

FreeBSD Porter 手册

目 录

1. 介 绍	4
2. 自行制作新 port	5
3. 新 的 port	6
3.1. 编写 Makefile	6
3.2. 创建描述文件	6
3.3. 创建校验和文件	8
3.4. 新 port	8
3.5. 用 portlint 来新 port	9
3.6. 提交新 port	9
4. 新的 Porting	10
4.1. 整个系统是如何新的?	10
4.2. 获取源代码	11
4.3. 修改 port	11
4.4. 打补丁	11
4.5. 配置	12
4.6. 处理用输入	13
5. 配置 Makefile	14
5.1. 作者分布的代码	14
5.2. 命名	14
5.3. 分 包	18
5.4. 源包文件	24
5.5. MAINTAINER (维护人)	33
5.6. COMMENT (一句说明)	33
5.7. 依赖性	34
5.8. MASTERDIR (主 port 所在的目录)	38
5.9. 手册	39
5.10. Info 文件	40
5.11. Makefile 选项	40
5.12. 指定工作目录	43
5.13. 处理冲突	44
5.14. 安装文件	45
6. 特殊情况	48
6.1. 共享	48
6.2. Ports 的运行限制	48
6.3. 新机制	49
6.4. 利用 GNU autotools	51
6.5. 使用 GNU gettext	53
6.6. 使用 perl	54

6.7. 使用 X11	55
6.8. 使用 GNOME	58
6.9. 使用 Qt	58
6.10. 使用 KDE	61
6.11. 使用 Java	62
6.12. Web 应用, Apache 和 PHP	65
6.13. 使用 Python	67
6.14. 使用 Tcl/Tk	68
6.15. 使用 Emacs	69
6.16. 使用 Ruby	69
6.17. 使用 SDL	70
6.18. 使用 wxWidgets	71
6.19. 使用 Lua	75
6.20. 使用 Xfce	80
6.21. 使用 Mozilla	80
6.22. 使用数据库	81
6.23. 启动和停止服务 (rc 脚本)	81
6.24. 添加用户和用	83
6.25. 依赖内核源代码的 Ports	84
7. 高级 pkg-plist 用法	85
7.1. 根据 make 量 pkg-plist 行修改	85
7.2. 空目录	86
7.3. 配置文件	86
7.4. 动态装箱与静态装箱的对比	87
7.5. 装箱 (package list) 的自动化制作	87
8. pkg-* 文件	89
8.1. pkg-message (安装包显示的消息文件)	89
8.2. pkg-install (安装包执行的脚本文件)	89
8.3. pkg-deinstall (卸包执行的脚本文件)	89
8.4. pkg-req (安装包是否可行操作的脚本文件)	90
8.5. 改 pkg-* 文件的名字	90
8.6. 使用 SUB_FILES 和 SUB_LIST	90
9. 包的 port	92
9.1. 行 make describe	92
10. 升一个 port	94
10.1. 使用 CVS 制作	95
10.2. UPDATING 和 MOVED 文件	96
11. Ports 的安全	97
11.1. 安全何如此重要	97
11.2. 修安全漏洞	97
11.3. 通知整个用户群体	97

12. 做什么和不做什么	103
12.1. 介绍	103
12.2. <code>WRKDIR</code> (项目使用的目录)	103
12.3. <code>WRKDIRPREFIX</code> (用于项目的目录的父目录名)	103
12.4. 区分不同的操作系统, 以及 OS 的版本	103
12.5. <code>__FreeBSD_version</code>	104
12.6. 在 <code>bsd.port.mk</code> 之后写一些内容	139
12.7. 在 <code>wrapper</code> 脚本中使用 <code>exec</code> 语句	140
12.8. 理性行事	141
12.9. 遵循 <code>CC</code> 和 <code>CXX</code> 设置	141
12.10. 遵循 <code>CFLAGS</code>	141
12.11. 编程	142
12.12. 反	142
12.13. <code>README.html</code>	143
12.14. 使用 <code>BROKEN</code> 、 <code>FORBIDDEN</code> 或 <code>IGNORE</code> 阻止用安装 port	143
12.15. 使用 <code>DEPRECATED</code> 或 <code>EXPIRATION_DATE</code> 表示某个 port 将被删除	144
12.16. 避免使用 <code>.error</code> 语句	145
12.17. 关于 <code>sysctl</code> 的使用	145
12.18. 重新布置的 <code>distfiles</code>	145
12.19. 项目	146
13. 示例的 <code>Makefile</code>	147
14. 保持同步	149
14.1. <code>FreshPorts</code>	149
14.2. 代理的 Web 界面	149
14.3. <code>FreeBSD Ports</code> 文件列表	149
14.4. 位于 pointyhat.FreeBSD.org 的 <code>FreeBSD Port</code> 项目集群	149
14.5. <code>FreeBSD</code> 的 <code>Ports Distfile</code> 扫描器	149
14.6. <code>FreeBSD</code> 的 <code>Ports</code> 追踪系统	150

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。所以在写文档的时候请务必记住些。

在 FreeBSD 的世界里，任何人都能提交新的 port，或志地一个已有的 port，如果那个 port 没人要的 - 不需要任何特殊的限制来做这件事情。

Chapter 2. 自行制作新 port

那，你有没有建自己的 port 或升一个已有的 port？太好了。

下面的内容将会提供一些建 FreeBSD port 的指引。如果想升一个已有的 port，那你在看完些内容并升一个 port。

因本文不是十分，你再参考一下 `/usr/ports/Mk/bsd.port.mk`，所有 port 的 Makefile 文件都会包含它。即使不是每天都去弄 Makefile，也会从那个文件里面得很多知识，里面的注释非常。有要充一下，如果你有其它的，可以 [FreeBSD ports 邮件列表](#) 个 mailing list 信。



在本文里提到的大部分的量 (VAR) 是不能修改的。大多 (但不是全部) 都在 `/usr/ports/Mk/bsd.port.mk` 的初始部分行了介绍；其它一些也可以在那里到。注意些文件使用了非标准的制表符：Emacs 和 Vim 能在打文件的时候自它，而 `vi(1)` 和 `ex(1)` 需要在打文件的时候通 `:set tabstop=4` 来修正默的位置。

想动手？参我的 [希望移植的邮件列表](#) 来看看是否有兴趣完成其中的任。

Chapter 3. 新的 port

这一章将介绍如何快速建立一个全新的 port。很多时候，部分内容是不完整的，需要阅读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` 量所指的地方。默认的情况下，它是 `/usr/ports/distfiles`。



下面的内容假定不需要修改代码就能在 FreeBSD 上通过。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 是个例子的：

```
# 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>
```

看看它是否能通过。不必担心 `$FreeBSD$` 那一行，当这个 port 被放入到 ports 目录的时候，CVS 会自行填写它。可以在 [示例的 Makefile](#) 那章看到更多的。

3.2. 创建描述文件

有 2 个描述文件对于任何一个 port 来说是必需的，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)

它是 port 里一个完整的描述文件。使用一段或几段文件文字来明确的描述这个 ports 是用来做什么的。



不是手册或者如何深入使用/一个port的说明！要是从 *README* 或者机手册里面中制文字的，必小心；通常，它不是个 port 明要的描述，或者用了以使用的格式（比如，机手册里有迫使端端的空格）。如果要移植的件有官方的WWW网，在里列出来。使用 **WWW:** 作前来表示一个网站，其它的自工具就能正常工作了。

下面是一个的 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)

文件列出了 port 所要安装的所有文件。由于 package 也是据此行打包，因此它也被称作 "装箱(packing list)". 个文件中，路径是相对于安装的路径的（通常是 /usr/local 或 /usr/X11R6）。如果使用 **MAN** 量的，不要在里列出任何机手册。假如 port 在安装程中会建一些目，必加的行，以便在 package 被卸予以自除。

