辑所生成的文件， 则需要在 post-install 中操作名为 TMPPLIST 的那个文件。

另一种可行的修改装箱单的方法， 则是根据 PLIST_FILES 和 PLIST_DIRS 这两个变量的设置来进行。 它们的值会作为目录名连同 PLIST 的内容一起写入 TMPPLIST。 在 PLIST_FILES 和 PLIST_DIRS 中列出的名字， 会经历前面所介绍的 %%变量%% 替换过程。 除此之外， 在 PLIST_FILES 中列出的文件， 会不加任何修改第出现在最终的装箱单中， 而 @dirrm 将作为前缀加到 PLIST_DIRS 所列的名字之前。 为了达到目的， PLIST_FILES 和 PLIST_DIRS 必须在写 TMPPLIST 之前， 也就是在 pre-install 或更早的阶段进行设置。

7.2. 空目录

7.2.1. 清理空目录

一定要让 port 在卸载时进行清理空目录。 通常， 可以通过为所有由 port 创建的目录增加对应的 @dirrm 行来实现。 在删除父目录之前， 需要首先删除它的子目录。

```
:
lib/X11/oneko/pixmaps/cat.xpm
lib/X11/oneko/sounds/cat.au
:
@dirrm lib/X11/oneko/pixmaps
@dirrm lib/X11/oneko/sounds
@dirrm lib/X11/oneko
```

然而， 有时 @dirrm 会由于其它 port 使用了同一个目录而发生错误。 利用 @dirrmtry 可以只删除那些空目录， 而避免给出警告。

```
@dirrmtry share/doc/gimp
```

按照上面的写法， 将不会显示任何错误信息， 而且，即使在 ${PREFIX}/shared/doc/gimp 由于其它 port 在其中安装了一些别的文件的时候， 也不会导致 pkg_delete(1) 异常退出。

7.2.2. 如何建立空目录

在 port 安装过程中创建的空目录需要特别留意。 安装 package 时并不会自动创建这些目录， 这是因为 package 只保存文件。 要确保安装 package 时会自动创建这些空目录， 需要在 pkg-plist 中加入与 @dirrm 对应的行：

```
@exec mkdir -p %D/shared/foo/templates
```

7.3. 配置文件

如果 port 需要把一些文件放到 PREFIX/etc， 不要 简单地安装它们， 并将其列入 pkg-plist， 因为这样会导致 pkg_delete(1) 删除用户精心编辑的文件， 而新安装时则又会把这些文件覆盖。

因此， 您应把配置文件的例子按其它的后缀来安装 (例如 filename.sample 就是一个不错的选择) 并显示一条 消息 告诉用户如何复制并编辑这个配置文件， 以便让软件能够正确工作。

因此， 应按其它的后缀来安装配置文件的例子 (filename.sample 就是一个不错的选择)。 如果实际的配置文件不存在， 则将其复制为实际文件的名字。 卸载时， 如果发现用户没有修改配置文件， 则将其删除。 您需要在 port 的 Makefile， 以及 pkg-plist (对于从 package 安装的情形) 进行处理。

示例的 Makefile 部分：

```
post-install:
  @if [ ! -f ${PREFIX}/etc/orbit.conf ]; then \
    ${CP} -p ${PREFIX}/etc/orbit.conf.sample ${PREFIX}/etc/orbit.conf ; \
  fi
```

示例的 pkg-plist 部分：

```
@unexec if cmp -s %D/etc/orbit.conf.sample %D/etc/orbit.conf; then rm -f
%D/etc/orbit.conf; fi
etc/orbit.conf.sample
@exec if [ ! -f %D/etc/orbit.conf ] ; then cp -p %D/%F %B/orbit.conf; fi
```

另外， 还应显示一条 消息 指出用户应在何处复制并编辑这个文件， 以便让软件能开始正常工作。

7.4. 动态装箱单与静态装箱单的对比

静态装箱单 是指在 Ports Collection 中以 pkg-plist 文件 (可能包含变量替换)， 或以 PLIST_FILES 和 PLIST_DIRS 的形式嵌入到 Makefile 出现的装箱单。 即使它是由工具或 Makefile 中的某个 target 在经由 committer 加入到 Ports Collection 之前 自动生成的也是如此，
因为可以在不下载或编译源代码包的前提下对其进行检视。

动态装箱单 是指在 port 编译并安装时生成的装箱单。 在下载并编译您所移植的应用程序的源代码之前，或在执行了 make clean 之后， 就无法查看其内容了。

尽管使用动态装箱单并不被禁止， 但监护人应尽可能使用静态装箱单， 因为它能够让用户使用 grep(1)来发现所需的 ports， 例如， 它是否会安装某个特定文件。 动态列表主要应用于复杂的，
其装箱单随所选功能会发生巨变 (因而使得维护静态装箱单不再可行)， 或那些随版本而改变装箱单内容的 port (例如， 使用 Javadoc 来生成文档的那些 ports)。

我们鼓励那些选择使用动态装箱单的监护人提供一个能够生成 pkg-plist 的 target，
以便于用户检视其内容。

7.5. 装箱单 (package list) 的自动化制作

首先， 请确认已经基本上完成了 port 的工作， 仅缺 pkg-plist。

接下来， 建立一个用于安装您的 port 的临时目录， 并在其中安装它所依赖的所有其他软件包：

```
# mkdir /var/tmp/`make -V PORTNAME`
# mtree -U -f `make -V MTREE_FILE` -d -e -p /var/tmp/`make -V PORTNAME`
# make depends PREFIX=/var/tmp/`make -V PORTNAME`
```

将目录结构保存到一新文件中。

```
# (cd /var/tmp/`make -V PORTNAME` && find -d * -type d) | sort > OLD-DIRS
```

建立一空白 pkg-plist 文件：

```
# :>pkg-plist
```

如果您的 port 遵循 PREFIX (应该如此) 则接下来应安装该 port 并创建装箱单。

```
# make install PREFIX=/var/tmp/`make -V PORTNAME`
# (cd /var/tmp/`make -V PORTNAME` && find -d * \! -type d) | sort > pkg-plist
```

此外还应把新建立的目录加入装箱单。

```
# (cd /var/tmp/`make -V PORTNAME` && find -d * -type d) | sort | comm -13 OLD-DIRS - |
sort -r | sed -e 's#^#@dirrm #' >> pkg-plist
```

最后需要手工整理 packing list； 这一过程不是 完全 自动的。 联机手册应列入 port 的 Makefile 中的 MANn， 而不是装箱单。 用户配置文件应被删除， 或以 filename.sample 这样的名字来安装。 info/dir 文件， 也不应列入， 同时应按照 info 文件 的说明来增加一些 install-info 行。 所有由 port 安装的库， 应按照 动态连接库 小节中介绍的方法处理。

另外， 也可以使用 /usr/ports/Tools/scripts/ 中的 plist 脚本来自动创建 package list。 plist 脚本是一个 Ruby 脚本， 它能够将前面介绍的手工操作自动化。

开始的步骤和上面的前三行一样， 也就是 mkdir， mtree 并 make depends。 然后联编和安装 port：

```
# make install PREFIX=/var/tmp/`make -V PORTNAME`
```

然后让 plist 生成 pkg-plist 文件：

```
# /usr/ports/Tools/scripts/plist -Md -m `make -V MTREE_FILE` /var/tmp/`make -V
PORTNAME` > pkg-plist
```

与前面类似， 如此生成的装箱单也需要手工进行一些清理工作。

另一个可以用来创建最初的 pkg-plist 的工具是 ports-mgmt/genplist。 和其他自动化工具类似， 您应对它生成的 pkg-plist 应手工检查并根据需要进行修改。

Chapter 8. pkg-* 文件

前面有一些没有提及的关于 pkg-* 文件的技巧， 它们可以方便地完成许多任务。

8.1. pkg-message (安装预编译包时显示的消息文件)

如果您需要在安装时显示一条消息给用户， 可以把这消息放在 pkg-message 中。 这一特性通常可以用于在 pkg_add(1) 之后显示一些附加的安装步骤， 或显示关于授权的信息。

当需要显示一些编译开关或警告时， 请使用 ECHO_MSG。 pkg-message 文件只是为显示安装后的执行操作指导使用的。 类似地， 还需要留意 ECHO_MSG 和 ECHO_CMD 之间的区别。 前一个是在屏幕上显示消息性的文字， 而后一个则用于在执行命令时使用管道。

下面是用到了这两个宏的例子 shells/bash2/Makefile：

```
update-etc-shells:
  @${ECHO_MSG} "updating /etc/shells"
  @${CP} /etc/shells /etc/shells.bak
  @( ${GREP} -v ${PREFIX}/bin/bash /etc/shells.bak; \
    ${ECHO_CMD} ${PREFIX}/bin/bash) >/etc/shells
  @${RM} /etc/shells.bak
```



pkg-message 文件， 并不需要明确地加到 pkg-plist 中。 此外， 在用户使用 port 而不是 package 来安装软件时， 它并不会被显示出来。 因此如果需要的话， 您应该在 post-install target 中指定显示它。

8.2. pkg-install (安装预编译包时执行的脚本文件)

如果您的 port 需要在预编译的安装包通过 pkg_add(1) 安装时执行一些命令， 则应通过 pkg-install 脚本来完成。 这个脚本会自动地加入 package， 并被 pkg_add(1) 执行两次： 第一次是 ${SH} pkg-install ${PKGNAME} PRE-INSTALL 而第二次是 ${SH} pkg-install ${PKGNAME} POST-INSTALL。 $2 可被用来检测脚本运行的模式。 环境变量 PKG_PREFIX 将设置为 package 的安装目录。 请参见 pkg_add(1) 以了解更进一步的细节。



在使用 make install 时这个脚本不会被自动运行。 如果需要运行它， 则必须在您的 port 中的 Makefile 里明确地予以调用， 其方法是加入类似 PKG_PREFIX=${PREFIX} ${SH} ${PKGINSTALL} ${PKGNAME} PRE-INSTALL 这样的命令。

8.3. pkg-deinstall (卸载时执行的脚本文件)

这一脚本将在 package 被卸载时执行。

此脚本会被 pkg_delete(1) 执行两次。 第一次是 ${SH} pkg-deinstall ${PKGNAME} DEINSTALL 而第二次则是 ${SH} pkg-deinstall ${PKGNAME} POST-DEINSTALL。

8.4. pkg-req (安装预编译包时检测是否应执行操作的脚本文件)

如果您的 port 需要确定它是否应被安装， 可以创建 pkg-req"requirements" 脚本。 它会在安装/卸载时自动运行， 以决定操作是否应被实施。

这个脚本会在使用 pkg_add(1) 安装时以 pkg-req ${PKGNAME} INSTALL 的命令行执行。 卸载时， 它将由 pkg_delete(1) 以 pkg-req ${PKGNAME} DEINSTALL 的命令行执行。

8.5. 改变 pkg-* 文件的名字

所有 pkg- 文件的名字， 皆系采用变量予以定义， 因此在需要时可以在您的 Makefile 中加以改变。
当您需要在多个 port 之间共享某些 pkg- 文件， 或需要写入某些文件时就非常有用了。 (参见 在 WRKDIR 以外的地方写文件， 以了解为什么直接将变更写入 pkg-* 子目录是个糟糕的主意)

下面是一组变量以及它们的默认值 (PKGDIR 默认情况下是 ${MASTERDIR}。)

变量	默认值
DESCR	${PKGDIR}/pkg-descr
PLIST	${PKGDIR}/pkg-plist
PKGINSTALL	${PKGDIR}/pkg-install
PKGDEINSTALL	${PKGDIR}/pkg-deinstall
PKGREQ	${PKGDIR}/pkg-req
PKGMESSAGE	${PKGDIR}/pkg-message

请修改这些变量， 而不是直接覆盖 PKG_ARGS 的值。 如果您改变了 PKG_ARGS， 这些文件将无法在安装 port 时正确地复制到 /var/db/pkg 目录。

8.6. 使用 SUB_FILES 和 SUB_LIST

SUB_FILES 和 SUB_LIST 这两个变量可以用来在 port 文件中使用某些动态的值， 例如 pkg-message 中的 installation PREFIX。

用 SUB_FILES 变量， 可以指定需要自动进行修改的文件列表。 在 SUB_FILES 中的每一个 文件， 在 FILESDIR 目录中都必须有一个对应的 文件.in。 修改后的版本将保存在 WRKDIR。 在 USE_RC_SUBR (或已经过时的 USE_RCORDER) 中定义的文件会自动加入到 SUB_FILES 中。 对于 pkg-message、 pkg-install、 pkg-deinstall and pkg-req， 对应的 Makefile 变量会被自动设置， 以指向处理过的版本。

SUB_LIST 这个变量的内容是一系列 VAR=VALUE 对。 SUB_FILES 所列出的文件中所有的 %%VAR%% 都将被替换为 VALUE。 系统自动定义了一些常用的替换对， 包括： PREFIX、 LOCALBASE、 DATADIR、 DOCSDIR， 以及 EXAMPLESDIR。 替换结果中所有以 @comment 开头的行，
都将在变量替换之后被删去。

下面的例子中， 将把 pkg-message 中的 %%ARCH%% 替换为系统所运行的架构名称：

```
SUB_FILES=    pkg-message
SUB_LIST=     ARCH=${ARCH}
```

注意， 在上述例子中， FILESDIR 里必须有 pkg-message.in 这个文件。

下面是一个正确的 pkg-message.in 例子：

```
Now it is time to configure this package.
Copy %%PREFIX%%/shared/examples/putsy/%%ARCH%%.conf into your home directory
as .putsy.conf and edit it.
```

Chapter 9. 测试您的 port

9.1. 运行 make describe

许多 FreeBSD port 维护工具， 例如 portupgrade(1)， 会依赖于一个名为 /usr/ports/INDEX 的数据库的正确性， 它提供了关于 port 的相关信息， 例如依赖关系等等。 INDEX 是由顶级的 ports/Makefile 通过 make index 来建立的， 这个命令会进入每一个 port 的子目录， 并在那里执行 make describe。 因此， 如果某个 port 的 make describe 失败， 就没有人能生成 INDEX， 人们很快会变得不高兴。



无论在 make.conf 中设置了什么选项， 这个文件都应能够正确地生成。 因此， 应避免在 (例如) 某个依赖关系无法满足时使用 .error。 (参见 避免使用 .error 结构。)

如果 make describe 只是产生一个字符串， 而不是错误信息， 可能就没什么问题。 请参见 bsd.port.mk 以了解所生成的串的意义。

最后要说明的是， 新版本的 portlint (在下一节中将进行介绍) 将会自动地运行 make describe。

9.1.1. Portlint

在提交或 commit 之前， 应使用 portlint 来进行检查。 portlint 会对常见的、 包括功能上的和格式上的错误给出警告。 对于新的 (或在 repocopy 代码库中复制的) port， portlint -A 可以完成全面检查； 对于暨存的 port， portlint -C 一般就足够了。

由于 portlint 采用启发式方法来检查错误， 有时它会产生误警。 另外， 有时由于 port 框架的限制可能没有办法修正它指出的问题。 如果您有疑虑， 请写信询问 FreeBSD ports 邮件列表。

9.1.2. 使用 Port Tools 来完成测试

在 Ports 套件中， 提供了一个 ports-mgmt/porttools 程序。

port 是一个能够帮助您简化测试工具的前端脚本。 如果希望对新增的 port 或更新 port 时进行测试， 可以用 port test 来完成这些测试工作， 这也包含了 portlint 检查。 这个命令会检测并列出没有在 pkg-plist 中列出的文件。 具体用法请参见下面的例子：

```
# port test /usr/ports/net/csup
```

9.1.3. PREFIX (安装时的顶级目录名) 和 DESTDIR

PREFIX 能够决定 port 安装时的目的位置。 一般情况下这个位置是 /usr/local 或 /opt， 但也可以设为其它的任意值。 您的 port 则必须遵循这个变量。

除此之外， 如果用户配置了 DESTDIR， 则表示希望将 port 安装到另一个环境， 通常是 jail 或在 / 以外的其他位置挂接的系统中。 实际上， port 会安装到 DESTDIR/PREFIX， 并注册到位于 DESTDIR/var/db/pkg 的预编译包数据库中。 由于 DESTDIR 是由 ports 框架藉由 chroot(8) 来实现的， 您在撰写符合 DESTDIR 规范的 ports 时并不需要什么额外的工作。

一般而言 PREFIX 会设为 LOCALBASE_REL (默认是 /usr/local)。 如果设置了 USE_LINUX_PREFIX， 则 PREFIX 会设为 LINUXBASE_REL (默认是 /compat/linux)。

避免将 /usr/local 或 /usr/X11R6 硬编码到源代码中， 能够大大提高 port 的灵活性， 并适应不同环境的需要。 对于使用 imake 的 X port， 这一工作是自动完成的； 其他情况下， 通常可以简单地将 port 所用到的 Makefile 脚本中出现的 /usr/local (或对于没有使用 imake 的 X port 而言， /usr/X11R6) 替换为读取 ${PREFIX} 变量就能达到目的了， 因为这个变量在联编和安装的过程中， 会自动向下传递。