下面是一个的例子：

```
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\(1\)](#) 的机手册以得更多有装箱的



建将个文件里的所有的文件名按字母排序。，在升个port的时候就能更方便地核所做的修改。



手工建一列表可能是一件非常枯燥的事情。如果的 port 需要安装大量的文件，自建装箱会省下不少。

只有一情况可以不用 pkg-plist文件。如果个 port 只安装很少量的一些文件或目的，些文件和目就可以分列在 Makefile 的 **PLIST_FILES**和**PLIST_DIRS** 量里。个例子来，我可以在上面那个 oneko port 里面不用 pkg-plist，而把下面的几行加到 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 不需要它自己建目录的，就不用置 `PLIST_DIRS` 变量了。

不，如果用这种方式来列出 port 要安装的文件和目录的，也就无法利用在 `pkg_create(1)` 里介绍的命令来制作 package 了。因此，这个方法只用于那些懒的 port，使它更懒化。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，考虑一下使用这个方法。

最后我们将看到 pkg-plist 以及 `PLIST_FILES` 如何处理更懒的任。

3.3. 建校和文件

只要输入 `make makesum`，port 便会自动建 distinfo 文件。

如果下载的文件的校验和正常化，而又能保证它的来源可靠（比如，来自于CD制造商，或某天生成的文文件），就可在 `IGNOREFILES` 里面声明些文件。同时，再行 `make makesum` 的时候便不会把些 `IGNORE` 的文件算在内了。

3.4. 懒 port

当定义的 port 做了希望它做的事情，包括打包。下面是需要重点的一些重要的工作。

- pkg-plist 中没有包括任何不想安装的文件
- pkg-plist 包含了所有安装的文件
- 的 port 能使用 `reinstall` 多次安装。
- 的 port 能在卸 (deinstall) 时，自动完成清理

Procedure: 推荐的顺序

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

相信在 `package` 和 `deinstall` 阶段没有任何警告。第三以后，是否所有新建的目录都被正确除了。在第四

以后，试着运行一下所装的软件，确保它以 package 方式安装的时候也能正常工作。

自动化这些最慢的方法是通 ports tinderbox 来进行的。它可以 jails 并在其中完成全部工作，而不会破坏正在运行的系统的状态。参考 ports/ports-mgmt/tinderbox 以了解更多的信息。

3.5. 用 portlint 来检查 port

使用 portlint 命令来检查的 port 是否符合我们的期望。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 Makefile 的格式是否符合期望，以及 package 的命名是否得体。

3.6. 提交新 port

在提交新 port 之前，先看看做什么和不做什么。

既然已经所制作的 port 相当好了，剩下的工作，便是将它放到 FreeBSD 的主 ports 目录，以便更多的人从中受益。我们并不需要 work 目录以及 pkgname.tgz 包，因此我们可以删除它们了。假定 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（参考 Bug Reports and General Commentary 以了解关于 send-pr(1) 的一切情况）将其送出。必须将 bug 报告分类 (category) 为 ports 并把子分类 (class) 置为 change-request（不要把报告表及机密的，即 confidential！）。此外，在 PR 的描述 ("Description") 一栏中的内容必须是 port 的简要介绍（例如 COMMENT 内容的简化版本），而 shar 文件则填入修正 ("Fix") 一栏中。



在报告里面使用了一段好的描述，能使我们工作得更容易。通常，我们会使用类似："New port: <category>/<portname> <short description of the port>" 的格式来表明是新的 port。如果我们也使用同样的格式，那我们将会更容易更方便地处理 PR，从而加快处理速度。

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

在提交的 port 以后耐心等待。有时在一个 port 正式加入 FreeBSD 之前需要花好几个个月，尽管也有可能是几天。可以看看正等待被 commit 到 FreeBSD 的 port PR。

一旦我们看到了报告，有必要的我们会联系，并把它放到 ports 目录里。你的名字也会出现在 Additional FreeBSD Contributors 和它的文件。不是很棒吗!?:-)

Chapter 4. 的 Porting

好了，也工作没那，port 需要做些修改才能在 FreeBSD 上起来。在这一章里，我们将会一一例来介绍如何修改来使的 port 能在 FreeBSD 上面行。

4.1. 整个系是如何的？

首先，一系列的作是由用的 port 目里敲入 make 后生的。也会在一个外的一个口里一下 bsd.port.mk 将会有助于的理解。

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

1. **fetch** 会首先被行。**fetch** 将在本地的 **DISTDIR** 目里是否存在 tar 包。如果 **fetch** 没有到就会 Makefile 中定了的 **MASTER_SITES** URL，有我主 FTP 站点 <ftp://ftp.FreeBSD.org/pub/FreeBSD/ports/distfiles/>，在那里我有了所有被可的 distfile。假如那个 **MASTER_SITES** 站点是直接 Internet 上的，就会着用 **FETCH** 指定的程序取回 distfile。如果成功的，文件会被保存在 **DISTDIR** 所指定的目以使用。
2. 接下来会行 **extract**。它会在 **DISTDIR** 中到的 tar 包 (通常是用 gzip 的 tar 包)，然后解到由 **WRKDIR** 所指定的目里 (默 work 目)。
3. 下一是行 **patch**。首先任何在 **PATCHFILES** 中定了的丁都会被打上。然后，在由 **PATCHDIR** 指定的目 (默 files 目) 中的 patch-*, 它将会以文件名的字母序被先后打上。
4. **configure** 会被行。一可能会有以下几种情形。
 - a. 如果存在 scripts/configure，就会行它
 - b. 如果定了 **HAS_CONFIGURE** 或者 **GNU_CONFIGURE**，就会行 **WRKSRCDIR/configure**。
 - c. 如果定了 **USE_IMAGE**，就会行 **XMKMF** (默: **xmkmf -a**)。
5. **build** 会被行。一将会入 ports 的工作目 (**WRKSRCDIR**) 然后行。如果定了 **USE_GMAKE**，就会使用 GNU **make**，反之，会使用系默的 **make**。

以上都是系默的。也可以定 **pre-something** 或者 **post-something**，或者把以此命名的脚本放到 scripts 目，它会在默的作之前或之后被行。

个例子，如果在 Makefile 里定了 **post-extract**，并在 script 目里放了一个 pre-build 脚本，那在 tar 包解之后 **post-extract** 将被用，pre-build 脚本会在默的之前被行。我推荐在 Makefile 定所有的作，如果不是十分的，，人能更容易明白的 port 需要行些非默的作。

默的行都是由 bsd.port.mk 定的 **do-something** 来表示的。例如，port 中用来解的命令是由 **do-extract** 来定的。如果默的置不意，可以通过在 Makefile 重新定 **do-something** 来做些改。



"主" 作 (例如 **extract**、**configure**，等等) 是用来定所有相的段都完成了，以及用真的作或脚本，它不被修改。如果想要修改解个作，可以修改 **do-extract**，但永都不要改 **extract** 的操作！

我已介绍了在用敲入 **make** 之后会生些事情了。接下来我将行一的学，来看一看如何建一个理想的 port。

4.2. 取源代码

取源代码的 tar 包 (通常是 `foo.tar.gz` 或者 `foo.tar.Z`) 并把它放到 `DISTDIR`。最好使用 主流 的版本。

需要置量 `MASTER_SITES` 来指向原始 tar 包的取位置。可以在 `bsd.sites.mk` 里到一些速度快的主流站点。使用些站点 - 和相的定 - 如果可能的, 尽量避免在同一个源代里出大量重的信息。些站点会随着而化, 如果个人都随意加入的会使得非常困。

如果不到一个有很好网接的 FTP/HTTP 站点, 或者它使用了非准的格式, 也就会想在自己的 FTP 或 HTTP 服务器上放上一副本。

如果不到可的地方放置 `distfiles`, 我也可以提供些空来保存它。我自己的 `ftp.FreeBSD.org`; 然而只是一个折衷的法。distfile 必放某人在 `freefall` 上的 `~/public_distfiles/` 目中。可以要求助 commit port 的人来放个 distfile, 而个人也需要把 `MASTER_SITES`、`MASTER_SITE_LOCAL` 以及 `MASTER_SITE_SUBDIR` 的置, 改在 `freefall` 上的用名。

如果的 port 的 distfile 一直在化, 而作者拒改其版本号, 可以考把 distfiles 放在自己的主, 并在 `MASTER_SITES` 里把原作者的列首位置。如果可能, 着与 port 的作者通一下他不要做, 将有助于建立源代的控制。在的主上放置自己的 distfile 会避免用得到 `checksum mismatch` 的, 而且能我 FTP 站点人的工作量。如果的 port 只有一个主站点的, 我建在自己的网站上做一, 并他列 `MASTER_SITES` 的第 2。

如果的 port 需要来自网上的一些丁, 把它放到 `DISTDIR` 里。不用担心它跟源代不是来自同一站点。我有法理 (参下面的 [丁文件](#))。

4.3. 修改 port

解 tar 包, 源代做出合理的修改使得个 port 能在最新版本的 FreeBSD 上面行。一定要仔所改, 包括除、添加、修改的文件等等, 些修改以后会在的 port 中以脚本或丁的方式出, 并且能通行它来自完成 port 的改要求。

如果的 port 要求用与交互/配置来完成或安装的, 可以看一下 Larry Wall 的典的 Configure 脚本, 当地模一下。Port collection 的目的, 就是使个 port 占用最少的空, 并做到件的 "即即用"。



除非明地声明, 否提交 FreeBSD ports collection 的丁, 脚本和它的文件都将以准的 BSD 版布。

4.4. 打丁

在准制作 port 的程中, 加或修改的文件, 都可以通 `diff(1)` 来做成丁。希望用到源代上的个丁, 都保存独的文件, 并命名 `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`。些文件, 都保存在 `PATCHDIR` (通常是 `files/`), 里的丁都会自用到源代上。所有的丁必是相于 `WRKSR` 的 (一般而言, 的 port 会将其 tarball 解在那里, 并完成余下的工作)。了修正和升更容易, 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 `WRKSR/foobar.c`) 情况。需要注意的是, 如果修改的文件的的路径中包含下 (`_`) 字符, 在丁文件名中用个下来代替。例如, 如果需要修改名 `src/freeglut_joystick.c` 的文件, 丁文件的名字 `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，必须不要使用除这些字符以外的其它字符。不要把补丁命名成 `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。我把文件放进 `ports` 的时候，CVS 会坏它，当我再 `check out` 出来的时候，它就会和原来的不一样，从而致打补丁失败。RCS 字符串是由美元符号 (\$) 开头的，通常由 `$Id` 或 `$RCS` 组成。

使用 `diff(1)` 的选项 `(-r)` 很好，但是看一下最后输出的 `patch`，确保没有任何的无用信息。特别地，有 2 个文件不需要 `diff`，并且要排除：一个是 `Makefile`，当该 port 使用了 `Imake`，或者 GNU `configure` 等等的。如果不得不使用 `configure.in` 以使 `autoconf` 去生成 `configure`，不要使用 `configure` 来做 `diff` (它常常会有好几千行!)；设定 `USE_AUTOTOOLS=autoconf:261` 并用 `configure.in` 来制作 `diff`。

另外，要尽量减少补丁中非功能性的空格及空白行。在开源世界中，遵循不同的编码的目录共享大量代码是很常有的事情。如果从某个目录中提取一部分功能用来修正一个程序中的问题，务必小心：补丁中很可能到处都是非功能性的空行。这不仅会导致 CVS 的膨胀，而且也会导致故障点，以及到底修改了什么得不甚清晰。

假如需要删除文件，要在 `post-extract` target，而不是作补丁的一部分来完成。

除此之外，port 的 `Makefile` 可以通过 `in-place` 模式的 `sed(1)` 来直接对行内容的替换操作。如果补丁需要使用变量，就非常有用。例如：

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

往往在移植某些代码的时候会遇到一些问题，特别是当代码是在 Windows® 上开发的时候，大多数的源代码都需要进行 CR/LF 的转换。很可能会在以后打补丁的时候，可能触发警告，并脚本的行数来麻烦 (`/bin/sh^M not found`)，等等。要迅速将所有文件中的 CR/LF 改只用 LF，可以在 port 的 `Makefile` 中加入 `USE_DOS2UNIX=yes`。除此之外，可以指定一个需要进行转换的文件列表：

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

如果希望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。关于正则表达式的说明，参考 `re_format(7)`。这个选项所有指定要展名的文件，例如只源代码文件的用途非常有用：

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

如果希望基于保存的文件来建补丁，可以把文件复制成 `.orig` 扩展名的名字，然后修改原文件。然后使用 `makepatch` 目录根据修改在 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到 `configure` 脚本并把它保存到 `scripts` 子目录。如前面提到的那样，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. 管理用端口入

如果的 port 要求用的输入以便配置、或安装配置程序，就必须在 Makefile 里置 `IS_INTERACTIVE` 量。如果用置了 `BATCH` 的，将用能跳的 port 来完成 "通宵" (如果用置了 `INTERACTIVE` 的，那只有那些要求互的 port 才会被) 将那些不停 ports 的机器省下很多。

通常我们建，如果于那些能有合理的缺省答案的，看一下 `PACKAGE_BUILDING` 量，并根据其置决定是否行交互脚本。将允我 CDROM 和 FTP 来 package。

Chapter 5. 配置 Makefile

配置 Makefile 是相当难的，我在此建一个在开始之前看一下有的例子。在手册里也有一个 [Makefile 例子](#)，照着里面量的顺序来写能使得的 port 更容易地被其它人看。

在，当开始写新的 Makefile 的时候，可以依次思考一下以下的：

5.1. 作者布的代

放在 `DISTDIR` 中的是不是准的用 `gzip` 的 `tar` 包，例如 `foozolix-1.2.tar.gz`？如果是，可以先略一。如果不是，应当看看是不是要覆一些量：`DISTVERSION`、`DISTNAME`、`EXTRACT_CMD`、`EXTRACT_BEFORE_ARGS`、`EXTRACT_AFTER_ARGS`、`EXTRACT_SUFX`，`DISTFILES`，取决于 port 的 `distfile` 格式有多怪。（最常的一个例子便是 `EXTRACT_SUFX=.tar.Z`，一般是因为 `tar` 包是用 `compress` 而不是 `gzip` 的时候。）

最糟的情况是，需要自己写 `do-extract` 来覆默认的定义，尽管不常，但如果遇到了，是需要做。

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。

5.2.1. `PORTNAME` 和 `PORTVERSION`

把 `PORTNAME` 置 port 的名字，`PORTVERSION` 是 port 的版本号。

5.2.2. `PORTREVISION` 和 `PORTEPOCH`

5.2.2.1. `PORTREVISION` (port 的修版本号)

`PORTREVISION` 量是一个的，如果不 0，就会被加到包名的后面，当 `PORTVERSION` 加的时候被置 0（也就是当官方有新版本布的时候）。`PORTREVISION` 会被自化工具（比如 `pkg_version(1)`）用来是否存在可用的新版本。

当 port 生并生成的 package 的内容或有著影，都加 `PORTREVISION`。

下面是一些当修改 `PORTREVISION` 的情况：

- 有新的丁用来修正安全漏洞、，或 port 添加了新的功能。
- 修改了 Makefile 里或禁用的。
- 修改了要安装文件的列表或安装的行（例如，修改了一个用来 package 初始化数据的脚本，如 `ssh host keys`）。
- 一个 port 依的共享版本改（在这种情况下，当安装了新版本的共享，后再去安装早的件就会出，因为它要依老的 `libfoo.x` 而不是 `libfoo.(x+1)`）。
- 原作者修改了 port `distfile`，并且 `distfile` 的新老版本之用 `diff -ru` 只能一些微的化，我只需要 `distinfo` 做相的修正，而不需要修改 `PORTVERSION`。

不需要修改 `PORTREVISION` 的例子：

- port 格式的改，但于打成的包没有功能的上的化。
- MASTER_SITES 生，或行了 port 功能的修改，但不致影最后打成的包。
- distfiles 如修正写之的丁，用而言没有升上的麻。
- 一个原本失的包的修改，使其可，而没有加入新功能。因 PORTREVISION 表示包的内容生了，如果先前没有可的包，也就不需要修改 PORTREVISION 来表示。

一个修改并提交 port 的原是：使得人能从中受益(改、修改已有，或使新的 package 能行)，要衡一下是否那些常更新 ports 的人升，如果回答是"是"的，PORTREVISION 就修改了。

5.2.2.2. PORTEPOCH (port 的加版本号)

有件商或 FreeBSD 的 porter 会使用比旧版的版本号小的数字做新版本号的情况。例来，从 foo-20000801 到 foo-1.0 (从形式上来是不的，因 20000801 在数上比1大很多)。

在这种情况下，PORTEPOCH 当加。如果 PORTEPOCH 非 0，就当加到包名字的后面。PORTEPOCH 永不能被少或清零，因那会致与前一期的 package 比版本生不正的果。(就是，那个 package 就不会被到已了。) 新的版本号 (比如前面在前面那个例子中的 1.0,1) 在数上比前一个版本(20000801)小，但多数自化的工具会 ,1 后意味着比前一个包的后 ,0 大。

的去除或重置 PORTEPOCH 会致很多不幸生；如果不明白前面的，多几次直至明白止，或到件列表上来提。

大多数 port 都不会用到 PORTEPOCH，并且如果某个件的下一个版本改了版本号，用巧妙的方法来定 PORTVERSION 也能避免使用 PORTEPOCH。然而，FreeBSD porter 也需要注意，当有新版本的件布，但并非正式版本 - 比如 "snapshot" 版本，原作者可能会使用当的日期来命名，在新的 "官方" 版本布的时候，就很容易引起前面提到的。

个例子，如果 snapshot 版本的布日期是 20000917，个件的上一个版本是1.2，那这个版本的 PORTVERSION 1.2.20000917 或似的子，而不是20000917，在 1.3 布以后，新版本就可以在数上大于旧的版本了。

5.2.2.3. 于 PORTREVISION 和 PORTEPOCH 的用例

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