一定要避免让您的 port 在 /usr/local 而不是正确的 PREFIX 中安装文件。 简单的测试方法是：

```
# make clean; make package PREFIX=/var/tmp/`make -V PORTNAME`
```

如果有文件安装到了 PREFIX 以外的地方， 打包过程将抱怨找不到这些文件。

这一步骤并不能帮助发现内部引用， 或纠正在引用其它 port 中的文件时使用的 LOCALBASE。 您需要在 /var/tmp/make -V PORTNAME 中测试安装好的软件， 才能够达到这样的目的。

您可以在自己的 Makefile 中改变 PREFIX 变量的值， 也可以通过用户环境变量来影响它。 然而， 一般情况下决不应该在 Makefile 中明确设置它的值。

此外， 引用其它 port 中的文件时， 应使用前面介绍的变量， 而不要直接指定它们的路径名。 例如， 如果您的 port 需要使用 PAGER 这个宏来指明 less 的完整路径， 应使用下面的编译选项：

```
-DPAGER=\"${LOCALBASE}/bin/less\"
```

而非 -DPAGER=\"/usr/local/bin/less\"。 这种方法能够增加在系统管理员把整个 /usr/local 目录挪到其它位置时安装成功的机会。

9.1.4. Tinderbox

如果您是非常热心的 ports 参与者， 则可以看看 Tinderbox。 这是一个强大的用于联编和测试 ports 的系统， 它基于 Pointyhat 的脚本。 您可以使用 ports-mgmt/tinderbox port 来安装 Tinderbox。 请一定仔细阅读随它安装的文档， 因为配置并不简单。

请访问 Tinderbox 网站 以了解进一步的细节。

Chapter 10. 升级一个 port

如果您发现某个 port 相对原作者所发布的版本已经过时， 则首先需要确认的是您的 port 是最新的。
您可以在 FreeBSD FTP 镜像的 ports/ports-current 目录中找到它们。 但是， 如果您正在使用较多的 port， 则可能使用 CVSup 来保持 Ports Collection 最新更为简单， 这在 使用手册 中进行了介绍。 此外，
这样做也有助于保持 port 依赖关系的正确性。

下一步是检查是否已经有在等待的更新。 要完成这项工作， 可以采用下列两种方法之一。 有一个用于搜索 FreeBSD 问题报告 (PR) 数据库 (也被称作 GNATS)。 在下拉框中选择 ports， 然后输入 port 的名字。

但是， 有些时候人们会忘记将避免混淆的 port 的名字放到 Synopsis 字段中。 这种时候， 您可以试试看 FreeBSD Ports 监视系统 (也被叫做 portsmon)。 这个系统会尝试按照 port 的名字来进行分类。
要搜索和某个特定 port 有关的 PR， 可以使用 port概览。

如果没有候审的 PR， 下一步是给 port 的维护者写信， 这可以通过执行 make maintainer 看到。
这个人可能正在进行升级工作， 或者由于某种理由暂时没有升级 (例如， 新版本有稳定性问题)；
一般您不希望重复他们的工作。 注意没有维护者的 port 的维护者会显示为 ports@FreeBSD.org，
这是一般性 port 问题的邮件列表， 因此发邮件给它一般没什么意义。

如果维护者要求您去完成升级， 或者没有维护者， 您就有机会通过自行完成升级来帮助 FreeBSD 了！
请使用基本系统提供的 diff(1) 命令来完成相关的工作。

如果只修改一个文件， 可以直接使用 diff 来生成补丁， 将需要修改的文件复制成 something.orig，
并将改动放进 something， 接着生成补丁：

```
% /usr/bin/diff something.orig something > something.diff
```

如果不是这样的话， 则您应使用 cvs diff 的方法 (使用 CVS 制作补丁)， 或将目录整个复制到另一个目录，
并使用 diff(1) 比较两个目录时在目录中递归产生的输出结果 (例如， 如果您修改后的 port 目录的名字是 superedit 而原始文件的目录是 superedit.bak， 则应保存 diff -ruN superedit.bak superedit 的结果)。
一致式 (unified) 或 上下文式 (context) diff 都是可以的， 但一般来说 port committer 会更喜欢一致式 diff。 请注意这里使用的选项 -N， 它的目的是强制 diff 正确地处理出现新文件， 或老文件被删除的情形。
在把 diff 发给我们之前， 请再次检查输出， 以便确认每一个修改都是有意义的。 (特别注意，
在对比目录之前要用 make clean 清理一下)。

为了简化常用的补丁文件操作， 您可以使用 /usr/ports/Tools/scripts/patchtool.py。 使用之前，
请首先阅读 /usr/ports/Tools/scripts/README.patchtool。

如果 port 目前还无人维护， 而且您自己经常使用它， 请考虑自荐为它的维护者。 FreeBSD 有超过 4000 个没有维护者的 port， 而这正是最需要志愿人员的领域。 (要了解关于维护者的任务描述， 请参见
开发手册中的相关部分。)

将 diff 发送给我们的最佳方式是通过 send-pr(1) (category 一栏写 ports)。 如果您正维护那个 port，
请务必在 synopsis 的开头写上 [maintainer update]， 并将您的 PR 的 "Class" 设置为 maintainer-update。 反之， 您的 PR 的 "Class" 就应该是 change-request。
请在信中逐个提及每一个删除或增加的文件， 因为这些都必须明确地在使用 cvs(1) 进行 commit
时明确地指定。 如果 diff 超过了 20K， 请考虑压缩并对其进行 uuencode； 否则， 简单地将其原样加入 PR 即可。

在您 send-pr(1) 之前， 请再次阅读 Problem Reports 一文中的 如何撰写问题报告 小节；
它给出了丰富的关于如何撰写更好的问题报告的介绍。



如果您的更新是由于安全考虑， 或修复已经 commit 的 port 中的严重问题， 请通知 Ports 管理团队 <portmgr@FreeBSD.org> 来申请立即重建和分发您的 port 的 package。 否则， 不愿怀疑的使用 pkg_add(1) 的用户，
可能会在未来数周之内继续通过使用 pkg_add -r 安装旧版本。



再次强调， 请使用 diff(1) 而非 shar(1) 来发送现有 port 的更新！ 这可以帮助 ports committer 理解需要修改的内容。

现在您已经了解了所需的所有操作， 您可能会像要阅读在 保持同步 中关于如何保持最新的描述。

10.1. 使用 CVS 制作补丁

如果可能的话， 请提交cvs(1) diff； 这种情形要比直接比较 "新、旧" 目录要容易处理。 此外，
这种方法也让您更容易看出到底改了什么， 并在其他人更新了 Ports Collection 时容易合并这些改动，
在提交之前， 这可以减少维护补丁所需的工作。

```
% cd ~/my_wrkdir ①
% cvs -d R_CVSROOT co pdnsd ② ③
% cd ~/my_wrkdir/pdnsd
```

① 当然， 这可以是您指定的任意目录； 联编 port 并不局限于 /usr/ports/ 的子目录。

② R_CVSROOT 是任何一个公共的 cvs 镜像服务器， 您可以在 FreeBSD 使用手册 中挑选一个。

③ pdnsd 是 port 的模块名字； 通常说来它和 port 的名字一样， 不过也有些例外， 特别是那些本地化类别
(german/selfhtml 对应的模块名字是 de-selfhtml)； 您可以通过 cvsweb 界面 查询，
或者也可以指定完整路径， 例如在我们这个例子中是 ports/dns/pdnsd。

在工作目录中， 您可以像往常一样进行任何更改。 如果您添加或删除了文件， 则需要告诉 cvs
来追踪这些改动：

```
% cvs add new_file
% cvs remove deleted_file
```

请反复检查 测试 port 列出的事项并使用 用 portlint 来检查 port 进行检查。

```
% cvs status
% cvs update ①
```

① 这会合并 CVS 中其他人做的改动和您的补丁； 在这个过程中， 您需要仔细观察输出。
文件名前面的那个字母会显示做了什么， 请参阅 cvs update 文件名前字母前缀的含义 中给出的说明。

表 47. cvs update 文件名前字母前缀的含义

U	文件更新无误。
P	文件更新无误 (通常只有在使用远程代码库时才会看到)。
M	文件有本地修改， 并合并成功而未产生任何冲突。
C	文件有本地修改， 进行了合并并产生了冲突。

如果您在执行 cvs update 时某些文件出现了 C， 则说明有其他人在 CVS 中做了修改， 而 cvs(1)
无法将这些改动与您本地的改动进行合并。 不过， 无论如何， 最好都检查一下合并的结果， 因为 cvs
并不知道 port 应该是什么样子， 因此它所做的合并无论是否产生了冲突， 都有可能 (并且并不罕见)
产生没有意义的结果。

最后一步是以 CVS 中的文件为基础生成 unified diff(1)：

```
% cvs diff -uN > ../`basename ${PWD}`.diff
```



指定 -N 十分重要， 因为它确保了添加或删除的文件也出现在补丁中。
补丁将包含删除的文件， 在打上补丁时， 这些文件会被清空， 所以最好在 PR 中提醒

> committer 删除它们。

根据 升级一个 port 的指导提交您的补丁。

10.2. UPDATING 和 MOVED 文件

如果在升级 port 时需要类似修改配置文件或运行特定的程序这样的特别步骤， 则应在 /usr/ports/UPDATING 文件中予以说明。 这个文件中的项目格式如下：

```
YYYYMMDD:
  AFFECTS: users of port类别/port名字
  AUTHOR: 您的名字 <您的电子邮件地址>

  所需执行的特别步骤
```

如果您需要在内文中加入具体的 portmaster 或 portupgrade 的说明， 请确保所用的 shell 命令使用了正确的转义字符。

如果 port 被删除或改名，则应在 /usr/ports/MOVED 中添加相应的说明项目。 这个文件中的项目格式如下：

```
原来的名字|新名字 (如果删除则应留空)|删除或改名的日期|原因
```

Chapter 11. Ports 的安全

11.1. 安全为何如此重要

软件中偶尔会引入 bug。 毋庸置疑， 安全漏洞是最为危险的。 从技术角度看，
这些漏洞可以通过消除导致它们的 bug 来修复。 然而， 处理一般的 bug 和安全漏洞的策略是截然不同的。

典型的小 bug 通常只影响那些启用了某些能够触发它的选项组合的用户。
开发人员最终会在发布没有那个问题的新版之后给出一个补丁来修正它， 而用户中的主体并不会立即升级，
因为他们并没有因存在问题而感到苦恼。 严重的 bug 可能会导致数据丢失和其它问题， 无论如何，
谨慎的用户知道， 除了软件 bug 之外还有很多其它事故可能会导致数据丢失， 因此他们会备份重要数据；
此外， 严重的 bug 通常会被很快发现。

安全漏洞则完全不同。 第一， 它们可能存在数年而不被发现， 因为它们可能并不导致软件无法正常工作。
第二， 通过利用漏洞， 恶意的一方可能会得到未获授权的访问权限，
并利用这些权限毁掉或修改敏感数据； 而更糟糕的情况则是用户可能根本注意不到损害已经发生。 第三，
暴露出安全漏洞的系统， 往往能够帮助攻击者闯入其它之前不可能进入的系统。 因此，
只是修正安全漏洞是不够的： 必须以清晰和全面的方式通知公众， 这样他们就能够评估风险，
并采取适当的措施。

11.2. 修复安全漏洞

当说起 port 或 package 时， 安全漏洞往往是出现在原作者的发行包， 或移植过程中加入的文件里。
对于前一种情况， 软件的原作者通常会立刻发布一个补丁甚至新版， 您只需要按照原作者的修正去更新
port 就可以了。 如果由于某种原因修正被延误， 则要么 将 port 标记为 FORBIDDEN， 要么在 port
中加入一个自己的补丁。 如果有存在漏洞的 port， 尽可能尽快修复其漏洞就是。 无论是哪种情况，
您还是需要按照 标准的提交流程 提交， 除非您有直接在 ports tree 上 commit 的权限。



作为 ports committer 并不能够随便 commit 所有 port。 请注意通常 port 都有维护者，
而他们应得到您的尊重。

在漏洞被修正之后， 一定要同时增加 port 的修订版本号。 这样， 规律性地升级安装的 package
的用户就能够看到他们需要进行升级。 另外， 还应联编预编译的安装包， 并通过 FTP 和 WWW 镜像发布，
以取代有漏洞的版本。 注意要增加 PORTREVISION 数字， 除非在修正问题时 PORTVERSION
发生了变化。 一般来说， 如果在 port 中增加了补丁文件， 就应该增加 PORTREVISION，
但例外的例子是您已经将软件升级到了最新版， 因为这时已经改掉 PORTVERSION 了。 请参见 相关小节
以了解进一步的信息。

11.3. 通知整个用户群体

11.3.1. VuXML 数据库

当发现了安全漏洞时的一项重要而紧迫的步骤， 就是让使用 port 的用户群了解其危险。
这类通知有两重目的。 首先， 如果危害真的很严重， 可能理性的办法就是立即应用一项缓解措施， 例如，
停止受到影响的服务， 甚至完全删除 port， 直到问题被修正为止。 其次，
许多用户只是偶尔升级所安装的软件包， 通过通知， 他们能够知道已经到了 必须 更新软件的时候，
因为已经有了修正这些问题的版本了。

由于现有的 port 数量极其庞大， 为每一个问题都发布安全公告，
毫无疑问地会发表和狼来了一样多的安全公告， 并增大受众在真的发生严重的问题时忽略问题的可能。
因此， 在 port 中发现的安全漏洞， 会在 FreeBSD VuXML 数据库 中记录。
安全官团队成员会持续地追踪这个数据库的修改， 以了解需要他们注意的内容。

如果您是 committer， 则可以自行更新 VuXML 数据库。 这样， 您就能够同时帮助安全官团队，
并尽早将至关重要的信息传达给用户群体。 然而， 如果您不是 committer，
或者您相信自己发现了一个异常严重的漏洞， 请不要犹豫， 按照 FreeBSD 安全信息
页面上的方法联系安全官团队。

正如其名称所暗示的那样， VuXML 数据库是一个 XML 文档。 其源文件 vuln.xml 被保存在 security/vuxml

port 的目录中。 所以， 它的全名是 PORTSDIR/security/vuxml/vuln.xml。 每当您发现 port 中的安全漏洞时， 请把新的纪录加入到那个文件中。 在熟悉 VuXML 之前， 您最好先看看是否有类似的您发现的问题的其它记录， 并复制它作为模板。

11.3.2. VuXML 简介

XML 是一个复杂的语言， 它远远超越了这本书的范围。 不过， 只需了解标记的命名规则， 就能 VuXML 记录的结构有一个大体的了解了。 XML 标记的名字应出现在一对尖括号之间。 每一个 <tag> 必须有一个对应的 </tag>。 标记可以嵌套， 如果嵌套的话， 内层的标记必须在外层标记之前结束。 这就形成了一个标记的层次结构， 也就是关于它们之间如何嵌套的规则。 听起来很像 HTML， 是不是? 最主要的区别在于， XML 是可扩展的 (eXtensible)， 例如通过定义新的标记等等。 由于其结构的内在性质， XML 能够赋予无组织的数据新的形态。 VuXML 是专门为描述安全漏洞设计的语言。

现在让我们来观察一个实际的 VuXML 记录：

```
<vuln vid="f4bc80f4-da62-11d8-90ea-0004ac98a7b9"> ①
 <topic>Several vulnerabilities found in Foo</topic> ②
 <affects>
  <package>
   <name>foo</name> ③
   <name>foo-devel</name>
   <name>ja-foo</name>
   <range><ge>1.6</ge><lt>1.9</lt></range> ④
   <range><ge>2.*</ge><lt>2.4_1</lt></range>
   <range><eq>3.0b1</eq></range>
  </package>
  <package>
   <name>openfoo</name> ⑤
   <range><lt>1.10_7</lt></range> ⑥
   <range><ge>1.2,1</ge><lt>1.3_1,1</lt></range>
  </package>
 </affects>
 <description>
  <body xmlns="http://www.w3.org/1999/xhtml">
   <p>J. Random Hacker reports:</p> ⑦
   <blockquote
    cite="http://j.r.hacker.com/advisories/1">
    <p>Several issues in the Foo software may be exploited
     via carefully crafted QUUX requests.  These requests will
     permit the injection of Bar code, mumble theft, and the
     readability of the Foo administrator account.</p>
   </blockquote>
  </body>
 </description>
 <references> ⑧
  <freebsdsa>SA-10:75.foo</freebsdsa> ⑨
```

```
    <freebsdpr>ports/987654</freebsdpr> ⑩
    <cvename>CAN-2010-0201</cvename> ⑪
    <cvename>CAN-2010-0466</cvename>
    <bid>96298</bid> ⑫
    <certsa>CA-2010-99</certsa> ⑬
    <certvu>740169</certvu> ⑭
    <uscertsa>SA10-99A</uscertsa> ⑮
    <uscertta>SA10-99A</uscertta> ⑯
    <mlist
msgid="201075606@hacker.com">http://marc.theaimsgroup.com/?l=bugtraq&amp;m=20
3886607825605</mlist> ⑰
    <url>http://j.r.hacker.com/advisories/1</url> ⑱
  </references>
  <dates>
    <discovery>2010-05-25</discovery> ⑲
    <entry>2010-07-13</entry> ⑳
    <modified>2010-09-17</modified>
  </dates>
</vuln>
```