```
PORTNAME=      gtkmmble
PORTVERSION=    0.10
```

PKGNAME 成 gtkmmble-0.10。

然后有人了一个安全漏洞，需要用 FreeBSD 的丁。PORTREVISION 就要相的加。

```
PORTNAME=      gtkmmble
PORTVERSION=    0.10
PORTREVISION=    1
```

PKGNAME 成了 gtkmmble-0.10_1

文件的作者发布了新的版本，版本 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" - 但是太晚了）。因而在次版本号 2 在数字上比上一个版本 10 小，PORTEPOCH 必须加，以使新的 package 被置为 "更新的"。由于那是作者发布的一个新版本，因此 PORTREVISION 被置 0（或者从 Makefile 里面删除它）。

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

PKGNAME 成了 gtkmumble-0.2,1

下一个版本将会是 0.3。由于 PORTEPOCH 从不减少，那就无需改：

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

PKGNAME 成 gtkmumble-0.3,1



如果在升级中 PORTEPOCH 被置为 0，那在装了 gtkmumble-0.10_1 包的机器上就无法得到 gtkmumble-0.3 包的更新，因 3 在数字上比 10 小。记住，是 PORTEPOCH 最重要的地方。

5.2.3. PKGNAMEPREFIX 和 PKGNAMESUFFIX

2 个可变的量，PKGNAMEPREFIX 和 PKGNAMESUFFIX 可以和 PORTNAME 有 PORTVERSION 配合使用，形成像 PKGNAME: \${PKGNAMEPREFIX}\${PORTNAME}\${PKGNAMESUFFIX}-\${PORTVERSION}。它必须符合我的包命名。当然，不允许在 PORTVERSION 中使用减号 (-)。如果包名有 language- 或 -compiled.specifcs 部分 (下文)，分别用 PKGNAMEPREFIX 和 PKGNAMESUFFIX，不要直接加到 PORTNAME 中。

5.2.4. LATEST_LINK

LATEST_LINK 在包的程序中用于指定可以 pkg_add -r 使用的短的名字。例如，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。这个名字是独一无二的，并且用而言是而易的名字。

有，在 ports 套件中可能会存在同一程序的多个版本。索引和包的系统都需要能将它不同的包，尽管其 PORTNAME、PKGNAMEPREFIX，甚至 PKGNAMESUFFIX 可能是一模一样的。遇到这种情况，就需要将除了 "主" port 之外的其他 port 中的 LATEST_LINK 量不同的 - 参 lang/gcc46 和 lang/gcc port，以及 www/apache* 系列，以了解它的用法。如果置了 NO_LATEST_LINK，系统便不会生成链接，对于非 "主" port 来是一个可行的。需要注意的是，如何确定 "主" 版本 - "最流行"、"受支持最好"，"最少"，等等 - 已超过了本系统出的建；这里只是向介绍在确定了一个 "主" port 之后如何指定其他 port 的版本。

5.2.5. 包命名

以下是命名包的包当遵守的。将使得我放包的目更利于，因我已 有数以万计的包了，

如果用得看包名很困难的，他可能会很快走的。

一个包的名字看起来像：language_region-name-compiled.specifics-version.numbers。

要像来定包的名字： `${PKGNAMEPREFIX}${PORTNAME}${PKGNAME_SUFFIX}-${PORTVERSION}`。 保所有的量符合上面的格式。

1. FreeBSD 会尽力去支持用当地的言。如果个 port 是某言用的，那 language- 部分是由 ISO-639 定的自然言的 2 个字母写。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。

如果是某言的某一地区的，再要加上 2 个字母的国家代。例如，en_US 表示美国英，fr_CH 表示瑞士法。

language- 部分在 PKGNAMEPREFIX 量里置。

2. name 部分的首字母小写。（余下的部分可以包含大写字母，所以当要一个包含大写字母件的名字，需要自己做出判断。）于 Perl 5 模的命名，有个的是，在前面加上 p5- 并把个冒号的部分改字号，如：Data::Dumper 模的名字，就是 p5-Data-Dumper。
3. port 的名字和版本之间有清晰的分隔，并放入 PORTNAME 和 PORTVERSION 量。在 PORTNAME 中包含版本部分的唯一理由是上游件包真的采用命名方式，似 textproc/libxml2 或 japanese/kinput2-freewnn port 。否，在 PORTNAME 中就不包含任何版本信息。多 port 采用同的 PORTNAME 名字是很正常的，www/apache* port 便是如此；在情况下，不同的版本（以及不同的索引）是由 PKGNAMEPREFIX、PKGNAME_SUFFIX，以及 LATEST_LINK 的不同而有所区别的。
4. 如果 port 可以使用不同的 硬默认配置 行（通常是一系列 port 的一部分目名），-compiled.specifics 部分就明示去的默认（此字号是可的）。通常的用例包括型和不同的字体尺寸。

-compiled.specifics 部分通 PKGNAME_SUFFIX 量来置。

5. 版本号随在字号（-）后面并由数字和字母成。特指出，外的字号是不允出在版本号里的。唯一例外的是字符串 pl（表示 "patchlevel"），只能用在件没有主版本号和次版本号的情况下。如果件的版本号里出了像 "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那些名字，那在数字和字母之间不有多余的小数点。

个方法是了更容易得凭版本号来排序 port。特注意的是，保版本号之的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 0.0.yyyy.mm.dd 的格式，而非 dd.mm.yyyy 甚至 yy.mm.dd 不合表示千年的格式。在版本号上使用 0.0. 前十分重要，因当件行正式的版本，其版本号数字很可能会小于表示年的 yyyy 数字。

里是一些真的例子，我藉此明如何把件作者件的命名，合我包的命名方式：

行版的名字	PKGNAMEPREFIX	PORTNAME	PKGNAME_SUFFIX	PORTVERSION	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

行版的名字	PKGNAMEPREFIX	PORTNAME	PKGNAME_SUFFIX	PORTVERSION	明
rdist-1.3alpha	(空)	rdist	(空)	1.3.a	像 alpha 的字符串是不允出的
es-0.9-beta1	(空)	es	(空)	0.9.b1	像 beta 的字符串是不允出的
mailman-2.0rc3	(空)	mailman	(空)	2.0.rc3	像 rc 的字符串是不允出的
v3.3beta021.src	(空)	tiff	(空)	3.3	那个是鬼西？
tvttwm	(空)	tvttwm	(空)	pl11	需要有个版本号
piewm	(空)	piewm	(空)	1.0	需要有个版本号
xvgr-2.10pl1	(空)	xvgr	(空)	2.10.1	pl 只允在没有主/次版本号的情况下才能出
gawk-2.15.6	ja-	gawk	(空)	2.15.6	日文版
psutils-1.13	(空)	psutils	-letter	1.13	大小已在的候被硬到程序里了
pkfonts	(空)	pkfonts	300	1.0	300dpi 字体的包

如果在原始的代里没有版本号，或者原作者并不打算外的版本，就把版本号成 **1.0** (就像前面 **piewm** 的例子那)。否，要求原始的作者加上版本号或使用日期 (**0.0.yyyy.mm.dd**) 来作版本号。

5.3. 分

5.3.1. CATEGORIES (所属分)

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` 下子目的符号接。些子目的名称是由 **CATEGORIES** 指定的。将方便于那些用在 FTP 站点或 CDROM 的一大堆包里面自己想要的包。看一下 [目前的分表](#)，并出一个合 port 的分。

此列表也会决定的 port 在 port 目中的位置。如果在里定了 1 个以上的分， port 文件放到以第一个分命名的子目中。参 [后面](#) 于如何正分的更多。

5.3.2. 目前的分表

是目前 port 中的分。那些用星号 (*) 的是 虚分 - 它在ports里没有相的子目，因而只用来做

次要的分，用以方便搜索。



对于非虚拟的分来，你会看到在相应子目中的 Makefile 里有写在 **COMMENT** 里的行描述。

分	描述	注意事
accessibility	助残障人士的 port。	
afterstep*	于 AfterStep 窗口管理器的支持。	
arabic	阿拉伯言支持。	
archivers	与工具。	
astro	有天文学的 port。	
audio	声音支持。	
benchmarks	程序。	
biology	生物学相关的件。	
cad	计算机辅助工具。	
chinese	中文言支持。	
comms	通件。	大部分是用于串口通的。
converters	字符。	
databases	数据。	
deskutils	在明计算机以前就已面上使用的西。	
devel	程序工具。	不要把放在里 - 除非再也不到更合的分，否则就不放在个分里。
dns	DNS 相关的件。	
docs*	有 FreeBSD 文的 Meta-ports。	
editors	通用器。	有特殊用途的器被置于相关的分中 (比如，数学-方程式器放在 math 分里。
elisp*	Emacs-lisp 相关的 port。	
emulators	其它操作系的模拟器。	端模拟器 不 属于 个分 - 基于 X 的放在 x11 而基于文本模式的放到 comms 或 misc 中去，取决于具体的功能。
finance	、金融以及相关的用程序。	
french	法言支持。	
ftp	FTP 客户端和服务器的程序。	如果的 port 同时支持 FTP 和 HTTP 的，把它放 ftp 并把 www 做第二分。

分口	描述	注意事口
games	游口。	
geography*	与地理学有口的口件。	
german	德口言支持。	
gnome*	口于 GNOME 口的支持。	
gnustep*	与 GNUstep 口面口境有口的口件。	
graphics	口形口象程序。	
hamradio*	口余无口口口好者使用的口件。	
haskell*	有口 Haskell 口程口言的口件。	
hebrew	希伯来口言支持。	