标记的名字都是简单明了的， 下面我们来介绍一下需要由您填写的字段：

① 这是 VuXML 记录的顶级 tag。 它有一个强制性的字段， vid， 用于为此记录 (它包含的部分) 指定一个全局唯一标识符 (UUID)。 您应为每一个新的 vuXML 生成新的 UUID (而且别忘了要把模板中的 UUID 换成新的， 如果您不是从头开始的话)。 您可以使用 uuidgen(1) 来生成 VuXML UUID。

② 关于问题的一句话描述。

③ 此处给出受到影响的 package 的名字。 可以给出多个名字， 因为可能有多个软件包基于同一个 master port 或软件产品。 这可能包括稳定和开发分支、 本地化版本， 以及提供了不同的编译时选项的 slave port。

④ 受影响的 package 版本， 可以使用 <lt>, <le>, <eq>, <ge>, 和 <gt> 表达成一个或多个版本及其范围。 注意给出的版本范围不应存在重叠。在描述范围的时候， * (星号) 表达最小的版本。 更具体地说， 2.* 小于 2.a。 因此， 星号可以用来匹配所有可能的 alpha、 beta， 以及 RC 版本。 例如， <ge>2.</ge><lt>3.</lt> 可以选择性地匹配每一个 2.x 的版本， 而 <ge>2.0</ge><lt>3.0</lt> 显然不能， 因为它会漏掉 2.r3 而匹配 3.b。上面的例子指定了受影响的版本， 是包括 1.6 到 1.9 上下界的所有版本， 以及 2.x 在 2.4_1 之前的版本， 和 3.0b1 版。

⑤ 受到影响的一组 package (本质上是 ports) 可以列在 <affected> 小节中。 如果多个软件产品都采用了同样的基础代码， (比如说 FooBar、 FreeBar 和 OpenBar) 而且包含同样的 bug 或漏洞。 请注意列出多个名字时， 应该在一个 <package> 小节中完成。

⑥ 如果可能， 版本的范围应包括 PORTEPOCH 和 PORTREVISION。 务必注意， 根据加权规则， 带有非零 PORTEPOCH 的版本， 系统会认为比没有 PORTEPOCH 的版本高， 例如 3.0,1 高于 3.1 甚至 8.9。

⑦ 关于问题的摘要性信息。 此处使用 XHTML。 务必要成对使用 和 。 可以使用较为复杂的标记， 但仅限于有助于让信息更准确和明了的修饰： 请不要过分地美化。

⑧ 这部分包含了相关的可供参考的文档。 请尽可能多提供参考文献。

⑨ 指定 FreeBSD 安全公告。

⑩ 指定 FreeBSD 问题报告。

⑪ 指定 Mitre CVE ID。

⑫ 指定 SecurityFocus Bug ID。

⑬ 指定 US-CERT 安全公告。

⑭ 指定 US-CERT 漏洞说明。

⑮ 指定 US-CERT 计算机安全警报。

⑯ 指定 US-CERT 技术性计算机安全警报。

⑰ 指向邮件列表存档的 URL。 属性 msgid 是可选项， 用以指定某一封信的 message ID。

⑱ 一般的 URL。 只有在没有其它更适合的参考文献时， 才应使用它。

⑲ 问题被全面披露的日期 (YYYY-MM-DD)。

⑳ 记录加入到数据库中的日期 (YYYY-MM-DD)。

记录最后一次被修改的日期 (YYYY-MM-DD)。 新记录不应包括这个字段。 只有在修改记录时才应加入它。

11.3.3. 测试您对 VuXML 数据库所作的修改

假定您打算撰写， 或已经写好了一个关于 package clamav 的问题描述， 并且， 已经知道 0.65_7 版本修正了这个问题。

您需要做的准备工作， 是 安装 一个新版本的 ports ports-mgmt/portaudit 程序、 ports-mgmt/portaudit-db， 以及 security/vuxml。



要运行 packaudit， 您必须拥有其 DATABASEDIR， 通常是 /var/db/portaudit 的写入权限。

您可以通过 DATABASEDIR 环境变量来指定一个不同的位置。

如果您的工作目录是 ${PORTSDIR}/security/vuxml 以外的其它地方， 则应使用环境变量 VUXMLDIR 来指明 vuln.xml 的位置。

首先， 检查一下是否已经有了关于这个漏洞的描述。 如果已经有过这样的记录， 那么它将匹配较早版本的 package， 0.65_6：

```
% packaudit
% portaudit clamav-0.65_6
```

如果什么都没有发现， 您就可以考虑写一个新的记录来描述这个漏洞了。 现在可以生成一个新的 UUID (假设它是 74a9541d-5d6c-11d8-80e3-0020ed76ef5a)， 然后将您的新记录加入到 VuXML 数据库中。 接下来， 用下面的命令来检查它是否符合语法：

```
% cd ${PORTSDIR}/security/vuxml && make validate
```



您需要安装下列 package 中的至少一个： textproc/libxml2、 textproc/jade。

接下来从 VuXML 文件重构 portaudit 数据库：

```
% packaudit
```

要验证您新加入的项的 <affected> 小节能够正确地匹配希望的 package， 可以使用下面的命令：

```
% portaudit -f /usr/ports/INDEX -r 74a9541d-5d6c-11d8-80e3-0020ed76ef5a
```



请参见 portaudit(1) 以了解关于这个命令语法的更多细节。

请确信新添加的记录不会在输出中匹配不应匹配的项。

现在检查您添加的记录所匹配的版本是否正确：

```
% portaudit clamav-0.65_6 clamav-0.65_7
Affected package: clamav-0.65_6 (matched by clamav<0.65_7)
Type of problem: clamav remote denial-of-service.
Reference: <http://www.freebsd.org/ports/portaudit/74a9541d-5d6c-11d8-80e3-
0020ed76ef5a.html>

1 problem(s) found.
```

显然， 前一个版本会匹配， 而后一个不会。

最后， 验证您从 VuXML 数据库中能够正确地得到预期的网页效果：

```
% mkdir -p ~/public_html/portaudit
% packaudit
% lynx ~/public_html/portaudit/74a9541d-5d6c-11d8-80e3-0020ed76ef5a.html
```

Chapter 12. 该做什么和不该做什么

12.1. 介绍

这里是一些在移植软件时可能会遇到的常见问题。 您应按照这个列表检查自己的 port， 同样地，
您也可以帮助检查 PR 数据库 中由其它人提交的 port。 请按照在 问题报告和一般性注释
中介绍的方法提交您的看法。 帮助检查 PR 数据库中的 ports 即能够帮助我们更快地 commit 它们，
也能证明您清楚地了解如何完成这些工作。

12.2. WRKDIR (联编时使用的临时目录)

任何时候都不要在 WRKDIR 以外的位置写文件。 WRKDIR 是在 port 联编过程中唯一的一处一定可写的地方
(参见 如何从 CDROM 安装 port 以了解从只读的目录中联编和安装 port 的例子)。 如果您需要改变 pkg-*
文件， 请按照 重新定义某个变量 介绍的方法， 而不是覆盖它们来实现。

12.3. WRKDIRPREFIX (用于联编的临时目录的父目录名)

一定要确保您的 port 尊重 WRKDIRPREFIX 的设置。 绝大多数 port 并不需要担心这个。 具体说来，
当引用其它 port 的 WRKDIR 时， 需要注意正确的位置应该是
WRKDIRPREFIXPORTSDIR/subdir/name/work 而不是 PORTSDIR/subdir/name/work 或
.CURDIR/../../subdir/name/work， 或别的什么。

另外， 如果您自行定义了 WRKDIR， 也要把 ${WRKDIRPREFIX}${.CURDIR} 放到前面。

12.4. 区分不同的操作系统， 以及 OS 的版本

在不同版本的 Unix 下可能需要对代码进行一些修改或增加少许编译选项， 才能够正确地编译和运行。
如果您需要根据一些条件来对代码进行修改， 请尽可能让这些修改通用， 这样，
我们就能够将这些代码移植回更早的 FreeBSD 系统， 并交叉移植到其它 BSD 系统， 例如来自 CSRG 的
4.4BSD， BSD/386， 386BSD， NetBSD 和 OpenBSD。

推荐的获得 4.3BSD/Reno (1990) 以及更新版本 BSD 代码版本号的方式， 是使用 sys/param.h 中所定义的
BSD 宏的值。 一般来说这个文件已经被引用了； 如果没有的话， 增加下述代码：

```
#if (defined(__unix__) || defined(unix)) && !defined(USG)
#include <sys/param.h>
#endif
```

到 .c 文件中合适的地方。 我们相信所有定义了这两个符号的系统中， 都提供了 sys/param.h。
如果您发现有不这样做的系统， 请通过致信 FreeBSD ports 邮件列表 让我们了解这一情况。

另一种方法是使用 GNU Autoconf 风格的方式：

```
#ifdef HAVE_SYS_PARAM_H
#include <sys/param.h>
#endif
```

采用这种方法时， 不要忘了把 -DHAVE_SYS_PARAM_H 加到 Makefile 中的 CFLAGS 里。

一旦引用了 sys/param.h， 您就可以使用：

```
#if (defined(BSD) && (BSD >= 199103))
```

来检测代码是否正在 4.3 Net2 代码基础， 或更新的系统上编译 (例如 FreeBSD 1.x， 4.3/Reno， NetBSD 0.9， 386BSD， BSD/386 1.1 以及更高版本)。

使用：

```
#if (defined(BSD) && (BSD >= 199306))
```

来检测代码是否正在 4.4 或更新的系统 (例如 FreeBSD 2.x， 4.4， NetBSD 1.0、 BSD/386 2.0 或更高版本)。

对于 4.4BSD-Lite2 代码系来说， BSD 宏的值应该是 199506。 这里只是作为信息提供，
您不应使用它来区分基于 4.4-Lite 的 FreeBSD 和基于 4.4-Lite2 的版本。 这些情况下， 您应使用 {freebsd} 宏。

保守地使用：

- {freebsd} 在所有版本的 FreeBSD 中皆有定义。 如果您正进行的修改 只 影响 FreeBSD，
  则应使用这个宏。 类似 sys_errlist[] 之于 strerror() 这样的移植问题是伯克利代码系公用的， 而并非 FreeBSD 所专有。
- 在 FreeBSD 2.x 中， {freebsd} 定义为 2。 更早版本中， 它曾经是 1。
  新的版本都会在主要的版本号变化时变更它。
- 如果您需要区分 FreeBSD 1.x 系统和 FreeBSD 2.x 及更高版本的区别， 通常应使用前述的 BSD 宏来进行。 如果事实上需要一个 FreeBSD 专有的修改 (例如， 在使用 ld 时需要特殊的共享库选项)，
  则可以用 {freebsd} 和 #if {freebsd} > 1 来检测 FreeBSD 2.x 和新系统上的变化。
  如果需要更细粒度地检测 FreeBSD 2.0-RELEASE 之后版本的变化， 则可以使用：

  ```
  #if __FreeBSD__ >= 2
  #include <osreldate.h>
  #   if __FreeBSD_version >= 199504
      /* 适用于 2.0.5+ 版本的代码 */
  #   endif
  #endif
  ```

在已有的数百个 port 中， 只有一两个应该使用 {freebsd}。 早期的 port
在不适当的地方使用了它并引发问题， 并不意味着您也必定如此。

12.5. __FreeBSD_version 值

下面是在 sys/param.h__FreeBSD_version 中定义的值及其意义的列表， 这里给出以方便您查阅：

表 48. __FreeBSD_version 值

值	日期	版本
119411		2.0-RELEASE
199501, 199503	March 19, 1995	2.1-CURRENT
199504	April 9, 1995	2.0.5-RELEASE
199508	August 26, 1995	2.1 之前的 2.2-CURRENT
199511	November 10, 1995	2.1.0-RELEASE

值	日期	版本
199512	November 10, 1995	2.1.5 之前的 2.2-CURRENT
199607	July 10, 1996	2.1.5-RELEASE
199608	July 12, 1996	2.1.6 之前的 2.2-CURRENT
199612	November 15, 1996	2.1.6-RELEASE
199612		2.1.7-RELEASE
220000	February 19, 1997	2.2-RELEASE
(not changed)		2.2.1-RELEASE
(无变化)		在 2.2.1-RELEASE 之后的 2.2-STABLE
221001	April 15, 1997	texinfo-3.9 之后的 2.2-STABLE
221002	April 30, 1997	top 之后的 2.2-STABLE
222000	May 16, 1997	2.2.2-RELEASE
222001	May 19, 1997	2.2.2-RELEASE 之后的 2.2-STABLE
225000	October 2, 1997	2.2.5-RELEASE
225001	November 20, 1997	2.2.5-RELEASE 之后的 2.2-STABLE
225002	December 27, 1997	合并 ldconfig -R 之后的 2.2-STABLE
226000	March 24, 1998	2.2.6-RELEASE
227000	July 21, 1998	2.2.7-RELEASE
227001	July 21, 1998	2.2.7-RELEASE 之后的 2.2-STABLE
227002	September 19, 1998	semctl(2) 修改之后的 2.2-STABLE
228000	November 29, 1998	2.2.8-RELEASE
228001	November 29, 1998	2.2.8-RELEASE 之后的 2.2-STABLE
300000	February 19, 1996	mount(2) 修改之前的 3.0-CURRENT
300001	September 24, 1997	mount(2) 修改之后的 3.0-CURRENT
300002	June 2, 1998	semctl(2) 修改之后的 3.0-CURRENT
300003	June 7, 1998	ioctl 参数变化之后的 3.0-CURRENT
300004	September 3, 1998	ELF 变换之后的 3.0-CURRENT
300005	October 16, 1998	3.0-RELEASE
300006	October 16, 1998	3.0-RELEASE 之后的 3.0-CURRENT
300007	January 22, 1999	3/4切分之后的 3.0-STABLE
310000	February 9, 1999	3.1-RELEASE
310001	March 27, 1999	3.1-RELEASE 之后的 3.1-STABLE

值	日期	版本
310002	April 14, 1999	C++ 构建/析构函数顺序变化之后的 3.1-STABLE
320000		3.2-RELEASE
320001	May 8, 1999	3.2-STABLE
320002	August 29, 1999	二进制不兼容的 IPFW 和 socket 变化之后的 3.2-STABLE
330000	September 2, 1999	3.3-RELEASE
330001	September 16, 1999	3.3-STABLE
330002	November 24, 1999	libc 中加入 mkstemp(3) 之后的 3.3-STABLE
340000	December 5, 1999	3.4-RELEASE
340001	December 17, 1999	3.4-STABLE
350000	June 20, 2000	3.5-RELEASE
350001	July 12, 2000	3.5-STABLE
400000	January 22, 1999	3/4切分之后的 4.0-CURRENT
400001	February 20, 1999	修改动态连接器处理方式之后的 4.0-CURRENT
400002	March 13, 1999	C++ 构建/析构函数顺序变化之后的
400003	March 27, 1999	提供 dladdr(3) 之后的 4.0- CURRENT
400004	April 5, 1999	修正了 __deregister_frame_info 的 4.0-CURRENT (也表示在 EGCS 1.1.2 集成之后的 4.0-CURRENT)
400005	April 27, 1999	suser(9) API 变化之后的 4.0- CURRENT (也表示 newbus 之后的 4.0-CURRENT)
400006	May 31, 1999	cdevsw 注册机制改变之后的 4.0- CURRENT
400007	June 17, 1999	加入了 socket 级凭据的 so_cred 之后的 4.0-CURRENT
400008	June 20, 1999	在 libc_r 中加入 poll 系统调用接口之后的 4.0- CURRENT
400009	July 20, 1999	将内核中 dev_t 类型改为 struct specinfo 指针之后的 4.0- CURRENT
400010	September 25, 1999	修正了一处 jail(2) 漏洞之后的 4.0- CURRENT
400011	September 29, 1999	sigset_t 数据类型改变之后的 4.0- CURRENT
400012	November 15, 1999	切换到 GCC 2.95.2 编译器之后的 4.0-CURRENT
400013	December 4, 1999	加入了可插的 linux模式 ioctl 处理程序后的 4.0-CURRENT
400014	January 18, 2000	引入 OpenSSL 之后的 4.0- CURRENT