hungarian	匈牙利口言支持。	
ipv6*	IPv6 相口件。	
irc	IRC 相口程序	
japanese	日口言支持。	
java	与 Java™ 口程口言有口的口件。	java 分口 port 而言不口是其唯一的分口。除了直接与 Java 口言相口的 port 之外，口口人应尽量口避免使用 java 作口 port 的主分口。
kde*	K 口面口境 (KDE) 相口的口件。	
kld*	可加口内核模口。	
korean	口口言支持。	
lang	口程口言。	
linux*	Linux 相口的口用程序。	
lisp*	和 Lisp 口程口言有口的口件。	
mail	口子口件口件。	
math	数口口算和其它数学相口的口件。	
mbone*	MBone 口用程序。	
misc	各式各口的口用程序。	通常不属于其它的任何分口，如果可能的口，尽量口的 port 口 misc 以外的分口，因口在口里的 port 比口容易被人忽略。
multimedia	多媒体口件。	
net	各口网口相口的口件。	
net-im	即口消息口件。	
net-mgmt	网口管理口件。	

分口	描述	注意事项
net-p2p	对等网 (Peer to peer network) 应用程序。	
news	USENET 新闻组相件。	
palm	Palm™ 系列相件。	
parallel*	并行计算相件。	
pear*	Pear PHP 架相件。	
perl5*	Perl5 相的件。	
plan9*	Plan9 相程序。	
polish	波语音支持。	
ports-mgmt	用于管理、安装和 FreeBSD ports 和包的 port。	
portuguese	葡萄牙语支持。	
print	打印相的件。	页面出版工具 (打印工具等等) 也可以放在此分口里。
python*	Python 编程语言相的件。	
ruby*	Ruby 编程语言相的件。	
rubygems*	移植版本的 RubyGems 件包。	
russian	俄语支持。	
scheme*	与 Scheme 语言有关的 port。	
science	科学相但不合放在 astro、biology, 以及 math 分口的 port。	
security	安全相的应用程序。	
shells	命令行 shell。	
spanish*	西班牙语支持	
sysutils	系统相的应用程序。	
tcl*	依赖于 Tcl 行的 port。	
textproc	文本处理的程序。	这个分口并不合于那些放到 print 的页面出版工具。
tk*	依赖于 Tk 行的 port。	
ukrainian	乌克兰语支持。	
vietnamese	越南语支持。	
windowmaker*	WindowMaker 窗口管理器的相支持。	

分口	描述	注意事口
www	Word Wide Web的相口件。	HTML口言相口的支持也可以放在口个分口里。
x11	X Window System以及相口件。	口个分口是口那些直接支持X Window System 的口件的。不要把常口的 X 口用程序也放口里；它口中的大多数都口被口到 x11-* (参口下文)。如果口的 port 是 X 口用程序，口定口 USE_XLIB (使用 USER_IMAKE 口含包括它)，然后把它放到合口的分口里。
x11-clocks	X11 下的口口程序。	
x11-drivers	X11 口口程序。	
x11-fm	X11 下的文件管理器。	
x11-fonts	X11 下的字体以及相口工具。	
x11-servers	X11 服口器。	
x11-themes	X11 主口。	
x11-toolkits	X11 工具包。	
x11-wm	X11 口口管理器。	
xfce*	与 Xfce 口面口境有口的 port。	
zope*	Zope 相口的支持。	

5.3.3. 口口正口的分口

由于不少分口是重口的，口通常在用口个分口作口 port 的主分口上做出口口。下面有几条口能口口解决口个口口。口是一个口口先口的表，按口先口降序口列：

- 第一个分口必口是个物理的分口 (参口 [前面](#))。口口于制作包是必要的。虚口分口和物理分口可能在包制作完成后混合在一起。
- 口于特定口言的分口通常放在第一位。例如，如果口的 port 会安装一些 X11 的日文字体，那口 **CATEGORIES** 那行 就口是 `japanese x11-fonts`。
- 有特定意的分口口当被列在无特定意的前面。例如，HTML 口器口口是口的 `www editors`，而不是其它的什口。同口地，口不口口列出 `net`，如果 port 属于 `irc`、`mail`、`news`、`security`，或是 `www`，因口 `net` 可以表示它口的超集。
- 只有当主要的分口是一口自然口言的口候，`x11` 能被做口第二分口。需要特口指出的是，口不口把 X 的口用程序也口口 `x11`。
- Emacs 模式口当于相口的口用程序放在同一个分口里，而不是 `editors` 分口。口例来口，一个用于口口某口口程口言源代码的 Emacs 模式口口被口 `lang` 一口。
- 需要安装可加口内核模口的 port 口在其 **CATEGORIES** 中口入虚口分口 `kld`。
- `misc` 分口的 port 不能有其它非虚口的分口。如果口在口的 **CATEGORIES** 里口了 `misc` 和口外的分口，那意味着可以安全地口除 `misc` 并把 port 放到其它的子目口中了！

- 如果它的 port 不属于它的分，才把它放到 misc。

如果不能确定使用哪个分，在提交的 [send-pr\(1\)](#) 里加上一行注，我就能在输入 port 之前看一下。如果是 committer，别忘记到 [FreeBSD ports 文件列表](#) 先看一下。很多情况是新的 port 被加到它的分里，然后又立即被移走。会造成源代码不必要和不良的膨胀。

5.3.4. 如何提交建立新的分

由于 Ports Collection 在持续，已引入了许多新的分。新的分既可以是虚的分 - 有些分在整个 ports 目录中没有属于自己的子目 - 或物理的分 - 它有自己的子目。接下来我们将与建立新的物理分有关的事，以便帮助你理解如何提交建立新的分。

我目前的做法是避免建立新的物理分，除非有非常多的 port 被输入一分，或者 port 属于某一特定的小体 (例如，与某人言相)，或者皆是。

这样做的原因是修改会 committer 和用者都不得不行 [多工作](#) 来在 Ports Collection 行或追踪修改。此外，提交新的分通常都会引起争。(可能是因对于某个分是否 "太大" 一直没有非常一致的意的故，一方面，分是否能有助于 (以及多少个分是合的)，等等，也都是。)

下面是具体的：

1. 在 [FreeBSD ports 文件列表](#) 提交新的分。提供建立新分的依据，包括什么现有的分不，以及希望移位置的一系列 port 的名字。(如果有尚在 GNATS 而未 commit 的 port，也一一列出。)如果是相 port 的人或提交者，说明一情况可能有助于的提交得到通。
2. 参与。
3. 如果有人支持你的建，你提交一个 PR，其中包括提交 PR 的理由，以及需要移的 port 的列表。理想情况下，个 PR 也包含下列文件的丁：
 - 行 repocopy 之后 Makefile 行的修改
 - 新分的 Makefile
 - 旧分的 Makefile
 - 依赖于旧 port 的 port 的 Makefile
 - (此外，作一加分因素，可以按照 [Committer 指南](#) 所介的流程，提供一些其它需要修改的文件。)
4. 由于是一影 ports 基施的，它不涉及 repo-copy 的使用，而且也可能影响集群的回操作，因此 PR 分派 Ports 管理 [<portmgr@FreeBSD.org>](mailto:portmgr@FreeBSD.org)。
5. 如果一 PR 得到批准，某个 committer 将按照在 [Committer 指南](#) 中所介的来完成余下的工作。

提交新的虚分和上述程似，但会容易多，因不需要地移任何 port。情况下，PR 附的丁，就只需要修改影到的 port 的 Makefile，以便在其中的 [CATEGORIES](#) 中加入新的分了。

5.3.5. 如何提交分行重新

有些时候会有一些人提交重新将分 2- 或某基于字的。目前止，没有行任何相的改，因

尽管有些修改比它容易完成，但修改整个 Ports Collection 所需要花的工作，至少也是令人生畏的。在发表它之前，它应该在软件列表存档中历史上所花行的提；此外，它也会被要求提供一个可用的原形。

5.4. 源包文件

在 Makefile 中的第二部分是描述用于 port 所必需下的文件，以及到什么地方去下它。

5.4.1. DISTVERSION/DISTNAME (源包版本号/名称)

DISTNAME 是作者称呼所 port 软件的名字。 **DISTNAME** 的默认是 `${PORTNAME}-${PORTVERSION}`，因此只有在需要时才手工指定。 **DISTNAME** 只在两个地方用到。第一个是源包文件列表 (**DISTFILES**)，其默认是 `${DISTNAME}${EXTRACT_SUFX}`。第二个是源包被展到的目录名，即 **WRKSR** 所指定的目录，其默认是 `work/${DISTNAME}`。

某些软件作者布置源包的时并不采取 `${PORTNAME}-${PORTVERSION}` 的模式，它可以通过置 **DISTVERSION** 来自理。 **PORTVERSION** 和 **DISTNAME** 会自地展，当然，也可以改掉它。下表出了一些例子：

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



PKGNAMEPREFIX 和 **PKGNAME_SUFFIX** 并不影响 **DISTNAME**。此外注意 **WRKSR** 等于 `work/${PORTNAME}-${PORTVERSION}`，而源代码的包可能是 `${PORTNAME}-${PORTVERSION}${EXTRACT_SUFX}` 以外的其它名字。一般情况下保持 **DISTNAME** 不变 - 更好的方法是定 **DISTFILES** 而不是同置 **DISTNAME** 和 **WRKSR** (可能有 **EXTRACT_SUFX**)。

5.4.2. MASTER_SITES (主流下站点)

它 FTP/HTTP-URL 指向 **MASTER_SITES** 中原始源的目录部分。不要忘了尾的斜 (/)！

make 宏将使用 **FETCH** 来取所指定的源包文件，如果无法在本地系中到些文件的。

建指定多个像站点，最好是在不同的大洲上的。它将有效地防止由于大网所致无法下的。我甚至打算加自距最近的站点并从那里下的功能；使用多个站点是它做的重要一。

如果原始的源包可以从比流行的件下站点，例如 SourceForge、GNU 或是 Perl CPAN 等等来得，它可能会希望使用似 **MASTER_SITE_*** 的写来表示它 (例如 **MASTER_SITE_SOURCEFORGE**、**MASTER_SITE_GNU** 以及 **MASTER_SITE_PERL_CPAN**)。只需将 **MASTER_SITES** 些量，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，它可以用更略的格式：