值	日期	版本
400015	January 27, 2000	GCC 2.95.2 中 ABI 默认值从 -fvtable-thunks 改为 -fno-vtable-thunks 之后的 4.0-CURRENT
400016	February 27, 2000	引入 OpenSSH 之后的 4.0-CURRENT
400017	March 13, 2000	4.0-RELEASE
400018	March 17, 2000	4.0-RELEASE 之后的 4.0-STABLE
400019	May 5, 2000	引入延迟校验和之后的 4.0-STABLE
400020	June 4, 2000	将 libxpg4 的代码并入 libc 之后的 4.0-STABLE
400021	July 8, 2000	Binutils 升级到 2.10.0 之后的 4.0-STABLE， ELF 标志变化， 以及将 tcsh 引入基本系统
410000	July 14, 2000	4.1-RELEASE
410001	July 29, 2000	4.1-RELEASE 之后的 4.1-STABLE
410002	September 16, 2000	setproctitle(3) 从 libutil 移入 libc 之后的 4.1-STABLE
411000	September 25, 2000	4.1.1-RELEASE
411001		4.1.1-RELEASE 之后的 4.1.1-STABLE
420000	October 31, 2000	4.2-RELEASE
420001	January 10, 2001	合并 libgcc.a 和 libgcc_r.a，并修改了相关的 GCC 连接方式之后的 4.2-STABLE
430000	March 6, 2001	4.3-RELEASE
430001	May 18, 2001	引入 wint_t 之后的 4.3-STABLE
430002	July 22, 2001	PCI 电源状态 API 合并之后的 4.3-STABLE
440000	August 1, 2001	4.4-RELEASE
440001	October 23, 2001	引入 d_thread_t 之后的 4.4-STABLE
440002	November 4, 2001	mount 结构改变之后的 4.4-STABLE (影响文件系统 kld)
440003	December 18, 2001	用户态部分的 smbfs 被引入之后的 4.4-STABLE
450000	December 20, 2001	4.5-RELEASE
450001	February 24, 2002	usb 结构元素改名之后的 4.5-STABLE
450004	April 16, 2002	在 rc.conf(5) 变量 sendmail_enable 默认值改为 NONE 之后的 4.5-STABLE
450005	April 27, 2002	默认将 XFree86 4 用于预编译包联编之后的 4.5-STABLE
450006	May 1, 2002	accept 过滤器修正了安全问题并且不再会轻易被 DoS 之后的 4.5-STABLE

值	日期	版本
460000	June 21, 2002	4.6-RELEASE
460001	June 21, 2002	修正了 sendfile(2) 以吻合文档，而不再根据发出的头计算发出数据量之后的 4.6-STABLE
460002	July 19, 2002	4.6.2-RELEASE
460100	June 26, 2002	4.6-STABLE
460101	June 26, 2002	MFC sed -i 之后的 4.6-STABLE
460102	September 1, 2002	MFC 许多 pkg_install 新特性之后的 4.6-STABLE
470000	October 8, 2002	4.7-RELEASE
470100	October 9, 2002	4.7-STABLE
470101	November 10, 2002	开始生成 std{in,out,err}p 引用，而不是 sF。 这将 std{in,out,err} 从编译时表达式变成了运行时值。
470102	January 23, 2003	MFC mbuf 相关的将 m_aux mbuf 改为 m_tag 的修改之后的 4.7-STABLE
470103	February 14, 2003	OpenSSL 升级到 0.9.7 之后的 4.7-STABLE
480000	March 30, 2003	4.8-RELEASE
480100	April 5, 2003	4.8-STABLE
480101	May 22, 2003	realpath(3) 变为线程安全的之后的 4.8-STABLE
480102	August 10, 2003	对 twe 的 3ware API 修改之后的 4.8-STABLE
490000	October 27, 2003	4.9-RELEASE
490100	October 27, 2003	4.9-STABLE
490101	January 8, 2004	kinfo_eproc 中加入 e_sid 之后的 4.9-STABLE
490102	February 4, 2004	MFC rtld 的 libmap 功能之后的 4.9-STABLE
491000	May 25, 2004	4.10-RELEASE
491100	June 1, 2004	4.10-STABLE
491101	August 11, 2004	MFC 20040629 版本的包维护工具之后的 4.10-STABLE
491102	November 16, 2004	修正了 VM 当解除 wire 不存在页面时的问题之后的 4.10-STABLE
492000	December 17, 2004	4.11-RELEASE
492100	December 17, 2004	4.11-STABLE
492101	April 18, 2006	将 libdata/ldconfig 目录加入 mtree 文件之后的 4.11-STABLE。
500000	March 13, 2000	5.0-CURRENT

值	日期	版本
500001	April 18, 2000	加入 ELF 头字段， 并改变我们的 ELF 执行文件标记方式之后的 5.0-CURRENT
500002	May 2, 2000	kld 元数据修改之后的 5.0-CURRENT
500003	May 18, 2000	buf/bio 修改之后的 5.0-CURRENT
500004	May 26, 2000	binutils 升级后的 5.0-CURRENT
500005	June 3, 2000	将 libxpg4 并入 libc， 以及引入 TASKQ 之后的 5.0-CURRENT
500006	June 10, 2000	加入 AGP 接口之后的 5.0-CURRENT
500007	June 29, 2000	Perl 升级到 5.6.0 之后的 5.0-CURRENT
500008	July 7, 2000	KAME 代码升级到 2000/07 之后的 5.0-CURRENT
500009	July 14, 2000	ether_ifattach() 和 ether_detach() 修改之后的 5.0-CURRENT
500010	July 16, 2000	将 mtree 改为原先的默认值， 并使用 -L 来跟随符号连接之后的 5.0-CURRENT
500011	July 18, 2000	kqueue API 修改之后的 5.0-CURRENT
500012	September 2, 2000	setproctitle(3) 从 libutil 挪到 libc 之后的 5.0-CURRENT
500013	September 10, 2000	首个 SMPng commit 之后的 5.0-CURRENT
500014	January 4, 2001	<sys/select.h> 改为 <sys/selinfo.h> 之后的 5.0-CURRENT
500015	January 10, 2001	libgcc.a 和 libgcc_r.a 以及 GCC 连接方式变动之后的 5.0-CURRENT
500016	January 24, 2001	修改以允许 libc 和 libc_r 连接到一起， 不再鼓励使用 -pthread 之后的 5.0-CURRENT
500017	February 18, 2001	从 struct ucred 切换到 struct xucred 以便使内核为 mountd 等程序导出的 API 稳定下来之后的 5.0-CURRENT
500018	February 24, 2001	加入 CPUTYPE 用于 CPU 专用的优化的 make 变量之后的 5.0-CURRENT
500019	June 9, 2001	machine/ioctl_fd.h 改为 sys/fdcio.h 之后的 5.0-CURRENT
500020	June 15, 2001	locale 名称改变之后的 5.0-CURRENT
500021	June 22, 2001	引入 bzip2 之后的 5.0-CURRENT， 同时也代表删去了 S/Key

值	日期	版本
500022	July 12, 2001	加入 SSE 支持之后的 5.0-CURRENT
500023	September 14, 2001	KSE 第2个里程碑之后的 5.0-CURRENT
500024	October 1, 2001	d_thread_t 之后的 5.0-CURRENT， 同时 UUCP 被移入 ports
500025	October 4, 2001	64-位平台上的描述符和 creds API 变化之后的 5.0-CURRENT
500026	October 9, 2001	采用 XFree86 4 作为默认的预编译包， 以及加入 strnstr() libc 函数之后的 5.0-CURRENT
500027	October 10, 2001	加入 strcasestr() libc 函数之后的 5.0-CURRENT
500028	December 14, 2001	引入了用户态的 smbfs 组件之后的 5.0-CURRENT
(未予增加)		加入了新的 C99 指定位宽整形变量之后的 5.0-CURRENT
500029	January 29, 2002	修改了 sendfile(2) 的返回值之后的 5.0-CURRENT
500030	February 15, 2002	引入适合表达文件标志的 fflags_t 类型之后的 5.0-
500031	February 24, 2002	usb 结构元素改名之后的 5.0-CURRENT
500032	March 16, 2002	引入 Perl 5.6.1 之后的 5.0-CURRENT
500033	April 3, 2002	rc.conf(5) 变量 sendmail_enable 默认值改为 NONE 之后的 5.0-CURRENT
500034	April 30, 2002	mtx_init() 增加了第三个参数之后的 5.0-CURRENT
500035	May 13, 2002	包含 Gcc 3.1 的 5.0-CURRENT
500036	May 17, 2002	在 /usr/src 中删去了 Perl 的 5.0-CURRENT
500037	May 29, 2002	加入 dlfunc(3) 之后的 5.0-CURRENT
500038	July 24, 2002	一些 struct sockbuf 的成员变为结构， 并重新排列顺序之后的 5.0-CURRENT
500039	September 1, 2002	引入 GCC 3.2.1 之后的 5.0-CURRENT。 头文件也不再使用 BSD_FOO_T 而开始使用 _FOO_T_DECLARED。 这个值还可以用于作为一个包含使用 bzip2(1) 的预编译包支持的预期点。

值	日期	版本
500040	September 20, 2002	以去掉对 disklabel 结构内容的依赖的名义，对磁盘相关的函数进行了许多修改之后的 5.0-CURRENT
500041	October 1, 2002	libc 中加入 getopt_long(3) 之后的 5.0-CURRENT
500042	October 15, 2002	Binutils 2.13 升级， 包含了新的 FreeBSD 模拟， vec 以及输出格式之后的 5.0-CURRENT
500043	November 1, 2002	libc 中加入了弱 pthread_XXX 符号之后的 5.0-CURRENT，从而淘汰了 libXThrStub.so。 5.0-RELEASE。
500100	January 17, 2003	创建 RELENG_5_0 分支之后的 5.0-CURRENT
500101	February 19, 2003	<sys/dkstat.h> 变成了一个空文件， 不应再被引用
500102	February 25, 2003	修改 d_mmap_t 接口之后的 5.0-CURRENT
500103	February 26, 2003	taskqueue_swi 以无全局锁的方式运行之后的 5.0-CURRENT，同时还加入了使用全局锁的 taskqueue_swi_giant
500104	February 27, 2003	去掉了 cdevsw_add() 和 cdevsw_remove() 出现 MAJOR_AUTO 分配机制
500105	March 4, 2003	采用新的 cdevsw 初始化方法之后的 5.0-CURRENT
500106	March 8, 2003	devstat_add_entry() 被 devstat_new_entry() 取代
500107	March 15, 2003	修改 devstat 接口； 请参见 sys/sys/param.h 1.149
500108	March 15, 2003	改变了 Token-Ring 接口
500109	March 25, 2003	加入 vm_paddr_t
500110	March 28, 2003	将 realpath(3) 改为线程安全之后的 5.0-CURRENT
500111	April 9, 2003	usbhid(3) 与 NetBSD 同步之后的 5.0-CURRENT
500112	April 17, 2003	加入新的 NSS 实现， 以及 POSIX.1 getpw*_r, getgr*_r 函数之后的 5.0-CURRENT
500113	May 2, 2003	删去旧式 rc 系统之后的 5.0-CURRENT
501000	June 4, 2003	5.1-RELEASE.
501100	June 2, 2003	创建 RELENG_5_1 分支之后的 5.1-CURRENT

值	日期	版本
501101	June 29, 2003	改正 sigtimedwait(2) 和 sigwaitinfo(2) 语义之后的 5.1-CURRENT
501102	July 3, 2003	在 bus_dma_tag_create(9) 中加入了 lockfunc 和 lockfuncarg 字段之后的 5.1-CURRENT
501103	July 31, 2003	集成了 GCC 3.3.1-pre 20030711 之后的 5.1-CURRENT
501104	August 5, 2003	twe 中 3ware API 变化之后的 5.1-CURRENT
501105	August 17, 2003	允许动态连接 /bin 和 /sbin，以及将某些库移动到 /lib 之后的 5.1-CURRENT
501106	September 8, 2003	增加内核级 Coda 6.x 支持之后的 5.1-CURRENT
501107	September 17, 2003	将 16550 UART 常量从 <dev/sio/sioreg.h> 挪到 <dev/ic/ns16550.h> 之后的 5.1-CURRENT。 此外， rtld 也从此无条件支持 libmap 功能
501108	September 23, 2003	更新 PFIL_HOOKS API 之后的 5.1-CURRENT
501109	September 27, 2003	增加 kiconv(3) 之后的 5.1-CURRENT
501110	September 28, 2003	默认的 cdevsw open 和 close 操作变化之后的 5.1-CURRENT
501111	October 16, 2003	cdevsw 的布局变化之后的 5.1-CURRENT
501112	October 16, 2003	增加 kobj 多继承之后的 5.1-CURRENT
501113	October 31, 2003	修改 struct ifnet 中的 if_xname 之后的 5.1-CURRENT
501114	November 16, 2003	将 /bin 和 /sbin 改为动态连接之后的 5.1-CURRENT
502000	December 7, 2003	5.2-RELEASE
502010	February 23, 2004	5.2.1-RELEASE
502100	December 7, 2003	创建 RELENG_5_2 分支之后的 5.2-CURRENT
502101	December 19, 2003	libc 中加入了 cxa_atexit/cxa_finalize 两个函数之后的 5.2-CURRENT
502102	January 30, 2004	默认线程库从 libc_r 改为 libpthread 之后的 5.2-CURRENT
502103	February 21, 2004	设备驱动 API 大规模翻修之后的 5.2-CURRENT
502104	February 25, 2004	增加 getopt_long_only() 之后的 5.2-CURRENT

值	日期	版本
502105	March 5, 2004	C 的 NULL 定义改为 ((void *)0) 之后的 5.2-CURRENT，这会产生更多的编译警告
502106	March 8, 2004	pf 连入联编和安装过程之后的 5.2-CURRENT
502107	March 10, 2004	在 sparc64 上将 time_t 改为 64-位 值之后的 5.2-CURRENT
502108	March 12, 2004	在一些头文件修改以支持 Intel C/C++ 编译器， 以及让 execve(2) 更严格地符合 POSIX 之后的 5.2-CURRENT
502109	March 22, 2004	引入 bus_alloc_resource_any API 之后的 5.2-CURRENT
502110	March 27, 2004	加入 UTF-8 locale 之后的 5.2-CURRENT
502111	April 11, 2004	删去 getvfsent(3) API 之后的 5.2-CURRENT
502112	April 13, 2004	为 make(1) 增加 .warning 语句之后的 5.2-CURRENT
502113	June 4, 2004	所有串口设备都强制使用 ttyioctl() 之后的 5.2-CURRENT
502114	June 13, 2004	引入 ALTQ 框架之后的 5.2-CURRENT
502115	June 14, 2004	修改 sema_timedwait(9) 使其成功时返回 0， 失败时返回非 0 的错误代码之后的 5.2-CURRENT
502116	June 16, 2004	将内核 dev_t 改为指向 struct cdev * 的指针之后的 5.2-CURRENT
502117	June 17, 2004	将内核 udev_t 改为 dev_t 之后的 5.2-CURRENT
502118	June 17, 2004	为 clock_gettime(2) 和 clock_getres(2) 增加 CLOCK_VIRTUAL 和 CLOCK_PROF 支持之后的 5.2-CURRENT
502119	June 22, 2004	对网络接口复制进行全面修改之后的 5.2-CURRENT
502120	July 2, 2004	package 工具升级为 20040629 之后的 5.2-CURRENT
502121	July 9, 2004	不再将蓝牙代码标记为 i386 专用之后的 5.2-CURRENT
502122	July 11, 2004	引入 KDB 调试器框架之后的 5.2-CURRENT。 同时还引入了 DDB 作为后台， 以及 GDB 后台。
502123	July 12, 2004	修改 VFS_ROOT 和 vflush 使其使用一个 struct thread 参数之后的 5.2-CURRENT。 struct kinfo_proc 增加了一个用户数据指针。 同时，默认的 X 实现切换为 xorg

值	日期	版本
502124	July 24, 2004	将使用 rc.d 和传统脚本的 port 分别启动之后的 5.2-CURRENT
502125	July 28, 2004	取消前一修改之后的 5.2-CURRENT
502126	July 31, 2004	删除 kmem_alloc_pageable() 并引入 gcc 3.4.2 的 5.2-CURRENT
502127	August 2, 2004	修改 UMA 内核 API 允许构建函数和初始化失败之后的 5.2-CURRENT
502128	August 8, 2004	vfs_mount 签名和全局替换 suser(9) API 的 PRISON_ROOT 为 SUSER_ALLOWJAIL 之后的 5.2-CURRENT
503000	August 23, 2004	pfil API 修改之前的 5.3-BETA/RC
503001	September 22, 2004	5.3-RELEASE
503100	October 16, 2004	创建 RELENG_5_3 分支之后的 5.3-STABLE
503101	December 3, 2004	加入了 glibc 风格的 strftime(3) 填充选项的 5.3-STABLE
503102	February 13, 2005	MFC OpenBSD 的 nc(1) 之后的 5.3-STABLE
503103	February 27, 2005	在 MFC 了 <src/include/stdbool.h> 和 <src/sys/i386/include/_types.h> 用于兼容 GCC 和 Intel C/C++ 编译器的修正之后的 5.4-PRERELEASE
503104	February 28, 2005	MFC 了将 ifi_epoch 由 wall 时钟时间改为 uptime 之后的 5.4-PRERELEASE
503105	March 2, 2005	MFC 了 vswprintf(3) 中的 EOVERFLOW 检查的 5.4-PRERELEASE
504000	April 3, 2005	5.4-RELEASE.
504100	April 3, 2005	创建 RELENG_5_4 分支之后的 5.4-STABLE
504101	May 11, 2005	加大默认线程堆栈尺寸之后的 5.4-STABLE
504102	June 24, 2005	加入 sha256 之后的 5.4-STABLE
504103	October 3, 2005	MFC if_bridge 之后的 5.4-STABLE
504104	November 13, 2005	bsdiff 和 portsnap MFC 之后的 5.4-STABLE
504105	January 17, 2006	在 MFC 了 ldconfig_local_dirs 修改之后的 5.4-STABLE。
505000	May 12, 2006	5.5-RELEASE.
505100	May 12, 2006	在创建 RELENG_5_5 分支之后的 5.5-STABLE
600000	August 18, 2004	6.0-CURRENT

值	日期	版本
600001	August 27, 2004	内核中永久性启用 PFIL_HOOKS 之后的 6.0-CURRENT
600002	August 30, 2004	最初将 ifi_epoch 加入 if_data 结构之后的 6.0-CURRENT。 此后不久即被撤销。 请不要使用这个值。
600003	September 8, 2004	if_data 中再次加入 ifi_epoch 成员之后的 6.0-CURRENT
600004	September 29, 2004	将 struct inpcb 参数加入 pfil API 之后的 6.0-CURRENT
600005	October 5, 2004	newsyslog 加入了 "-d DESTDIR" 参数之后的 6.0-CURRENT
600006	November 4, 2004	加入了 glibc 风格的 strftime(3) 填充选项之后的 6.0-CURRENT
600007	December 12, 2004	加入了 802.11 框架更新之后的 6.0-CURRENT
600008	January 25, 2005	修改 VOP_*VOBJECT() 并为无全局锁的文件系统引入 MNTK_MPSAFE 标志之后的 6.0-CURRENT
600009	February 4, 2005	加入 cpufreq 框架和驱动之后的 6.0-CURRENT
600010	February 6, 2005	引入 OpenBSD 的 nc(1) 之后的 6.0-CURRENT
600011	February 12, 2005	删去并不存在的 SVID2 matherr() 支持之后的 6.0-CURRENT
600012	February 15, 2005	增大默认线程堆栈尺寸之后的 6.0-CURRENT
600013	February 19, 2005	增加了针对 <src/include/stdbool.h> 和 <src/sys/i386/include/_types.h> 的用于 Intel C/C++ 编译器的 GCC-兼容性修正。
600014	February 21, 2005	修正了 vswprintf(3) 的 EOVERFLOW 检查之后的 6.0-CURRENT
600015	February 25, 2005	将 struct if_data 成员 ifi_epoch 从 wall 时钟时间改为 uptime 之后的 6.0-CURRENT
600016	February 26, 2005	修改 LC_CTYPE 磁盘格式之后的 6.0-CURRENT
600017	February 27, 2005	修改 NLS 编录磁盘格式之后的 6.0-CURRENT
600018	February 27, 2005	修改 LC_COLLATE 磁盘格式之后的 6.0-CURRENT
600019	February 28, 2005	将 acpica 头文件安装到 /usr/include
600020	March 9, 2005	为 send(2) API 加入了 MSG_NOSIGNAL
600021	March 17, 2005	在 cdevsw 上增加了一些字段

值	日期	版本
600022	March 21, 2005	基本系统中删去了 gtar
600023	April 13, 2005	unix(4) 中加入了 LOCAL_CREDS, LOCAL_CONNWAIT 两个 socket 选项
600024	April 19, 2005	加入了 hwpmc(4) 及其相关工具之后的 6.0-CURRENT
600025	April 26, 2005	加入 struct icmphdr 之后的 6.0-CURRENT
600026	May 3, 2005	pf 更新到了 3.7
600027	May 6, 2005	引入了内核 libalias 和 ng_nat
600028	May 13, 2005	将 ttyname_r(3) 接口改为符合 POSIX 标准， 并通过 unistd.h 和 libc
600029	May 29, 2005	将 libpcap 升级为 v0.9.1 alpha 096 之后的 6.0-CURRENT
600030	June 5, 2005	引入 NetBSD 的 if_bridge(4) 之后的 6.0-CURRENT
600031	June 10, 2005	将 struct ifnet 从驱动的 softc 中拆出之后的 6.0-CURRENT。
600032	July 11, 2005	引入了 libpcap v0.9.1 之后的 6.0-CURRENT。
600033	July 25, 2005	所有自 RELENG_5 以来没有修改过的共享库的版本递增之后的 6.0-STABLE。
600034	August 13, 2005	为 dev_clone 事件处理函数增加身份信息参数之后的 6.0-STABLE。 6.0-RELEASE。
600100	November 1, 2005	6.0-RELEASE 之后的 6.0-STABLE
600101	December 21, 2005	将 local_startup 目录中的脚本集成到基本系统的 rcorder(8) 之后的 6.0-STABLE。
600102	December 30, 2005	更新 ELF 类型和常量之后的 6.0-STABLE。
600103	January 15, 2006	MFC 了 pidfile(3) API 之后的 6.0-STABLE。
600104	January 17, 2006	在 MFC 了 ldconfig_local_dirs 修改之后的 6.0-STABLE。
600105	February 26, 2006	在 csh(1) 中加入了 NLS 目录支持之后的 6.0-STABLE。
601000	May 6, 2006	6.1-RELEASE
601100	May 6, 2006	6.1-RELEASE 之后的 6.1-STABLE。
601101	June 22, 2006	引入 csup 之后的 6.1-STABLE。
601102	July 11, 2006	更新了 iwi(4) 之后的 6.1-STABLE。

值	日期	版本
601103	July 17, 2006	将域名解析函数更新至 BIND9，并导出了可重入版本的 netdb 函数之后的 6.1-STABLE。
601104	August 8, 2006	在 OpenSSL 中启用了 DSO (动态共享库) 支持之后的 6.1-STABLE。
601105	September 2, 2006	由于 802.11 修正变动了 IEEE80211_IOC_STA_INFO ioctl API 之后的 6.1-STABLE。
602000	November 15, 2006	6.2-RELEASE
602100	September 15, 2006	6.2-RELEASE 之后的 6.2-STABLE。
602101	December 12, 2006	加入 Wi-Spy quirk 之后的 6.2-STABLE。
602102	December 28, 2006	增加 pci_find_extcap() 之后的 6.2-STABLE。
602103	January 16, 2007	MFC 了对 dlsym 进行修改，使其在指定 dso 及其暗指的依赖中查找符号之后的 6.2-STABLE。
602104	January 28, 2007	MFC 了 netgraph 节点 ng_deflate(4) 和 ng_pred1(4) 以及用于 ng_ppp(4) 节点的新压缩及加密模式之后的 6.2-STABLE。
602105	February 20, 2007	MFC 了从 NetBSD 移植的 BSD 授权的 gzip(1) 之后的 6.2-STABLE。
602106	March 31, 2007	MFC 了 PCI MSI 和 MSI-X 支持之后的 6.2-STABLE。
602107	April 6, 2007	MFC 了包含宽字符支持的 ncurses 5.6 之后的 6.2-STABLE。
602108	April 11, 2007	MFC 了实现 Linux SCSI SG 直通设备 API 子集的 CAM 'SG' 设备之后的 6.2-STABLE。
602109	April 17, 2007	MFC 了 readline 5.2 patchset 002 之后的 6.2-STABLE。
602110	May 2, 2007	MFC 了用于 amd64 和 i386 的 pmap_invalidate_cache()、pmap_change_attr()、pmap_mapbios()、pmap_mapdev_attr()、 and pmap_unmapbios() 之后的 6.2-STABLE。
602111	June 11, 2007	由于 MFC 了 BOP_BDFLUSH 导致文件系统模块 KBI 变化之后的 6.2-STABLE。
602112	September 21, 2007	一系列 libutil(3) MFC 之后的 6.2-STABLE。

值	日期	版本
602113	October 25, 2007	MFC 了宽字符和单字节 ctype 函数分拆之后的 6.2-STABLE。新编译的引用了 ctype.h 的可执行文件，可能会需要一个在旧系统上不存在的新增符号 __mb_sb_limit。
602114	October 30, 2007	恢复了 ctype ABI 向前兼容性之后的 6.2-STABLE。
602115	November 21, 2007	回退了宽字符和单字节 ctype 分拆之后的 6.2-STABLE。
603000	November 25, 2007	6.3-RELEASE
603100	November 25, 2007	在 6.3-RELEASE 之后的 6.3-STABLE。
603101	December 7, 2007	修正了 bit macro 的多字节支持之后的 6.3-STABLE。
603102	April 24, 2008	为 flock 结构加入 l_sysid 之后的 6.3-STABLE。
603103	May 27, 2008	MFC 了 memrchr 函数之后的 6.3-STABLE。
603104	June 15, 2008	为 make(1) MFC :u 变量修饰符之后的 6.3-STABLE。
604000	October 4, 2008	6.4-RELEASE
604100	October 4, 2008	6.4-RELEASE 之后的 6.4-STABLE。
700000	July 11, 2005	7.0-CURRENT。
700001	July 23, 2005	所有自 RELENG_5 以来没有修改过的共享库的版本递增之后的 7.0-CURRENT。
700002	August 13, 2005	为 dev_clone 事件处理函数中增加身份信息参数之后的 7.0-CURRENT。
700003	August 25, 2005	将 memmem(3) 加入 libc 之后的 7.0-CURRENT。
700004	October 30, 2005	将 solisten(9) 改为接受一 backlog 参数之后的 7.0-CURRENT。
700005	November 11, 2005	将 IFP2ENADDR() 改为返回一 IF_LLADDR() 指针之后的 7.0-CURRENT。
700006	November 11, 2005	在 struct ifnet 中增加 if_addr 成员， 并删除 IFP2ENADDR() 之后的 7.0-CURRENT。
700007	December 2, 2005	将 local_startup 目录中的脚本集成到基本系统的 rcorder(8) 之后的 7.0-CURRENT。
700008	December 5, 2005	去掉 MNT_NODEV 挂接选项之后的 7.0-CURRENT。

值	日期	版本
700009	December 19, 2005	对 ELF-64 类型和符号版本进行变更之后的 7.0-CURRENT。
700010	December 20, 2005	增加 hostb 和 vgapci 驱动、pci_find_extcap()， 并将 AGP 驱动改为不再影射 aperature 之后的 7.0-CURRENT。
700011	December 31, 2005	除 Alpha 之外的所有平台上 tv_sec 改为 time_t 之后的 7.0-CURRENT。
700012	January 8, 2006	修改 ldconfig_local_dirs 之后的 7.0-CURRENT。
700013	January 12, 2006	在修改了 /etc/rc.d/abi 以支持 /compat/linux/etc/ld.so.cache 以某只读文件系统上的符号连接形式存在之后的 7.0-CURRENT。
700014	January 26, 2006	引入 pts 之后的 7.0-CURRENT。
700015	March 26, 2006	在引入 hwpmc(4) 的第 2 版 ABI 之后的 7.0-CURRENT。
700016	April 22, 2006	在 libc 中加入了 fcloseall(3) 之后的 7.0-CURRENT。
700017	May 13, 2006	删去 ip6fw 之后的 7.0-CURRENT。
700018	July 15, 2006	引入了 snd_emu10kx 之后的 7.0-CURRENT。
700019	July 29, 2006	引入了 OpenSSL 0.9.8b 之后的 7.0-CURRENT。
700020	September 3, 2006	增加了 bus_dma_get_tag 函数之后的 7.0-CURRENT。
700021	September 4, 2006	在引入了 libpcap 0.9.4 和 tcpdump 3.9.4 之后的 7.0-CURRENT。
700022	September 9, 2006	在对 dlsym 进行修改，使其在指定 dso 及其暗指的依赖中查找符号之后的 7.0-CURRENT。
700023	September 23, 2006	为 OSSv4 混音器 API 加入新的声音 IOCTL 之后的 7.0-CURRENT。
700024	September 28, 2006	汇入 OpenSSL 0.9.8d 之后的 7.0-CURRENT。
700025	November 11, 2006	加入了 libelf 之后的 7.0-CURRENT。
700026	November 26, 2006	对音效相关的 sysctl 进行大幅调整之后的 7.0-CURRENT。
700027	November 30, 2006	加入 Wi-Spy quirk 之后的 7.0-CURRENT。
700028	December 15, 2006	在 libc 中加入 sctp 调用之后的 7.0-CURRENT。

值	日期	版本
700029	January 26, 2007	将 GNU gzip(1) 实现替换为从 NetBSD 移植的采用 BSD 授权版本之后的 7.0-CURRENT。
700030	February 7, 2007	在 IPv4 多播转发代码中删去了 IPIP 隧道封装 (VIFF_TUNNEL) 之后的 7.0-CURRENT。
700031	February 23, 2007	修改了 bus_setup_intr() (newbus) 之后的 7.0-CURRENT。
700032	March 2, 2007	引入了 ipw(4) 和 iwi(4) 固件之后的 7.0-CURRENT。
700033	March 9, 2007	在 ncurses 中引入了宽字符支持之后的 7.0-CURRENT。
700034	March 19, 2007	修改了 insmntque()、getnewvnode() 以及 vfs_hash_insert() 工作方式之后的 7.0-CURRENT。
700035	March 26, 2007	增加 CPU 频率变动通知机制之后的 7.0-CURRENT。
700036	April 6, 2007	引入了 ZFS 文件系统之后的 7.0-CURRENT。
700037	April 8, 2007	新增了实现 Linux SCSI SG 直通设备 API 子集的 CAM 'SG' 设备之后的 7.0-CURRENT。
700038	April 30, 2007	将 getenv(3)、 putenv(3)、setenv(3) 和 unsetenv(3) 改为符合 POSIX 之后的 7.0-CURRENT。
700039	May 1, 2007	回退了 700038 中的变动之后的 7.0-CURRENT。
700040	May 10, 2007	在 libutil 中增加了 flopen(3) 之后的 7.0-CURRENT。
700041	May 13, 2007	启用了符号版本， 并将 libthr 改为默认线程库之后的 7.0-CURRENT。
700042	May 19, 2007	引入了 gcc 4.2.0 之后的 7.0-CURRENT。
700043	May 21, 2007	将 RELENG_6 之后未修改过版本的共享库版本增加之后的 7.0-CURRENT。
700044	June 7, 2007	将 vn_open()/VOP_OPEN() 的参数由文件描述符数组下标改为 struct file * 之后的 7.0-CURRENT。
700045	June 10, 2007	修改 pam_nologin(8) 使其向 PAM 框架提供帐号管理功能而非身份验证功能之后的 7.0-CURRENT。
700046	June 11, 2007	更新 802.11 无线支持之后的 7.0-CURRENT。

值	日期	版本
700047	June 11, 2007	增加 TCP LRO 网络接口能力之后的 7.0-CURRENT。
700048	June 12, 2007	在 IPv4 协议栈中加入了 RFC 3678 API 支持之后的 7.0-CURRENT。先前 IP_MULTICAST_IF ioctl 的 RFC 1724 行为被删去； 0.0.0.0/8 不再能够用于指定接口索引下标，而应使用 struct ipmreqn 代替。
700049	July 3, 2007	引入 OpenBSD 4.1 的 pf 之后的 7.0-CURRENT。
(not changed)		为 FAST_IPSEC 增加 IPv6 支持，删去 KAME IPSEC， 并将 FAST_IPSEC 更名为 IPSEC 之后的 7.0-CURRENT。 (未变动)
700050	July 4, 2007	将 setenv/putenv/等等调用，从传统 BSD 改为 POSIX 标准之后的 7.0-CURRENT。
700051	July 4, 2007	增加新的 mmap/lseek/等等这些系统调用之后的 7.0-CURRENT。
700052	July 6, 2007	将 I4B 头文件移动到 include/i4b 之后的 7.0-CURRENT。
700053	September 30, 2007	增加了 PCI domain 支持之后的 7.0-CURRENT。
700054	October 25, 2007	MFC 了宽字符和单字节字符 ctype 分拆之后的 7.0-CURRENT。
700055	October 28, 2007	7.0-RELEASE， 以及 MFC 了恢复对 FreeBSD 4/5/6 版本的 PCIOCGETCONF、 PCIOCREAD 和 PCIOCWRITE IOCTL ABI 向下兼容之后的 7.0-CURRENT，这一变动导致 PCIOCGETCONF IOCTL 的 ABI 再次发生变化。
700100	December 22, 2007	7.0-RELEASE 之后的 7.0-STABLE
700101	February 8, 2008	MFC m_collapse() 之后的 7.0-STABLE。
700102	March 30, 2008	MFC kdb_enter_why() 之后的 7.0-STABLE。
700103	April 10, 2008	为 flock 结构加入 l_sysid 之后的 7.0-STABLE。
700104	April 11, 2008	在 procstat(1) MFC 之后的 7.0-STABLE。
700105	April 11, 2008	在 MFC umtx 特性之后的 7.0-STABLE。
700106	April 15, 2008	为 psm(4) MFC write(2) 支持之后的 7.0-STABLE。
700107	April 20, 2008	为 fcntl(2) MFC F_DUP2FD 之后的 7.0-STABLE。