```
MASTER_SITES= GNU/make
```

这些量是在 `/usr/ports/Mk/bsd.sites.mk` 中定义的。新项目会随时间增加，因此在提交 port 之前，先看一看该文件的最新版本。

在常用条件下网站的许多“魔法”宏，能够自动判断目录。对于某些站点，只要使用与之相应的写，系统便会自动生成相应的子目录配置。

```
MASTER_SITES= SF
```

如果系统猜的路径不对，可以使用下面表的配置来替换。

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

表 1. 常用的魔法 `MASTER_SITES` 宏

宏	自动猜的子目录
APACHE_JAKARTA	<code>/dist/jakarta/\${PORTNAME:S,-,/,}/source</code>
BERLIOS	<code>/\${PORTNAME:L}</code>
CHEEESHOP	<code>/packages/source/source/\${DISTNAME:C/(.)*\1/}/\${DISTNAME:C/(.)*-[0-9].*\1/}</code>
DEBIAN	<code>/debian/pool/main/\${PORTNAME:C/^((lib)?).*\$/\1}/\${PORTNAME}</code>
GCC	<code>/pub/gcc/releases/\${DISTNAME}</code>
GNOME	<code>/pub/GNOME/sources/\${PORTNAME}/\${PORTVERSION:C/^([0-9]+\.[0-9]+).*\$/\1/}</code>
GNU	<code>/gnu/\${PORTNAME}</code>
MOZDEV	<code>/pub/mozdev/\${PORTNAME:L}</code>
PERL_CPAN	<code>/pub/CPAN/modules/by-module/\${PORTNAME:C/-.*//}</code>
PYTHON	<code>/ftp/python/\${PYTHON_PORTVERSION:C/rc[0-9]//}</code>
RUBYFORGE	<code>/\${PORTNAME:L}</code>
SAVANNAH	<code>/\${PORTNAME:L}</code>
SF	<code>/project/\${PORTNAME:L}/\${PORTNAME:L}/\${PORTVERSION}</code>

5.4.3. `EXTRACT_SUFX` (源包所用的扩展名)

如果一个源包文件，而它使用了某奇怪的扩展名来表明方法，设置 `EXTRACT_SUFX`。

例如，如果源包文件的名称是 `foo.tgz` 而非更一般的 `foo.tar.gz`，写上：

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

`USE_BZIP2` 和 `USE_ZIP` 变量会自动根据需要将 `EXTRACT_SUFX` 置为 `.tar.bz2` 或 `.zip`。如果两个都没置，`EXTRACT_SUFX` 的默认将是 `.tar.gz`。



任何时候都不需要同时置 `EXTRACT_SUFX` 和 `DISTFILES`。

5.4.4. `DISTFILES` (全部源代码包)

有些时候所下的文件名字和 `port` 的名字没有任何关系。例如，可能是 `source.tar.gz`，或者与此类似的其它名字。也有一些其它的组件，它的源代码可能被存放到了不同的包中，而且全都需要下。

如果遇到这种情况，可以将 `DISTFILES` 置为以空格分隔的一个需要下的文件列表。

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

如果没有予以明确的置，`DISTFILES` 的默认将是 `${DISTNAME}${EXTRACT_SUFX}`。

5.4.5. `EXTRACT_ONLY` (只解部分源文件)

如果只有一部分 `DISTFILES` 需要解 - 例如，其中的一个是源代码，而其它是未的文档 - 此把那些需要解的文件加到 `EXTRACT_ONLY` 中。

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

如果 `DISTFILES` 中没有需要解的文件，将 `EXTRACT_ONLY` 置为空串。

```
EXTRACT_ONLY=
```

5.4.6. `PATCHFILES` (通下得到的补丁文件)

如果的 `port` 需要来自 FTP 或 HTTP 的一些外的补丁，将 `PATCHFILES` 置为些文件的名字，并将 `PATCH_SITES` 指向包含些文件的目录的 URL (格式与 `MASTER_SITES` 相同)。

如果些补丁，由于包含了其它的目录名，而致它不是相对于源代码目录的目录 (也就是 `WRKSRC`) 的，就需要相应地置 `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 `foozolib-1.0/`，就置 `PATCH_DIST_STRIP=-p1`。

不需要担心补丁文件本身是否是的；如果文件名以 `.gz` 或 `.Z` 结尾，系会自动解。

如果补丁是同某些其它文件，例如文档，一同以 `gzip` 的 `tar` 格式布的，就不能地使用 `PATCHFILES` 了。情况下，将些补丁包的文件和位置加入到 `DISTFILES` 和 `MASTER_SITES` 中。然后，用 `EXTRA_PATCHES` 置来指出些文件，`bsd.port.mk` 就会自地用些补丁了。需要特注意的是，不要将补丁文件制到

PATCHDIR 目录中 - 一个目录可能是不可写的。



包会以同源代包的方式解包，因此不需要自行完成解包操作，并控制文件。如果一定要做，就要注意，不要解包出来的文件覆盖先前已存在的文件。此外，需要做需要手工加命令，以便在 `pre-clean` target 中删除些控制出来的文件。

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

(包一在某程度上被称作 "源包"；开始源包的作者可能会希望先跳一部分)。

包一提供了被称作 `MASTER_SITES:n` 和 `MASTER_SITES_NN` 的下控制机制。这里我把它称作 `MASTER_SITES:n`。

首先出一些背景。OpenBSD 在其 `DISTFILES` 和 `PATCHFILES` 量中提供了一个很棒的功能，即，允许多些文件和包有 `:n` 后，其中 `n` 可以使用 `[0-9]`，来表。例如：

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

在 OpenBSD 中，源包文件 `alpha` 被到量 `MASTER_SITES0` 而不是公共的 `MASTER_SITES` 量上；而 `beta` 到 `MASTER_SITES1` 上。

是一个很有意思的功能，它可以避免无休止地搜索正的下站点的程。

想象 `DISTFILES` 中指定了 2 个文件，而 `MASTER_SITES` 包含了 20 个站点的情形，其中多站点慢如牛，而 `beta` 可以在 `MASTER_SITES` 的所有站点到，而 `alpha` 只能在第 20 个上面到。如果人了解一点，那所有的站点无疑是在浪，不是吗？当然不是始一个愉快周末的好法！

在有了一个感性的了，想象一下 `DISTFILES` 和更多的 `MASTER_SITES`。然，我的 "distfiles 先生" 会感少他浪在等待下上所耗的。

下一中，将按照 FreeBSD 上述想法的来加以。我 OpenBSD 所提出的概念行了一些改。

5.4.7.1. 化信息

包一将介如何迅速地不同的站点以及子目下多个源包和文件行精的控制。里，我将描述 `MASTER_SITES:n` 的一用法。于多数情况而言做是足的。然而，如果需要更多信息，需要参考下面的几。

一些用程序需要从多个站点下不同的源包。例如，Ghostscript 包括了程序核心本身，以及大量的文件，以及取决于用的打印机品牌和型号的程序。某些文件已随程序核心附，但也有很多需要从其它站点下。

了需要，一个 `DISTFILES` 跟随一个冒号，以及一个 "名"。在 `MASTER_SITES` 的个站点也跟随冒号和名，以便指定从个网站下源包文件。

例如，考一个将源代码包分部分，即 `source1.tar.gz` 和 `source2.tar.gz` 的文件，它必从个不同的站点下。port 的 Makefile 包括似化的 `MASTER_SITES:n` 用法，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
```