值	日期	版本
700108	May 5, 2008	对 lockmgr(9) 做了一些修改之后的 7.0-STABLE，在使用 lockmgr(9) 时必需包含 sys/lock.h。
700109	May 27, 2008	MFC 了 memrchr 函数之后的 7.0-STABLE。
700110	August 5, 2008	MFC 了内核 NFS locked 客户端之后的 7.0-STABLE。
700111	August 20, 2008	加入了对物理连续巨帧支持之后的 7.0-STABLE。
700112	August 27, 2008	在 MFC 内核 DTrace 支持之后的 7.0-STABLE。
701000	November 25, 2008	7.1-RELEASE
701100	November 25, 2008	7.1-RELEASE 之后的 7.1-STABLE。
701101	January 10, 2009	合并了 strndup 之后的 7.1-STABLE。
701102	January 17, 2009	加入了 cpuctl(4) 支持之后的 7.1-STABLE。
701103	February 7, 2009	合并了 多/无-IPv4/v6 jail 之后的 7.1-STABLE。
701104	February 14, 2009	在 struct mount 中保存了挂起属主，以及在 struct vfsops 中引入了 vfs_susp_clean 方法之后的 7.1-STABLE。
701105	March 12, 2009	对 kern.ipc.shmsegs sysctl 变量不兼容的修改，以允许在 64bit 构架上分配更多的 SysV 共享内存段之后的 7.1-STABLE。
701106	March 14, 2009	合并了一个对 POSIX semaphore 等待操作修正之后的 7.1-STABLE。
702000	April 15, 2009	7.2-RELEASE
702100	April 15, 2009	7.2-RELEASE 之后的 7.2-STABLE。
702101	May 15, 2009	ichsmb(4) 改为使用左邻接辅编址来保持与其它 SMBus 控制器驱动一致性之后的 7.2-STABLE。
702102	May 28, 2009	MFC 了 fdopendir 函数之后的 7.2-STABLE。
702103	June 06, 2009	MFC 了 PmcTools 之后的 7.2-STABLE。
702104	July 14, 2009	MFC 了 closefrom 系统调用之后的 7.2-STABLE。
702105	July 31, 2009	MFC 了 SYSVIPC ABI 改动之后的 7.2-STABLE。

值	日期	版本
702106	September 14, 2009	MFC 了 x86 PAT 增强， 并新增了 d_mmap_single() 以及 scatter/gather 型 VM 对象类型之后的 7.2-STABLE。
703000	February 9, 2010	7.3-RELEASE
703100	February 9, 2010	7.3-RELEASE 之后的 7.3-STABLE。
704000	December 22, 2010	7.4-RELEASE
704100	December 22, 2010	7.4-RELEASE 之后的 7.4-STABLE。
800000	October 11, 2007	8.0-CURRENT。 分拆了宽字符和单字节字符 ctype。
800001	October 16, 2007	引入了 libpcap 0.9.8 和 tcpdump 3.9.8 之后的 8.0-CURRENT。
800002	October 21, 2007	将 kthread_create() 系列函数改名为 kproc_create() 之后的 8.0-CURRENT。
800003	October 24, 2007	恢复了对 FreeBSD 4/5/6 版本的 PCIOCGETCONF、 PCIOCREAD 和 PCIOCWRITE IOCTL ABI 向下兼容之后的 8.0-CURRENT，这一变动导致 PCIOCGETCONF IOCTL 的 ABI 再次发生变化。
800004	November 12, 2007	将 agp(4) 驱动从 src/sys/pci 挪到 src/sys/dev/agp 之后的 8.0-CURRENT。
800005	December 4, 2007	修改了 jumbo frame 分配器之后的 8.0-CURRENT。
800006	December 7, 2007	在给 hwpmc(4) 加入了 callgraph 捕捉功能后的 8.0-CURRENT
800007	December 25, 2007	kdb_enter() 增加 "why" 参数之后的 8.0-CURRENT。
800008	December 28, 2007	在去除 LK_EXCLUPGRADE 选项后的 8.0-CURRENT。
800009	January 9, 2008	引入 lockmgr_disown(9) 之后的 8.0-CURRENT。
800010	January 10, 2008	修改 vn_lock(9) 原型之后的 8.0-CURRENT。
800011	January 13, 2008	修改 VOP_LOCK(9) 和 VOP_UNLOCK(9) 原型之后的 8.0-CURRENT。
800012	January 19, 2008	引入 lockmgr_recursed(9)、BUF_RECURSED(9) 和 BUF_ISLOCKED(9) 并删除了 BUF_REFCNT() 之后的 8.0-CURRENT。
800013	January 23, 2008	引入 "ASCII" 编码之后的 8.0-CURRENT。

值	日期	版本
800014	January 24, 2008	修改 lockmgr(9) 并删除了 lockcount() 和 LOCKMGR_ASSERT() 之后的 8.0-CURRENT。
800015	January 26, 2008	扩展了 fts(3) 数据结构之后的 8.0-CURRENT。
800016	February 1, 2008	为 MEXTADD(9) 增加了一个参数之后的 8.0-CURRENT。
800017	February 6, 2008	为 lockmgr(9) 引入 LK_NODUP 和 LK_NOWITNESS 选项后的 8.0-CURRENT。
800018	February 8, 2008	引入 m_collapse 之后的 8.0-CURRENT。
800019	February 9, 2008	为 sysctl 变量 kern.proc.filedesc 加入 当前工作目录，root 目录和 jail 目录支持之后的 8.0-CURRENT。
800020	February 13, 2008	引入 lockmgr_assert(9) 之后的 8.0-CURRENT。
800021	February 15, 2008	引入 lockmgr_args(9) 和移除 LK_INTERNAL 标志之后的 8.0-CURRENT。
800022	(backed out)	把 BSD ar(1) 作为系统默认的 ar 之后的 8.0-CURRENT。
800023	February 25, 2008	修改了 lockstatus(9) 和 VOP_ISLOCKED(9); 原型，特别时去掉 struct thread 参数之后的 8.0-CURRENT。
800024	March 1, 2008	砍掉了 lockwaiters 和 BUF_LOCKWAITERS 函数，brelvp 的返回值从 void 修改成 int，并引入 lockinit(9) 新标志之后的 8.0-CURRENT。
800025	March 8, 2008	为 fcntl(2) 引入 F_DUP2FD 之后的 8.0-CURRENT。
800026	March 12, 2008	修改了 cv_broadcastpri 优先权参数之后的 8.0-CURRENT，比如 0 表示无优先权。
800027	March 24, 2008	修改了 bpf 监测 ABI, 加入了 zerocopy bpf buffer 之后的 8.0-CURRENT。
800028	March 26, 2008	为 flock 结构增加了 l_sysid 之后的 8.0-CURRENT。
800029	March 28, 2008	重新整合了 BUF_LOCKWAITERS 函数并加入 lockmgr_waiters(9) 之后的 8.0-CURRENT。
800030	April 1, 2008	引入 rw_try_rlock(9) 和 rw_try_wlock(9) 之后的 8.0-CURRENT。

值	日期	版本
800031	April 6, 2008	引入 lockmgr_rw 和 lockmgr_args_rw 函数之后的 8.0-CURRENT。
800032	April 8, 2008	实现了 openat 和相关的系统调用，为 open(2) 引入了 O_EXEC 标志，和提供了相应的 linux 兼容的系统调用之后的 8.0-CURRENT。
800033	April 8, 2008	为 psm(4) 增加了原生的 write(2) 支持之后的 8.0-CURRENT。现在任意命令可写入 /dev/psm%d 并读出状态。
800034	April 10, 2008	引入 memrchr 函数之后的 8.0-CURRENT。
800035	April 16, 2008	引入 fdopendir 函数之后的 8.0-CURRENT
800036	April 20, 2008	无线部分转向 multi-bss (也叫做 vaps) 支持之后的 8.0-CURRENT。
800037	May 9, 2008	加入多路由表支持 (也就是 setfib(1)、 stfib(2)) 后的 8.0-CURRENT。
800038	May 26, 2008	删去了 netatm 和 ISDN4BSD 后的 8.0-CURRENT。这个版本也表示增加了 Compact C Type (CTF) 工具。
800039	June 14, 2008	移除 sgtty 之后的 8.0-CURRENT。
800040	June 26, 2008	增加了内核级 NFS lockd 客户端的 8.0-CURRENT。
800041	July 22, 2008	增加了 arc4random_buf(3) 和 arc4random_uniform(3) 之后的 8.0-CURRENT。
800042	August 8, 2008	增加了 cpuctl(4) 之后的 8.0-CURRENT。
800043	August 13, 2008	修改 bpf(4) 使用单一的设备节点而不是克隆之后的 8.0-CURRENT。
800044	August 17, 2008	在提交了 vimage 项目第一步之后的 8.0-CURRENT。把全局变量重命名为虚拟化带 V_ 前缀并用宏映射到原来的全局名称。
800045	August 20, 2008	引入 MPSAFE TTY 层之后的 8.0-CURRENT，包括对相关驱动和工具的修改。
800046	September 8, 2008	将 amd64 架构上 GDT 拆分到不同 CPU 之后的 8.0-CURRENT。
800047	September 10, 2008	删去了 VSVTX、 VSGID 和 VSUID 之后的 8.0-CURRENT。

值	日期	版本
800048	September 16, 2008	将内核中 NFS 挂接部分的代码改为能够通过 nmount() iovec， 而不再是大的 nfs_args 结构体作为参数之后的 8.0-CURRENT。
800049	September 17, 2008	删去了 suser(9) 和 suser_cred(9) 之后的 8.0-CURRENT。
800050	October 20, 2008	修改了缓冲存储器 API 之后的 8.0-CURRENT。
800051	October 23, 2008	删去了 MALLOC(9) 和 FREE(9) 宏之后的 8.0-CURRENT。
800052	October 28, 2008	引入了 accmode_t 和重新命名 VOP_ACCES 'a_mode' 为 a_accmode 之后的 8.0-CURRENT。
800053	November 2, 2008	修改了 vfs_busy(9) 原型并引入了 MBF_NOWAIT 和 MBF_MNTLSTLOCK 标志之后的 8.0-CURRENT。
800054	November 22, 2008	增加了 buf_ring、 内存栅以及 ifnet 函数， 以方便撰写支持多硬件传输队列的驱动， 以及无锁环形缓冲实现的驱动程序， 并更高效地管理包队列功能之后的 8.0-CURRENT。
800055	November 27, 2008	引入了 hwpmc(4) 对于 Intel™ Core，Core2 和 Atom 的支持之后的 8.0-CURRENT。
800056	November 29, 2008	引入了 multi-/no-IPv4/v6 jail 之后的 8.0-CURRENT。
800057	December 1, 2008	将 ath hal 改为使用源代码之后的 8.0-CURRENT。
800058	December 12, 2008	引入了 VOP_VPTOCNP 操作之后的 8.0-CURRENT。
800059	December 15, 2008	引入了新的 arp-v2 重写之后的 8.0-CURRENT。
800060	December 19, 2008	引入了 makefs 之后的 8.0-CURRENT。
800061	January 15, 2009	引入了 TCP Appropriate Byte Counting 之后的 8.0-CURRENT。
800062	January 28, 2009	删去了 minor()、 minor2unit()、unit2minor() 等之后的 8.0-CURRENT。
800063	February 18, 2009	在 GENERIC 配置中改为使用 USB2 栈之后的 8.0-CURRENT； 这个数值同时也标志新增了 fdevname(3)。
800064	February 23, 2009	将 USB2 栈移动并替换 dev/usb 之后的 8.0-CURRENT。

值	日期	版本
800065	February 26, 2009	在对 libmp(3) 中所有函数更名之后的 8.0-CURRENT。
800066	February 27, 2009	更改了 USB devfs 管理和布局之后的 8.0-CURRENT。
800067	February 28, 2009	加入了 getdelim()， getline()， stpncpy()， strnlen()， wcsnlen()， wcscasecmp()， 和 wcsncasecmp() 之后的 8.0-CURRENT。
800068	March 2, 2009	在 usbhub devclass 更名为 uhub 之后的 8.0-CURRENT。
800069	March 9, 2009	重命名 libusb20.so.1 为 libusb.so.1 之后的 8.0-CURRENT。
800070	March 9, 2009	合并 IGMPv3 和 Source-Specific Multicast (SSM) 入 IPv4 栈之后的 8.0-CURRENT。
800071	March 14, 2009	为 gcc 打上了在 c99 和 gnu99 模式中使用 C99 inline 语义补丁之后的 8.0-CURRENT。
800072	March 15, 2009	移除了 IFF_NEEDSGIANT 标志; 不再支持非线程安全的网络设备驱动之后的 8.0-CURRENT。
800073	March 18, 2009	实现了 rpath 动态字符替换之后的 8.0-CURRENT.
800074	March 24, 2009	引入了 tcpdump 4.0.0 和 libpcap 1.0.0 之后的 8.0-CURRENT。
800075	April 6, 2009	修改了 structs vnet_net、vnet_inet 和 vnet_ipfw 结构布局之后的 8.0-CURRENT。
800076	April 9, 2009	为 dummynet 新增了延迟评估工具之后的 8.0-CURRENT。
800077	April 14, 2009	删去了 VOP_LEASE() 和 vop_vector.vop_lease 之后的 8.0-CURRENT
800078	April 15, 2009	在 struct rt_metrics 和 struct rt_metrics_lite 中添加了 rt_weight 字段， 导致其结构发生变化之后的 8.0-CURRENT。 此后 RTM_VERSION 增加， 但又回退了。
800079	April 15, 2009	在 struct route 和 struct_in6 中添加了 struct llentry 指针之后的 8.0-CURRENT。
800080	April 15, 2009	改变了 struct inpcb 布局之后的 8.0-CURRENT。
800081	April 19, 2009	改变了 malloc_type 布局之后的 8.0-CURRENT。

值	日期	版本
800082	April 21, 2009	改变了 struct ifnet 布局，并增加了 if_ref() 和 if_rele() 引用计数维护功能之后的 8.0-CURRENT。
800083	April 22, 2009	实现了底层蓝牙 HCI API 之后的 8.0-CURRENT。
800084	April 29, 2009	修改了 IPv6 SSM 和 MLDv2 之后的 8.0-CURRENT。
800085	April 30, 2009	启用了包括一个活跃映像的 VIMAGE 内核支持之后的 8.0-CURRENT。
800086	May 8, 2009	为 patch(1) 增加任意长输入行支持之后的 8.0-CURRENT。
800087	May 11, 2009	修改了一些 VFS KPI 之后的 8.0-CURRENT。 VFS 的 FSD 部分中删去了线程参数。 VFS_* 函数并不需要这些上下文信息，因为它总是与 curthread 相关。在某些特殊情况中，则保留了原先的行为。
800088	May 20, 2009	对 net80211 监视模式进行调整之后的 8.0-CURRENT。
800089	May 23, 2009	增加了 UDP 控制块支持之后的 8.0-CURRENT。
800090	May 23, 2009	将网络接口克隆虚拟化之后的 8.0-CURRENT。
800091	May 27, 2009	增加了层次式 jail 并取消全局 securelevel 之后的 8.0-CURRENT。
800092	May 29, 2009	修改了 sx_init_flags() KPI 之后的 8.0-CURRENT。 SX_ADAPTIVESPIN 退役，而新增的 SX_NOADAPTIVE 标志则表达相反语义。
800093	May 29, 2009	为 struct mount 增加 mnt_xflag 之后的 8.0-CURRENT。
800094	May 30, 2009	新增了 VOP_ACCESSX(9) 之后的 8.0-CURRENT。
800095	May 30, 2009	调整轮询 KPI (polling KPI) 之后的 8.0-CURRENT。 轮询处理程序会返回处理过的包的数量。 新增的 IFCAP_POLLING_NOCOUNT 则表示返回值不重要，并跳过计数。
800096	June 1, 2009	对新的 netisr 进行了改进，并调整了保存和存取 FIB 方式之后的 8.0-CURRENT。
800097	June 8, 2009	引入了 vnet 析构挂钩和相关基础设施之后的 8.0-CURRENT。

值	日期	版本
800097	June 11, 2009	引入了 netgraph 输出到输入路径调用检测和排队机制，并调整了 struct thread 布局之后的 8.0-CURRENT。
800098	June 14, 2009	引入了 OpenSSL 0.9.8k 之后的 8.0-CURRENT。
800099	June 22, 2009	更新了 NGROUPS 并将路由虚拟化挪到它自己的 VImage 模块之后的 8.0-CURRENT。
800100	June 24, 2009	修改了 SYSVIPC ABI 之后的 8.0-CURRENT。
800101	June 29, 2009	删去了与网络接口一一对应的 /dev/net/* 字符设备之后的 8.0-CURRENT。
800102	July 12, 2009	在 struct sackhint、struct tcpcb 以及 struct tcpstat 上增加占位元素之后的 8.0-CURRENT。
800103	July 13, 2009	将 TOE 驱动接口中的 struct tcpopt 替换为 TCP syncache 中的 struct toeopt 之后的 8.0-CURRENT。
800104	July 14, 2009	新增了基于 linker-set 的 per-vnet 分配器之后的 8.0-CURRENT。
800105	July 19, 2009	递增了所有未使用符号版本的动态连接库版本之后的 8.0-CURRENT。
800106	July 24, 2009	引入 VM 对象类型 OBJT_SG 之后的 8.0-CURRENT。
800107	August 2, 2009	通过加入 newbus sxlock 使 newbus 子系统不再使用 Giant，以及 8.0-RELEASE。
800108	November 21, 2009	实现了 EVFILT_USER kevent 过滤器之后的 8.0-STABLE。
800500	January 7, 2010	令 pkg_add -r 使用 packages-8-stable 的 __FreeBSD_version 版本变化的 8.0-STABLE。
800501	January 24, 2010	调整了 scandir(3) 和 alphasort(3) 函数原型，使其符合 SUSv4 之后的 8.0-STABLE。
800502	January 31, 2010	新增了 sigpause(3) 之后的 8.0-STABLE。
800503	February 25, 2010	新增了用于管理网络接口说明的 SIOCGIFDESCR 和 SIOCSIFDESCR ioctl 之后的 8.0-STABLE。这组接口受到了 OpenBSD 的启发。
800504	March 1, 2010	MFC 了 x86emu，来自 OpenBSD 的 x86 CPU 实模式模拟器之后的 8.0-STABLE。

值	日期	版本
800505	May 18, 2010	MFC 了添加 liblzma, xz, xzdec 以及 lzmainfo 之后的 8.0-STABLE。
801000	June 14, 2010	8.1-RELEASE
801500	June 14, 2010	8.1-RELEASE 之后的 8.1-STABLE。
801501	November 3, 2010	用于 PL_FLAG_SCE/SCX/EXEC/SI 的 struct sysentvec 的 KBI 以及用于 ptrace(PT_LWPINFO) 的 pl_siginfo 的 KBI 改变之后的 8.1-STABLE。
802000	December 22, 2010	8.2-RELEASE
802500	December 22, 2010	8.2-RELEASE 之后的 8.2-STABLE。
802501	February 28, 2011	合并了 DTrace 变动，包含用户态跟踪支持之后的 8.2-STABLE。
802502	March 6, 2011	在 libm 中合并了 log2 和 log2f 之后的 8.2-STABLE。
802503	May 1, 2011	将 gcc 升级至 FSF gcc-4_2-branch 最后一个 GPLv2 版本之后的 8.2-STABLE。
802504	May 28, 2011	引入模块化拥塞控制支持基础设施和 KPI 之后的 8.2-STABLE。
802505	May 28, 2011	引入了 Hhook 和 Khelp KPI 之后的 8.2-STABLE。
802506	May 28, 2011	在 tcpcb 结构中增加 OSD 之后的 8.2-STABLE。
802507	June 6, 2011	引入 ZFS v28 之后的 8.2-STABLE。
802508	June 8, 2011	删去了 sv_schedtail struct sysvec 方法之后的 8.2-STABLE。
802509	July 14, 2011	在 binutils 中合并了 SSE3 支持之后的 8.2-STABLE。
802510	July 19, 2011	为 rfork(2) 添加了 RFTSIGZMB 标志之后的 8.2-STABLE。
900000	August 22, 2009	9.0-CURRENT。
900001	September 8, 2009	引入了 x86emu， 来自 OpenBSD 的 x86 CPU 实模式模拟器之后的 9.0-CURRENT。
900002	September 23, 2009	实现了 EVFILT_USER kevent 过滤器之后的 9.0-CURRENT。
900003	December 2, 2009	新增了 sigpause(3) 以及 csu 的 PIE 支持之后的 9.0-CURRENT。
900004	December 6, 2009	新增了 libulog 及其 libutempter 兼容接口之后的 9.0-CURRENT。
900005	December 12, 2009	新增了用于查询指定休眠队列上等待者数量的 sleepq_sleepcnt() 函数之后的 9.0-CURRENT。

值	日期	版本
900006	January 4, 2010	调整了 scandir(3) 和 alphasort(3) 函数原型， 使其符合 SUSv4 之后的 9.0-CURRENT。
900007	January 13, 2010	删去了 utmp(5) 并增加了 utmpx (参阅 getutxent(3)) 以改善用户登录日志和系统事件支持之后的 9.0-CURRENT。
900008	January 20, 2010	9.0-CURRENT 引入了 BSD 授权的 bc/dc 并将 GNU bc/dc 标注为过时之后的 9.0-CURRENT。
900009	January 26, 2010	新增了用于管理网络接口说明的 SIOCGIFDESCR 和 SIOCSIFDESCR ioctl 之后的 9.0-CURRENT。 这组接口受到了 OpenBSD 的启发。
900010	March 22, 2010	引入了 zlib 1.2.4 之后的 9.0-CURRENT。
900011	April 24, 2010	添加了 soft-updates 日志功能之后的 9.0-CURRENT。
900012	May 10, 2010	添加了 liblzma, xz, xzdec 以及 lzmainfo 之后的 9.0-CURRENT。
900013	May 24, 2010	添加了针对 linux(4) 的 USB 修正之后的 9.0-CURRENT。
900014	Jun 10, 2010	添加了 Clang 之后的 9.0-CURRENT。
900015	July 22, 2010	引入了 BSD grep 之后的 9.0-CURRENT。
900016	July 28, 2010	在 struct malloc_type_internal 中加入了 mti_zone 之后的 9.0-CURRENT。
900017	August 23, 2010	默认 grep 改回使用 GNU grep 并增加 WITH_BSD_GREP 开关之后的 9.0-CURRENT。
900018	August 24, 2010	将 pthread_kill(3) 产生的信号在 si_code 中改为使用 SI_LWP 标记之后的 9.0-CURRENT。 之前， si_code 对应的标志为 SI_USER。
900019	August 28, 2010	为 mmap(2) 新增了 MAP_PREFAULT_READ 标志之后的 9.0-CURRENT。
900020	September 9, 2010	为 sbuf 增加了 drain 功能并改变了 struct sbuf 布局之后的 9.0-CURRENT。
900021	September 13, 2010	DTrace 增加用户态跟踪支持之后的 9.0-CURRENT。
900022	October 2, 2010	新增了 BSDL man 工具， 并淘汰 GNU/GPL man 工具之后的 9.0-CURRENT。

值	日期	版本
900023	October 11, 2010	引入 20101010 git 快照版本 xz 之后的 9.0-CURRENT。
900024	November 11, 2010	将 libgcc.a 替换为 libcompiler_rt.a 之后的 9.0-CURRENT。
900025	November 12, 2010	引入了模块化拥塞控制之后的 9.0-CURRENT。
900026	November 30, 2010	引入串行管理协议 (SMP) 直通，以及与之对应的 CAM CCB XPT_SMP_IO 和 XPT_GDEV_ADVINFO 之后的 9.0-CURRENT。
900027	December 5, 2010	在 libm 中增加 log2 之后的 9.0-CURRENT。
900028	December 21, 2010	添加了 Hhook (Helper Hook)、Khelp (Kernel Helpers) 和 Object Specific Data (OSD) KPI 之后的 9.0-CURRENT。
900029	December 28, 2010	修改 TCP 协议栈使其允许 Khelp 模块通过 helper hook 指针， 与 TCP 控制块交互并保存连接数据之后的 9.0-CURRENT。
900030	January 12, 2011	将 libdialog 更新至版本 20100428 之后的 9.0-CURRENT。
900031	February 7, 2011	添加了 pthread_getthreadid_np(3) 之后的 9.0-CURRENT。
900032	February 8, 2011	删除了 uio_yield 函数原型和符号之后的 9.0-CURRENT。
900033	February 18, 2011	将 binutils 更新至 2.17.50 之后的 9.0-CURRENT。
900034	March 8, 2011	修改了 struct sysvec (sv_schedtail) 之后的 9.0-CURRENT。
900035	March 29, 2011	将基本系统中 gcc 和 libstdc++ 升级至最后的 GPLv2 授权版本之后的 9.0-CURRENT。
900036	April 18, 2011	在基本系统中删去了 libobjc 和 Objective-C 支持之后的 9.0-CURRENT。
900037	May 13, 2011	在基本系统中引入了 libprocstat(3) 函数库以及 fuser(1) 工具之后的 9.0-CURRENT。
900038	May 22, 2011	为 VFS_FHTOVP(9) 添加锁标志参数之后的 9.0-CURRENT。
900039	June 28, 2011	引入了来自 OpenBSD 4.5 的 pf 之后的 9.0-CURRENT。

值	日期	版本
900040	July 19, 2011	将 amd64 和 ia64 平台上的 MAXCPU 提高到 64， 并把 XLP (mips) 上的值提高到 128 之后的 9.0-CURRENT。
900041	August 13, 2011	实现了 Capsicum capabilities 之后的 9.0-CURRENT。 fget(9) 新增了权限参数。
900042	August 28, 2011	提高修改过 ABI 的动态连接库版本号之后的 9.0-CURRENT。
900043	September 2, 2011	增加了对不支持 SCSI 快取缓存同步功能的 USB 大容量存储设备自动检测功能之后的 9.0-CURRENT。
900044	September 10, 2011	重构了 auto-quirk 之后的 9.0-CURRENT。
900045	Oct 13, 2011	将非兼容性系统调用入口点全部增加 sys_ 前缀之后的 9.0-CURRENT。



请注意， 2.2.5-RELEASE 之后有一段时间的 2.2-STABLE 会声称自己是 "2.2.5-STABLE"。 这种模式的版本号表示的是年月。 但随后， 我们决定， 从 2.2 开始，
将它改为更为简洁的 主/次 版本号的形式来命名版本。
这是因为并行地在多个分支上进行开发，
使得通过实际的发布日期来区分不同的版本变得不再现实。 如果您正在做新的 port，
应该不需要担心较早的 -CURRENT； 在此列出仅供参考。

12.6. 在 bsd.port.mk 之后写一些内容

不要在 .include <bsd.port.mk> 这行之后增加任何内容。 这通常可以通过在您的 Makefile 中间的某处引用 bsd.port.pre.mk， 并在结尾的地方引用 bsd.port.post.mk 来避免。



只能够采用 bsd.port.pre.mk/bsd.port.post.mk 或 bsd.port.mk 两种写法之一；
任何时候都不要同时使用两种写法。

bsd.port.pre.mk 只定义了很少的变量， 它们可以在 Makefile 中用于进行一些测试， 而 bsd.port.post.mk 则定义了所有其它的变量。

下面是一些由 bsd.port.pre.mk 定义的比较重要的变量 (这并不是一份完整的列表， 您可以阅读 bsd.port.mk 以获得全部变量的名字)。

变量	描述
ARCH	由 uname -m 输出得到的硬件架构的名字 (例如， i386)
OPSYS	由 uname -s 返回的操作系统类型 (例如， FreeBSD)
OSREL	操作系统的版本号 (例如 2.1.5 或 2.2.7)
OSVERSION	操作系统的版本号的数值形式； 它等于 __FreeBSD_version。
PORTOBJFORMAT	系统默认的执行文件格式 (elf 或 aout； 请注意， "现代的" FreeBSD 版本中， aout 已在淘汰之列。)
LOCALBASE	"local" 目录的根 (例如， /usr/local/)

变量	描述
PREFIX	port 应被安装到哪里 (参见 关于 PREFIX 的更多说明)。



如果您需要定义 USE_IMAKE， USE_X_PREFIX， 或 MASTERDIR 这些变量， 则应在引用 bsd.port.pre.mk 之前完成。

下面是一些在引用 bsd.port.pre.mk 之后可以进行的判断：

```
# 如果 perl5 已经在系统中提供， 则不必编译 lang/perl5
.if ${OSVERSION} > 300003
BROKEN= perl is in system
.endif

# ELF 只使用一个 shlib 版本
.if ${PORTOBJFORMAT} == "elf"
TCL_LIB_FILE=  ${TCL_LIB}.${SHLIB_MAJOR}
.else
TCL_LIB_FILE=  ${TCL_LIB}.${SHLIB_MAJOR}.${SHLIB_MINOR}
.endif

# 软件会自动为 ELF 创建符号链接， 但 a.out 则需要另行创建
post-install:
.if ${PORTOBJFORMAT} == "aout"
    ${LN} -sf liblinpack.so.1.0 ${PREFIX}/lib/liblinpack.so
.endif
```

您还记得应该在 BROKEN= 和 TCL_LIB_FILE= 后面使用制表符， 而不是空格， 对吧？ :-)

12.7. 在 wrapper 脚本中使用 exec 语句

如果 port 安装了用以启动其他程序的脚本， 并且运行其他程序是这些脚本的最后一项操作， 请务必使用 exec 语句来运行这些程序， 例如：

```
#!/bin/sh
exec %%LOCALBASE%%/bin/java -jar %%DATADIR%%/foo.jar "$@"
```

使用 exec 语句表示执行指定的程序来取代 shell 进程。 如果省略了 exec， 则 shell 进程会一直在内存中， 从而不必要地消耗了额外的系统资源。

12.8. 理性行事

任何 Makefile 都应该简单并理性地行事。 如果您能让其中的条目更为简单和易读， 一定要这样做。 例如， 使用 make 提供的 .if 结构， 而不要使用 shell 的 if， 只要能重定义 EXTRACT* 就不要重载 do-extract， 尽量使用 GNU_CONFIGURE 而不是 CONFIGURE_ARGS += --prefix=${PREFIX}。

如果您在尝试做什么事情的时候发现不得不写大量的代码， 请回过头来复审一下 bsd.port.mk， 看看是否有您正打算做的事情的现成实现。 尽管读起来可能很费劲， 但有很多貌似很难的问题， 在

bsd.port.mk 中都给出了十分简便的解决方案。

12.9. 遵循 CC 和 CXX 设置

port 应遵循 CC 和 CXX 变量的设置。 这也就是说， port 不应使用绝对的方式来设置这个变量的值，
而罔顾已经存在的设置； 与此相反， 它应该在其值后加入需要的其它值。 这样，
就可以设置全局的联编选项， 令其影响所有的 port 联编过程了。

如果实在无法这样做， 请在 Makefile 中加入 NO_PACKAGE=ignores cflags。

下面的 Makefile 实例给出了如何遵循 CC 和 CXX 变量的设置。 注意这里用到的 ?=：

```
CC?= gcc
```

```
CXX?= g++
```

下面则是没有遵循 CC 和 CXX 的例子：

```
CC= gcc
```

```
CXX= g++
```

在 FreeBSD 系统中， CC 和 CXX 这两个变量都可以在 /etc/make.conf 中自行定义。 第一个例子只有在
/etc/make.conf 中没有定义时才对这两个变量进行定义， 从而保持了系统范围的配置。
而第二个例子则会覆盖任何现有的配置。

12.10. 遵循 CFLAGS

您的 port 应遵循 CFLAGS 变量的设置。 这也就是说， port 不应使用绝对的方式来设置这个变量的值，
而罔顾已经存在的设置； 与此相反， 它应该在其值后加入需要的其它值， 这样，
就可以设置全局的联编选项， 令其影响所有的 port 联编过程了。

如果实在无法这样做， 请在 Makefile 中加入 NO_PACKAGE=ignores cflags。

下面的 Makefile 例子， 可以帮助我们理解如何遵循 CFLAGS 的设置。 注意所用的 +=：

```
CFLAGS+= -Wall -Werror
```

下面是一个未能遵循 CFLAGS 设置的例子：

```
CFLAGS= -Wall -Werror
```

一般来说， CFLAGS 在 FreeBSD 系统中是在 /etc/make.conf 里配置的。 第一个例子在 CFLAGS
变量中增加了一些参数， 并保持了所有系统预定义的标志。 而第二个例子，
则会覆盖掉任何先前定义的参数。

您应从第三方软件的 Makefile 中去掉特殊的优化设置。 系统的 CFLAGS
给出了全系统范围内的优化设置参数。 下面是一个未经修改的 Makefile 实例：

```
CFLAGS= -O3 -funroll-loops -DHAVE_SOUND
```

如果使用系统的优化参数， 则 Makefile 中的设置应该类似下面这样：

```
CFLAGS+= -DHAVE_SOUND
```

12.11. 线程库

在 FreeBSD 上， 线程库必须通过特殊的连接器参数 -pthread 连接到可执行文件。 如果 port 一定要直接连接 -lpthread 或 -lc_r， 则应将其改为使用由 ports 框架提供的 PTHREAD_LIBS。 这个变量的值通常是 -pthread， 但在某些特定平台上的 FreeBSD 版本中， 它可能是其它值， 因此， 不要将 -pthread 硬编码到您的补丁中， 而应使用 PTHREAD_LIBS 变量。