多个源包可以使用同一个包。 前面的例子， 假定加了第三个源包， `source3.tar.gz`， 从 `ftp.example2.com` 下。 Makefile 的部分写成 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
```

5.4.7.2. 深入介绍

前面的例子无法满足的需求？ 一旦， 我将介绍 `MASTER_SITES:n` 的精确控制是如何工作的， 以及如何修改包的 port 来使用它。

1. 元素可以包含 `:n` 包的后面， 其中 `n` 是 `[^:;]+`， 概念上即 `n` 可以取任意数字或字母， 但我目前将其限定为 `[a-zA-Z_][0-9a-zA-Z_]+`。

此外， 字符串匹配对大小写是敏感的； 换言之， `n` 与 `N` 不同。

但是， 由于表特殊的意义， 下列包不能用于后面：`default`、`all` 和 `ALL` (它会在 ii 中介包的部分用到)。 此外， `DEFAULT` 是一个有特殊用途的包 (参 3)。

2. 后面 `:n` 的包属于 `n`， 而 `:m` 属于 `m`， 依此类推。
3. 没有后面的元素是无包的， 也就是它们都属于那个特殊的 `DEFAULT` 包。 包元素加入 `DEFAULT` 后通常是多余的， 除非有同包属于 `DEFAULT` 和其它包的元素 (参 5)。

下面的例子是等价的， 但通常用第一个：

```
MASTER_SITES= alpha

MASTER_SITES= alpha:DEFAULT
```

4. 包之间不是互斥的， 同一元素可以同时隶属于多个包， 而包可以空或者有任何多个元素。 同一包中的重复元素， 并不会被自动消除。

5. 如果希望同一元素同时属于多个，可以用逗号 (,) 分隔。

这种方法可以避免指定不同的而多次重复同一元素。例如 `:m,n,o` 表示一个元素同时属于 `m`、`n` 和 `o` 三者。

下面些写法都是等价的，但只推荐使用最后一：

```
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. 同一中的所有站点，会根据 `MASTER_SORT_AWK` 排序。在 `MASTER_SITES` 和 `PATCH_SITES` 中的也会行排序。

7. 在 `MASTER_SITES`、`PATCH_SITES`、`MASTER_SITE_SUBDIR`、`PATCH_SITE_SUBDIR`、`DISTFILES`，以及 `PATCHFILES` 中，都可以使用，其法：

- 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，其后 `:n` 必须出在符号 `/` 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 `n` 产生混淆。为了兼容性的考虑，因之前 `/` 符号在 `MASTER_SITE_SUBDIR` 和 `PATCH_SITE_SUBDIR` 元素中都不是必需的，如果后所跟的字符不是 `/`，`:n` 将被认为是元素的一部分，而不被当作后，即使元素有 `:n` 后。参考在 `MASTER_SITE_SUBDIR` 中 `MASTER_SITES:n` 的用法和 用到逗号分隔符、多个文件，多个站点和不同子目的 `MASTER_SITES:n` 用法 以了解一的。

例 3. 在 `MASTER_SITE_SUBDIR` 中 `MASTER_SITES:n` 的用法

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

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

```
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用于下行的精确控制。站点的列表按照使用的顺序出。

- 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。
- file2 将和 file1 以同样的方式下，因为它属于同一
 - `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。

- file4 将从
 - MASTER_SITE_OVERRIDE
 - <http://site4/>
 - <http://site5/>
 - <http://site6/>
 - <http://site7/>
 - <http://site8/directory-one/>
 - MASTER_SITE_BACKUP

下。

- file5 将从
 - MASTER_SITE_OVERRIDE
 - MASTER_SITE_BACKUP

下。

- file6 将从
 - MASTER_SITE_OVERRIDE
 - <http://site8/>
 - MASTER_SITE_BACKUP

下。

8. 如何来自 `bsd.sites.mk` 的特殊量，例如 `MASTER_SITE_SOURCEFORGE` 行分？

参 `MASTER_SITE_SOURCEFORGE` 中 `MASTER_SITES:n` 的用法。

例 5. `MASTER_SITE_SOURCEFORGE` 中 `MASTER_SITES:n` 的用法

```
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。

9. 如何与 `PATCH*` 量用？

前面的例子介绍的都是 `MASTER*` 量，但于 `PATCH*` 也是完全一的，它在 `化的 PATCH_SITES` 中的 `MASTER_SITES:n` 用法。有所介。

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

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

5.4.7.3. 会改 ports 的哪些行？哪些不会？

- i. 所有普通的 ports 的行都会保持不变。 `MASTER_SITES:n` 功能的代码，只有在某些元素包含了前述，特别是 7 中所提及的 `:n` 后，才会应用。
- ii. 不受影响的 port target：`checksum`、`makesum`、`patch`、`configure`、`build`，等等。当然，`do-fetch`、`fetch-list`、`master-sites` 和 `patch-sites` 的行会发生变化。

- `do-fetch`：会按照新的、有后的 `DISTFILES` 和 `PATCHFILES` 在 `MASTER_SITES` 和 `PATCH_SITES` 所匹配的的元素，以及 `MASTER_SITE_SUBDIR` 和 `PATCH_SITE_SUBDIR` 来行。参 用到逗号分隔符、多个文件，多个站点和不同子目的 `MASTER_SITES:n` 用法。
- `fetch-list`：和旧式的 `fetch-list` 似，但以同 `do-fetch` 相似的方式理。
- `master-sites` 和 `patch-sites`：（与旧版本不兼容）返回 `DEFAULT` 的元素；事实上，它会行 `master-sites-default` 和 `patch-sites-default` 个 target。

更一，使用 `master-sites-all` 或 `patch-sites-all` 个 target 之一，要比直接 `MASTER_SITES` 或 `PATCH_SITES` 更好。此外，未来版本可能不再保直接能正工作。参 B 以了解于些新 target 的更多技。

iii. port 中的新 target

- a. 一系列 `master-sites-n` 和 `patch-sites-n` target 可以分用来列出 `MASTER_SITES` 和 `PATCH_SITES` 中的 `n` 的内容。例如，`master-sites-DEFAULT` 和 `patch-sites-DEFAULT` 都会返回 `DEFAULT` 的内容，而 `master-sites-test` 和 `patch-sites-test` 返回 `test` 的内容，等等。
- b. 新的 `master-sites-all` 和 `patch-sites-all` 个 target，会完成先前 `master-sites` 和 `patch-sites` 所做的工作。它会返回所有的元素，就像些元素都属于同一，并且会列出与 `MASTER_SITE_BACKUP` 或 `MASTER_SITE_OVERRIDE` 中在 `DISTFILES` 或 `PATCHFILES` 中指定的同多个；分于 `master-sites-all` 和 `patch-sites-all`。

5.4.8. `DIST_SUBDIR` (独立的源包子目)

避免的 port 使 `/usr/ports/distfiles` 陷入混乱。如果的 port 需要下很多文件，或者需要下可能与其它 port 的源文件名冲突的文件（例如，`Makefile`），将 `DIST_SUBDIR` 置 port 的名字（通常可以用 `${PORTNAME}` 或 `${PKGNAMEPREFIX}${PORTNAME}`）。将把 `DISTDIR` 从默的 `/usr/ports/distfiles` 改 `/usr/ports/distfiles/DIST_SUBDIR`，并将与的 port 有的文件放到那个目中。

此外，它也会在文件主服务器 `ftp.FreeBSD.org` 上同一子目下的文件（直接在的 `Makefile` 中置 `DISTDIR` 不会有果，因此使用 `DIST_SUBDIR`。）



置并不影响在 `Makefile` 中定的 `MASTER_SITES`。

5.4.9. ALWAYS_KEEP_DISTFILES (一直保存源包)

如果 port 采用的是自己的包，但却采用了某要求源代码必须与特定版本一同提供的授权，例如 GPL，则使用 `ALWAYS_KEEP_DISTFILES` 来告诉 FreeBSD 集群保留一份在 `DISTFILES` 中文件的副本。一般来说某些 port 的用途并不需要这些文件，因此，只在定义了 `PACKAGE_BUILDING` 符号的时候，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
```

当在 `DISTFILES` 加入其它文件时，必须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示到 `WRKDIR` 中，对于某些 ports，可能导致一些不希望副作用，因而需要进行特殊的处理。

5.5. MAINTAINER (维护人)

在此写上自己的电子邮件地址。:-)

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，其格式为 `user@hostname.domain`。不要在此写任何说明性的文字，例如自己的真姓名 - 这会令 `bsd.port.mk` 来麻烦。

维护人有责任保持 port 随时更新，并确保其能正确地运行。每个 port 维护人应明确，参考 [port 维护人面临的挑战](#) 一文。

对于 port 的修改，必须由 port 的维护人进行，且在 commit 之前需要得到其维护人的同意。假如某一 port 的维护人没有在一周之内（不包括主要的公共假日）来自用心的更新请求，则可请其他人超，在这种情况下可以在没有维护人明确同意的情形下进行更新。如果维护人在多三个月的时间内没有进行任何操作，则可以请其他人不辞而别，允许出此范围的 port 进行他人更改。尽管如此，维护人 Ports 管理 <portmgr@FreeBSD.org> 或者 Security Officer 跟 <security-officer@FreeBSD.org> 的 port 不受此限。维护人某些小的 port 行未许可的 commit 是不允许的。

我们保留维护人所提交修正案进行修改的权力，以便使其更符合运行的 Ports Collection 需要，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 在没有得到维护人同意的情形下进行修改。但此类修改都不影响 port 本身的功能。

Ports 管理 <portmgr@FreeBSD.org> 保留以任何原因收回或请任何人明确的权力，而 Security Officer 跟 <security-officer@FreeBSD.org> 保留以安全原因收回或请明确的权力。

5.6. COMMENT (一句说明)

我们用于指定 port 的一句说明。请勿将 package 的名字（或文件的版本）放在说明中。说明的第一个字母大写，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 变量接着 MAINTAINER 变量出。

必将 COMMENT 行限制在不超 70 个字符之内，因行内容会成 pkg_info(1) 呈现用的 port 的一句介。

5.7. 依系

多 ports 会依其它 port。是包括 FreeBSD 在内的多数 Unix 系的很方便的功能。功能，可以避免在个 port 或包中都上重的依的代，而可以以依系的方式去共享它。有七个变量用于助保所需的文件都存在于用的机器上。此外，也提供了用于支持常情形的依系变量，以及依系行的更多控制。

5.7.1. LIB_DEPENDS (依的函数/共享)

个变量用于指定 port 所依的共享。其内容是由一系列 lib:dir:target 元成的表，其中 lib 是共享的名字，而 dir 是在不到从里和安装，最后，target 用于指定在那个目中用的 target。例如，

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

会主版本号 9 的 jpeg 共享，如果它不存在，会入到的 ports 目中的 graphics/jpeg 子目，并和安装它。如果指定的 target 就是 DEPENDS_TARGET (默是 install)，可以略去不写。



lib 部分是一个正则式，用于在 ldconfig -r 的输出中行。可以使用似 intl.[5-7] 和 intl 的。前一模式，即 intl.[5-7]，能匹配 intl.5、intl.6 和 intl.7 中的任意一个。第二模式，即 intl 可以匹配任意版本的 intl。

依系会被次，一次是在 extract target 中，而一次是在 install target。外，依系的名字会放到 package 中，以便 pkg_add(1) 能自地在用系上安装所需的未安装的其它 package。

5.7.2. RUN_DEPENDS (依的行境)

个变量可以用来指定 port 在行所需要的可行文件，以及源文件。它是一系列 path:dir:target 元的列表，里，path 所需的可行，或者源文件的名字，dir 是在无法到些文件或目，去什地方完成和安装以便得些文件；而 target 用来指定在个目中所用的 target 的名字。假如 path 以斜 (/) 始，会当作普通文件，使用 test -e 来；反之，系会假定是一个可行文件，并且用 which -s 来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
```

将文件，或者目录 `/usr/local/etc/innd` 是否存在，如果不到，`port` 目录的 `news/inn` 子目录加以安装。系统也会检查是否在搜索路径中找到名为 `xmlcatmgr` 的文件，如果不到的，会加入 `ports` 目录中的 `textproc/xmlcatmgr` 子目录，并执行和安装的操作。



情况下，`innd` 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，就需要指定完整路径了。

`ports` 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
```

一个依赖系会在 `install` target 的过程中执行。此外，依赖系的名字会被放到 `package` 中，以便 `pkg_add(1)` 能在用的系统中尚未安装相关件自地安装那些 `package`。如果希望指定一个的 `target` 和默的 `DEPENDS_TARGET` 相同，可以略去不写。

一比常的情形是 `RUN_DEPENDS` 和 `BUILD_DEPENDS` 完全一，情况在移植的件是采用脚本言写，或境与行境需求相同尤其普遍。情况可以用下面明了的方式直接将其中一个量一个量：

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

不，有可能令行境被某些没有在 `port` 原本的 `BUILD_DEPENDS` 明定的依赖系染。致情况的原因是 `make(1)` 算量默认采用的是延后算 (`lazy evaluation`)。例如，如果在 `Makefile` 中使用了 `USE_*` 量，些量就会由 `ports/Mk/bsd.*.mk` 理，并填写与之的依赖系。例如，`USE_GMAKE=yes` 会把 `devel/gmake` 加入到 `BUILD_DEPENDS`。如果希望避免些附加的依赖系染 `RUN_DEPENDS`，在使用的时候就需要小心考展的情况，例如，可以在展之前制量的：

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

5.7.3. BUILD_DEPENDS (依赖的行境)

此量用于指定用来 `port` 的可执行文件或源文件。与 `RUN_DEPENDS` 似，它是一个 `path:dir:target` 元的列表。例如，

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

将名为 `unzip` 的可执行文件是否存在，如果不存在，会入到的 `ports` 目录中的 `archivers/unzip` 并完成和安装工作。



里的 "build" 表示从解到全部程。依赖系是在 `extract` target 的过程中。假如要指定的 `target` 和 `DEPENDS_TARGET` 相同，可以略去不写。

5.7.4. `FETCH_DEPENDS` (依赖的下环境)

一个量用于指定 `port` 在下环境所需的可执行文件或源文件。和前一个类似，它是一个 `path:dir:target` 元组。例如，

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

将名称 `ncftp2` 的可执行文件是否存在，如果找不到，将入到 `ports` 目录中的 `net/ncftp2` 子目录并加以编译和安装。

这个依赖系是在 `fetch` target 过程中完成的。如果与 `DEPENDS_TARGET` 相同，可以省略 `target` 部分。

5.7.5. `EXTRACT_DEPENDS` (依赖的解环境)

此量用于指定 `port` 在解环境所需的可执行文件或其它源文件。和前一个量类似，它是一系列 `path:dir:target` 元组的列表。例如，

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

将名称 `unzip` 的可执行文件是否存在，如果不存在，会入到 `ports` 目录中的 `archivers/unzip` 子目录，予以编译和安装。

这个依赖系是在 `extract` target 的过程中完成的。如果与 `DEPENDS_TARGET` 相同，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` 都不能得到需要，才使用这个量。

5.7.6. `PATCH_DEPENDS` (依赖的打补丁环境)

一个量用于指定 `port` 在执行 `patch` 操作所需的可执行文件或其它源文件。和前一个量类似，它是一个 `path:dir:target` 元组的表。例如，

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

表示入到 `ports` 目录中的 `java/jfc` 子目录，并将其解。

这个依赖系是在 `patch` target 的过程中完成的。 `target` 部分如果和 `DEPENDS_TARGET` 相同，就可略去不写。

5.7.7. `USE_*`

提供了一系列量，用以封装大量 `port` 都用到的依赖系。虽然使用这些量是可的，但它能显著少 `port` 的 Makefile 冗性。这些量的共同特征在于，它的名字都是 `USE_*` 的形式。这些量的使用，严格限制于 `port` 的 Makefile 以及 `ports/Mk/bsd.*.mk`，而不用于表示用能设置的 - 情况下采用 `WITH*` 和 `WITHOUT*` 的量。

在任何情况下，都不在 `/etc/make.conf` 中配置任何 `USE_*`。例如，



```
USE_GCC=3.4
```

将致个 port 都依 `gcc34`，甚至包括 `gcc34` 本身！

表 2. 常用的 `USE_*` 量

量	含
<code>USE_BZIP2</code>	此 port 的源包是使用 <code>bzip2</code> 的。
<code>USE_ZIP</code>	此 port 的源包是用 <code>zip</code> 的。
<code>USE_BISON</code>	此 port 在时使用 <code>bison</code> 。
<code>USE_CDRTTOOLS</code>	此 port 需要使用 <code>cdrecord</code> ，根据用的喜好，可能是 <code>sysutils/cdrtools</code> 或 <code>sysutils/cdrtools-cjk</code> 。
<code>USE_GCC</code>	此 port 需要使用某一特定版本的 <code>gcc</code> 才能完成。可以使用似 <code>3.4</code> 的来精指定版本。如果希望使用不低于某一版本的器，可以用 <code>3.4+</code> 的形式。如果与所希望的版本吻合，将使用基本系中所提供的 <code>gcc</code> ，反之，系会从 ports 中安装所希望版本的 <code>gcc</code> ，并整 <code>CC</code> 以及 <code>CXX</code> 量的置。

与 `gmake` 和 `configure` 脚本有的量在 机制 中行了介，而 `autoconf`、`automake` 以及 `libtool` 的介可以在 利用 GNU autotools 到。使用 `perl` 介了与 Perl 有的量。使用 `X11` 中列出了于 `X11` 的量。于 `GNOME` 的量在 使用 `GNOME`，而于 `KDE` 的在 使用 `KDE`。使用 `Java` 述了和 `Java` 有的量，而 `Web 用`，`Apache` 和 `PHP` 包含了于 `Apache`、`PHP` 以及 `PEAR` 的介性信息。于 `Python`，在 使用 `Python` 行了，而于 `Ruby` 的介，可以在 使用 `Ruby` 中到。使用 `SDL` 提供了用于 `SDL` 用程序的量介，最后，使用 `Xfce` 包含了于 `Xfce` 的信息。

5.7.8. 在依系中指定最低版本

在依某个其他 port，可以采用下面的句法，通除 `LIB_DEPENDS` 之外的 `*_DEPENDS` 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 package 的名字，用以与 package 数据中的某匹配，然后是比算符，以及 package 的版本号。前面的例子中，如果系中安装了 `p5-Spiffy-0.26` 足了依条件。

5.7.9. 于依系的充明

如前面所提到的那，在需要某一依的 port，将用 `DEPENDS_TARGET` 所指定的 target。一量的默是 `install`。不是一个用量，它不在 port 的 `Makefile` 中予以定。如果的 port 需要使用特殊的 target 来理依系，使用 `*_DEPENDS` 的 `:target` 部分，而不是重定 `DEPENDS_TARGET` 来完成。

当入 `make clean`，其依的 port 也会自行清理。如果不希望如此，定境量 `NOCLEANDEPENDS`。

如果 port 依赖一些重新编译需要花很长时间的 port，例如 KDE，GNOME 或 Mozilla，这种方法会非常有用。

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。只有在需要时使用其它 port 提供的源代码才这样做。通常也可以通指定来缩短所需的。例如

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

表示依赖 jpeg port 并将其解包。

5.7.10. 循环的依赖系是致命的



不要在 ports tree 中引入任何循环依赖系!

ports 技巧不能容忍循环依赖系。如果引入了的系，就一定会有人安装的 FreeBSD 会因此而坏，而且现象会越来越多。有些情形很；如果有疑，在行修改之前，必行：`cd /usr/ports; make index`。这个程在旧的机器上会很慢，但能大量的用 - 也包括自己 - 救于由所造成的困惑之中。

5.8. MASTERDIR (主 port 所在的目录)

如果 port 需要依赖某些量的置 (例如来，分辨率或型) 来略有不同的包，可以一个的包建立不同的目，可以用更容易地看到他想要安装的版本，但又能在些 port 之共用尽可能多的文件。一般情况下，如果用得当，除主目之外都只需要很短的 Makefile。些 Makefile 中，可以用 `MASTERDIR` 来指定其它文件所在的目。外，使用一个量作为 `PKGNAME_SUFFIX` 的一部分，以便不同的包出不同的命名。

用例子来述些会更明晰。以下是 `japanese/xdvi300/Makefile` 的部分代：