如果设置了 PTHREAD_LIBS， 而在联编时出现 unrecognized option '-pthread' 这样的错误， 可能需要通过将 CONFIGURE_ENV 设为 LD=$Cheng Cui <cc@FreeBSD.org> 来使用 gcc 作为连接器。 -pthread 这一选项并不为 ld 所直接支持。

12.12. 反馈

如果进行了一些很好的修改和补丁， 一定要把它们发回给原作者， 或维护者， 以便在下一版本的代码中包含它们。 这会让您在软件发布新版本的时候变得轻松一些。

12.13. README.html

不要包含 README.html 文件。 这个文件并非 CVS 代码库中的一部分， 它是由 make readme 命令生成的。

12.14. 使用 BROKEN、 FORBIDDEN 或 IGNORE 阻止用户安装 port

某些时候会需要阻止用户安装某个 port。 想要告诉用户某个 port 不应被安装， 有许多可以在 port 的 Makefile 中使用的 make 变量。 下列 make 的值， 将是在用户试图安装时得到的提示信息。 务请使用正确的 make 变量， 因为每一个都表达了截然不同的意义， 而且许多自动化系统， 例如 port 联编集群、 FreshPorts， 以及 portsmon， 都依赖于 Makefile 的正确性。

12.14.1. 变量

- BROKEN 专门用于表达目前无法正确编译、 安装或卸载这类问题。 如果是临时性的问题， 则可以使用它。

  如果进行了相关的配置， 则联编集群仍将尝试联编它， 以确认导致问题的深层问题是否已被解决。 (不过， 一般情况下， 联编集群并不会这样做。)

  举例来说， 当 port 发生下述情况时， 应使用 BROKEN：

  - 无法编译 (does not compile)
  - 无法正确进行配置或安装操作
  - 在 ${LOCALBASE} 以外的地方安装文件
  - 卸载时无法删除所安装的全部文件 (不过， 留下用户改过的文件可接受的， 因为可能希望这样作)
- FORBIDDEN 用于表示 ports 中包含安全漏洞， 或者可能会给安装了这个 port 的 FreeBSD 系统带来严重的安全隐患 (例如： 一个很不安全的程序， 或包含了能够被轻易攻陷的服务的软件)。 如果发现了安全漏洞， 而其作者没有发布升级版本， 则应立即把那个 port 标记为 FORBIDDEN。

理想情况下，包含安全漏洞的 port 应被尽快升级，以便减少包含漏洞的 FreeBSD 主机的数量(我们希望保持良好的安全记录)，然而，有时在安全漏洞的披露和软件更新之间可能会有一个间隔，此时应予以说明。除了安全之外，请不要以任何其它理由将 port 标记为 FORBIDDEN。

- IGNORE 用来表示 port 由于某些其它原因不应予以联编。如果认为发生了结构性的问题，则应使用它。任何情况下，联编集群都不会联编标记为 IGNORE 的 port。以下是使用 IGNORE 的一些例子：
  - 能够编译但无法正常运行
  - 无法与运行的 FreeBSD 版本一同工作
  - 联编时需要 FreeBSD 内核的源代码，但用户没有安装它们
  - 由于授权原因，必须手工下载 distfile
  - 无法与的某个已安装的 port 一同工作 (例如，port 依赖于 www/apache21 而安装的则是 www/apache13)



如果 port 与某个已经安装的 port 冲突 (例如，它们在同一位置安装同名但功能不同的文件)，则应 使用 CONFLICTS 来标记它。CONFLICTS 将自动地设置 IGNORE。

- 如果 port 只应在某些平台上标记为 IGNORE，还有另外两个方便使用的 IGNORE 变量可供选择：ONLY_FOR_ARCHS 和 NOT_FOR_ARCHS。例如：

```
ONLY_FOR_ARCHS= i386 amd64
```

```
NOT_FOR_ARCHS= alpha ia64 sparc64
```

可以使用 ONLY_FOR_ARCHS_REASON 和 NOT_FOR_ARCHS_REASON 来配置定制的 IGNORE 消息。此外，还可以使用 ONLY_FOR_ARCHS_REASONARCH_ 和 NOT_FOR_ARCHS_REASONARCH_ 来分别指定与具体平台有关的信息。

- 如果 port 会下载并安装用于 i386 的预编译二进制文件，则应设置 IA32_BINARY_PORT。如果设置了这个变量，则系统会检查是否已经在 /usr/lib32 目录中安装了 IA32 版本的函数库，以及内核是否提供了 IA32 兼容支持。如果这些依赖条件不满足，则会自动设置 IGNORE。

12.14.2. 实现说明

这些字串不应使用引号括起来。此外，由于显示给用户的方式不同，这些字串的措辞也应有所不同。例如：

```
BROKEN= this port is unsupported on FreeBSD 5.x
```

```
IGNORE= is unsupported on FreeBSD 5.x
```

它们分别会在 make describe 时产生下面的输出：

```
===>  foobar-0.1 is marked as broken: this port is unsupported on FreeBSD 5.x.
```

```
===>  foobar-0.1 is unsupported on FreeBSD 5.x.
```

12.15. 使用 DEPRECATED 或 EXPIRATION_DATE 表示某个 port 将被删除

一定要记得 BROKEN 和 FORBIDDEN 只应作为当某个 port 无法正常工作时的临时解决方案。
永久性地坏掉了的 port 应被从 ports tree 中完全删除。

需要时还可以使用 DEPRECATED 和 EXPIRATION_DATE 来通知用户某个 port 不应被使用，
并即将被删除。 前一个变量用来表达为什么计划删除 port； 而后一个是则是一个 ISO 8601 格式的日期 (YYYY-MM-DD)。 两者都会向用户呈现。

也可以设置 DEPRECATED 而不给出 EXPIRATION_DATE (例如， 建议使用某个新版本的 port)，
但反之则没有意义。

目前还没有确切的关于需要给出多少通知的政策。 当前的实践是， 对于与安全有关的问题为一个月，
而与联编有关的问题则为两个月。 这也让有兴趣的 committer 能够有一点时间来修正问题。

12.16. 避免使用 .error 结构

在 Makefile 中给出信号， 表示由于某种外界因素 (例如， 用户指定了无效的联编选项)
而无法安装的方法是将变量 IGNORE 设为一非空值。 这个值将被格式化， 并在用户执行 make install
是给出提示。

用 .error 实现这一目的是一种常见的误用。 这样做的问题是， 许多在 ports
树上运行的自动化工具会因此而失败。 最常见的情况见于联编 /usr/ports/INDEX 的过程 (参见 运行 make describe)。 然而， 即使十分普通的命令， 例如 make maintainer， 在这种情况下也会失败。
这是不可接受的。

例 25. 怎样避免使用 .error

考虑有人在 make.conf 中设置了

```
USE_POINTYHAT=yes
```

的情形。 接下来的例子中， 第一个 Makefile 中的问题将导致 make index 失败， 而第二个则不会：

```
.if USE_POINTYHAT
.error "POINTYHAT is not supported"
.endif
```

```
.if USE_POINTYHAT
IGNORE=POINTYHAT is not supported
.endif
```

12.17. 对于 sysctl 的使用

除了在 target 中之外， 是不鼓励使用 sysctl 的。 这是因为计算 makevar， 例如在 make index
中所进行的那种， 都不得不运行一条命令， 这会使这一操作变得更慢。

在使用 sysctl(8) 时， 务必通过 SYSCTL 变量来进行， 因为此变量将展开成命令的完整路径，
并且用户可以根据需要另行指定。

12.18. 重新发布的 distfiles

有时， 一些软件的作者会修改业已发布的 distfile 的内容， 而并不修改文件名。 这种情况下，
您需要验证这些变动是来自软件作者的官方改动。 在过去， 曾经发生过下载服务器上的 distfile
被悄悄换成注入过恶意代码的版本， 并给用户安全造成威胁或损害的事情。

您应保留一份旧的 distfile， 并下载一份新的， 分别展开， 用 diff(1) 来对比其内容。
如果没有发现可疑的变动， 就可以更新 distinfo 了。 请务必在您的 PR 或 commit log
中对这些差异进行描述， 以便让别人了解您已经仔细对比过差异， 并确认没有问题了。

除此之外， 也可以联系软件的作者， 以确认这些修改是否是他们做的。

12.19. 杂记

需要仔细地反复检查 pkg-descr 和 pkg-plist 这两个文件。 如果您正在复审一个 port，
并认为这两个文件应该改进， 请一定要这样做。

请不要在系统中复制多份 GNU General Public License。

一定要非常小心地处理法律问题！ 不要让我们发布没有得到合法授权的软件！

Chapter 13. 示范的 Makefile

这里是一个您可以在建立新 port 时参考的 Makefile。 请务必删除不需要的那些注释 (方括号中间的文字)!

建议您按照下面这样的格式 (变量顺序， 小节之间的空行等) 来编写。
这个格式的作用是便于查找重要的信息。 我们建议您使用 portlint 来检查 Makefile。

```
[头部... 主要是让我们更容易地分辨不同的 port。]
# New ports collection makefile for:   xdvi
[版本这行， 只有在 PORTVERSION 变量不足以描述 port 时才需要]
# Date created:            26 May 1995
[这是最初将软件移植到 FreeBSD 上的日期， 一般来说是建立这份 Makefile 的日期。
请注意不要在之后再次修改这个日期。]
# Whom:                    Satoshi Asami <asami@FreeBSD.org>
#
# $FreeBSD$
[ ^^^^^^^^^ 这是 CVS 在文件 commit 到我们的代码库时， 自动进行替换的 RCS ID。
如果您正在升级 port， 不要把它改回 "$FreeBSD$"。
CVS 会自动进行处理。]
#

[这个小节描述 port 本身以及主要下载站点 - PORTNAME 和 PORTVERSION
 应放在最前面， 随后是 CATEGORIES， 然后是 MASTER_SITES， 接下来是
 MASTER_SITE_SUBDIR。 如果需要的话， 接下来应指定
 PKGNAMEPREFIX 和 PKGNAMESUFFIX。 随后是 DISTNAME， EXTRACT_SUFX，
 以及 DISTFILES， EXTRACT_ONLY， 如果需要的话。]
PORTNAME=     xdvi
PORTVERSION=  18.2
CATEGORIES=   print
[如果不想使用 MASTER_SITE_* 宏， 一定不要忘记结尾的斜线 ("/")！]
MASTER_SITES=  ${MASTER_SITE_XCONTRIB}
MASTER_SITE_SUBDIR= applications
PKGNAMEPREFIX= ja-
DISTNAME=     xdvi-pl18
[如果源代码包不是标准的 ".tar.gz" 形式， 就需要设置这个]
EXTRACT_SUFX=  .tar.Z

[分散的补丁 -- 可以为空]
PATCH_SITES=  ftp://ftp.sra.co.jp/pub/X11/japanese/
PATCHFILES=   xdvi-18.patch1.gz xdvi-18.patch2.gz

[监护人(maintainer); *必须有*！ 这是某个资源处理 port 更新、 联编失败，
 以及回答用户直接提问或汇报 bug 的人。 为了保证 Ports Collection
```

```
 有尽可能高的品质， 我们不再接受指定给 "ports@FreeBSD.org" 的新 port。]
MAINTAINER=   asami@FreeBSD.org
COMMENT=     A DVI Previewer for the X Window System

[依赖的其它软件包 -- 可以为空]
RUN_DEPENDS=  gs:${PORTSDIR}/print/ghostscript
LIB_DEPENDS=  Xpm.5:${PORTSDIR}/graphics/xpm

[这节是其它不适合上几节的标准 bsd.port.mk 变量]
[如果需要在 configure、 build 或 install 过程中提问...]
IS_INTERACTIVE=      yes
[如果解压缩到 ${DISTNAME} 以外的目录...]
WRKSRC=            ${WRKDIR}/xdvi-new
[如果作者发布的补丁不是相对于 ${WRKSRC} 的， 可能需要调整这个]
PATCH_DIST_STRIP=    -p1
[如果需要运行由 GNU autoconf 生成的 "configure" 脚本]
GNU_CONFIGURE= yes
[如果需要使用 GNU make， 而不是 /usr/bin/make 来完成联编...]
USE_GMAKE=   yes
[如果是一个 X 应用程序， 并使用 "xmkmf -a" 来运行...]
USE_IMAKE=   yes
[et cetera.]

[将在接下来的部分使用的非标准的变量]
MY_FAVORITE_RESPONSE=  "yeah, right"

[接下来是特殊规则， 按调用顺序排列]
pre-fetch:
  i go fetch something, yeah

post-patch:
  i need to do something after patch, great

pre-install:
  and then some more stuff before installing, wow

[结语]
.include <bsd.port.mk>
```

Chapter 14. 保持同步

FreeBSD 的 Ports Collection 在持续地进行修改。 这里提供了一些关于如何保持同步的信息。

14.1. FreshPorts

最简单的了解已经被 commit 到 ports 中的更新的方法， 是订阅 FreshPorts。 您可以选择多个 ports 并对其进行监视。 强烈建议维护人员订阅它， 这样就不仅能接收到他们自己所做的修改， 而且能看到其它 FreeBSD committer 所做的改动。 (保持与所依赖的 ports 框架同步是必要的-虽然一般来说您会在这样的 commit 之前收到一个礼貌性的通知， 但有时可能会有人没有注意到需要这样做， 或者这样做很困难。 另外， 有些时候通知的修改也可能是微不足道的。 我们希望每一个人能够正确地进行判断。)

如果想使用 FreshPorts， 之需要建立一个账号。 如果您注册的邮件地址是 @FreeBSD.org， 您会看到 web 页面右侧的 opt-in 连接。 如果您已经注册了 FreshPorts 账号， 但没有使用 @FreeBSD.org 邮件地址， 则只需把邮件地址改为 @FreeBSD.org， 重新订阅， 并将其改回。

FreshPorts 也会对每一个 FreeBSD ports tree 上的 commit 进行自动的合法性检查。 如果您订阅了这项服务， 则如果发现了错误， 就会收到来自 FreshPorts 的检测报告。

14.2. 代码库的 Web 访问界面

可以通过 web 界面来浏览源代码库中的文件。 影响整个 ports 系统的修改， 现在都会在 CHANGES 文件中说明。 影响某一个 port 的变动， 则在 UPDATING 文件中说明。 尽管如此， 所有问题最为权威的答案， 毫无疑问应该是 bsd.port.mk 的源代码， 以及相关的文件。

14.3. FreeBSD Ports 邮件列表

如果您维护了某个或某一些 ports， 则应该考虑订阅 FreeBSD ports 邮件列表。 对于 ports 工作方式的重要修改都会在此宣示， 并提交到 CHANGES。

14.4. 位于 pointyhat.FreeBSD.org 的 FreeBSD Port 联编集群

FreeBSD 的一个最不为人所知的强项是， 它拥有一个专用于持续联编 Ports Collection 的集群， 这个集群会联编所有主要的 OS 版本在每一个 Tier-1 架构上的 package。 您可以在 package 联编和错误日志 找到其结果。

每一个 port 都会被联编， 除非标记为 IGNORE。 标记了 BROKEN 的 port 仍然会被继续尝试， 以了解是否某些依赖关系的变动解决了其问题 (这是通过给 port 的 Makefile 传 TRYBROKEN 参数来完成的)。

14.5. FreeBSD 的 Ports Distfile 扫描器

联编集群是一组专门用于联编所有 port 最新版本的机器， 其上已经下载了所有的 distfiles。 然而， 由于 Internet 在持续地发生变化， distfile 可能很快就消失了。 FreeBSD Ports distfile 扫描器 试图查询每一个 port 的所有下载站点， 以期找出这些文件是否依然存在。 维护者应规律性地检查这些报告， 这不仅会提高用户联编的速度， 同时也避免了浪费那些镜像了全部 distfile 的志愿者的带宽。

14.6. FreeBSD 的 Ports 追踪系统

另一个非常方便的资源， 就是 FreeBSD Ports 追踪系统 (也被称作 portsmon)。 这个系统包含了一个处理若干信息来源的数据库， 并提供了一个可以通过 web 方式浏览的界面。 目前， 它利用到了和 ports 有关的问题报告 (PR)、 来自联编集群的错误日志， 以及来自 Ports Collection 的文件所提供的信息。 未来， 还会对它进行进一步的扩展， 从而提供包括 distfile 普查， 以及其它来源在内的更多信息。

要使用这个工具， 可以从查看关于某一个 port 的全部资料的 Port 的纵览 开始。

本文撰写时, 这是唯一一个能够将 GNATS PR 项， 同对应的 port 名字映射起来的资源。 (提交 PR 的用户，有时并不在 Synopsis (概要) 中指明 port 的名字， 尽管我们希望他们这样做)。 因此， portsmon 在您想要查找是否有人提交某个现存的 port 的 PR， 以及它的联编是否出现了错误； 或在您创建新的 port 之前想要查找一下是否已经有人提交过时， 就非常有用了。