```
PORTNAME=      xdvi
PORTVERSION=   17
PKGNAMEPREFIX= ja-
PKGNAME_SUFFIX= ${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` 也提供了全部常的丁，以及打包用到的文件等等内容。如果在那里入 `make`，它将使用默的分辨率 (300) 并正常地 port。

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 定会告诉 bsd.port.mk 常的目，例如 FILESDIR 以及 SCRIPTDIR 在 xdvi300 中。RESOLUTION=118 行将覆在 xdvi300/Makefile 中所作的 RESOLUTION=300 置，从而 port 将以分辨率 118 的置来。

5.9. 机手册

MAN[1-9LN] 些量，会自地将机手册加到 pkg-plist (也意味着 不能 在 pkg-plist 中列出机手册 - 参 PLIST 的生成 来了解更多)。此外，也会安装段自地根据在 /etc/make.conf 中所作的 NO_MANCOMPRESS 置来自机手册文件行或解操作。

如果 port 通使用符号接或硬接将机手册安装多个名字，就必须使用 MLINKS 量来予以明示。由 port 建的接，将由 bsd.port.mk 除和重建，以它指向了正的文件。任何在 MLINKS 中列出的文件都不在 pkg-plist 中再出。

要指定是否在安装机手册行，可以使用 MANCOMPRESSED 量。一量可以取三，yes、no 和 maybe 之一。yes 表示机手册已以的形式安装，no 表示没有，而 maybe 表示所安装的文件会尊重 NO_MANCOMPRESS 的置，因此 bsd.port.mk 不需要特做什么事情。

如果置了 USE_IMAKE 而未定 NO_INSTALL_MANPAGES，MANCOMPRESSED 会自 yes，反之是 no。除非默不合，否就不需要在 port 中明地加以改。

如果 port 将机手册放到了 PREFIX 之外的其它目，使用 MANPREFIX 来加以置。此外，如果只有某些部分的机手册会安装到不准的位置，例如某些 perl 模的 port，可以使用 MAN_sect_PREFIX (此 sect 是 1-9、L 或 N 之一) 来指定。

如果的机手册需要装入用于某一言用的子目，需要将 MANLANG 那言的名字。此量的默是 "" (也就是只有英)。

下面是一个合的例子。

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

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 `port` 安装，也可能不会。无论如何，都会建立一个符号链接，把 `foo(1)` 和 `alt-name(8)` 的手册链接起来。

假如只有部分手册是翻写的，可以使用一些根据 `MANLANG` 内容生成的量：

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

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，需要在 `INFO` 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。所有列出的目录均是相对于 `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` 写成类似：

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

安装/卸载时就会自动地在注册包之前将它加入到包的 `pkg-plist` 中了。

5.11. Makefile 技巧

某些大型应用程序可以在安装时使用一系列配置选项，用以在系统中已安装了某些库或应用程序时添加一些功能。例如，某些自然（人）语言，GUI 或命令行界面，由于并不是所有的用户都希望使用某些库或者应用程序，`port` 系统提供了一方便机制，来让 `port` 的作者控制包的配置。支持某些特性可以用更体面更好，并达到事半功倍的效果。

5.11.1. 旋钮 (Knobs)

5.11.1.1. WITH_* 和 WITHOUT_*

某些量是系统管理标准的。许多量的量被标准化并置于 [ports/KNOBS](#) 文件。

在建一个 port 的时候，不要使用某个应用程序有的 knob 名称，比如对于 Avahi 一个 port，用 `WITHOUT_MDNS` 而不是 `WITHOUT_AVAHI_MDNS`。



不要假定一个 `WITH_*` 都会有它的 `WITHOUT_*` 量，反之亦然。一般而言，会使用默认。



除非有说明，某些量都是是否定，而不是它设置了 `YES` 或 `NO`。

表 3. 常见的 `WITH_*` 和 `WITHOUT_*` 量

量	意义
<code>WITHOUT-NLS</code>	表示不需要国际化支持，可以节省所消耗的空间。默认情况下，会用国际化支持。
<code>WITH_OPENSSL_BASE</code>	使用基本系统中的 OpenSSL 版本。
<code>WITH_OPENSSL_PORT</code>	从 security/openssl 安装 OpenSSL，即使基本系统中的版本是最新的。
<code>WITHOUT_X11</code>	如果 port 能在是否包含 X 支持的情况下分门别类，一般情况下默认以包含 X 支持的配置来定。如果定义了该量，则表示不包含 X 支持的版本。

5.11.1.2. 旋钮 (knob) 的命名

我们建 port 的人使用相似的，以便最易用，并减少名称的数目。最常用的名字可以在 [KNOBS](#) 文件中找到。

名字反映其功能。如果 port 的 `PORTNAME` 包括 `lib-` 前，名字中去 `lib-` 前。

5.11.2. OPTIONS (菜单式可选项)

5.11.2.1. 背景

`OPTIONS` 将正在安装 port 的用户提供一个包含可用选项的框架，并将用的选项保存到 `/var/db/ports/portname/options` 中。下次重新建 port 时，这些选项将被再次使用。这样一来，就不需要神去之前建 port 的那几十个 `WITH_*` 和 `WITHOUT_*` 了！

当运行 `make config` (或首次运行 `make build`) 时，框架会首先看 `/var/db/ports/portname/options`。如果该文件不存在，它会使用 `OPTIONS` 的用来生成一个可以用或禁用各个选项的框架。随后，用的选项将保存到 `options` 文件中，并被用于该 port。

如果新版本的 port 新了 `OPTIONS`，系统会再次出框架，并根据先前的 `OPTIONS` 配置先前存在的配置。

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

5.11.2.2. 用法

OPTIONS 变量的用法是：

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

默认必须是 **ON** 和 **OFF** 之一。三元组可以使用多次。

定义 **OPTIONS** 变量时，必须在引入 `bsd.port.options.mk` 之前进行。而 **WITH_*** 和 **WITHOUT_*** 变量只能在引入了 `bsd.port.options.mk` 之后才可以进行。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 都在使用该用法。不过，请注意 `bsd.port.pre.mk` 会要求某些变量已先行定义，如 **USE_*** 等。

例 8. 新的 **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身而激活。通常明确地指定相关的 `--without-xxx` 或 `--disable-xxx` 参数到 `CONFIGURE_ARGS` 来禁用不希望的特性。

例 10. 处理错误的做法

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

在前面的例子中，假设系统中已安装了 `libfoo`。用户可能并不希望用程序使用 `libfoo`，因此他在 `make config` 框中删掉了该行。但是，用程序的 `configure` 脚本到了系统中存在该行，并将其加入到了最可行文件支持的功能中。在，如果用决定从系统中卸 `libfoo`，`ports` 系统就无法保证个用程序免遭破坏了（因没有 `libfoo` 的依赖系）。

例 11. 处理错误的正做法

```
.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` 被明确禁用。即使系统中已安装了该行，`configure` 脚本也不会用相关的功能了。

5.12. 指定工作目录

每个 `port` 都会被解到一个工作目录中，该目录必须是可写的。`ports` 系统默认情况下会将 `DISTFILES` 解到一个叫做 `${DISTNAME}` 的目录中。简言之，如果：

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

该 `port` 的源包文件的目录将是 `foo-1.0`。

如果不是所希望的情形，可以修改一系列量的位置。

5.12.1. `WRKSRC` (开始操作的目录名)

该量出了在用程序的源代码包解之后所生成的目录的名字。如果我之前的例子解生成一个叫做 `foo`

(而不是 foo-1.0) 的目录， 如下：

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

或者，也可能是

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

5.12.2. NO_WRKSUBDIR (不需要目录的目录)

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

```
NO_WRKSUBDIR= yes
```

5.13. 冲突

不同的 package 或 port 之间的冲突情形， 系统提供了不同的量来帮助人们行表示： **CONFLICTS**、**CONFLICTS_INSTALL** 和 **CONFLICTS_BUILD**。



一些用于描述冲突的量会自地置 **IGNORE**， 后者的完整介绍， 可以在 [使用 BROKEN、FORBIDDEN 或 IGNORE 阻止用安装 port 到](#)。

在去相互冲突的 port 时， 建议将 **CONFLICTS** 保留几个月， 以便那些不常更新系统的用户能看到。

5.13.1. CONFLICTS_INSTALL

如果的包不能与某些包同安装（例如由于安装同的文件到相同的位置、 行不兼容等等）， 可把其它包的名字列在 **CONFLICTS_INSTALL** 量中。 此可以使用 shell 通配符， 如 * 和 ?。 列出其它包的名字需要遵循它在 /var/db/pkg 中出的子。 可保 **CONFLICTS_INSTALL** 不会匹配到正制作的个包的名字， 否则， 使用 **FORCE_PKG_REGISTER** 来制安装就没有法行了。 于 **CONFLICTS_INSTALL** 的是在程之后、 安装始之前行的。

5.13.2. CONFLICTS_BUILD

如果的包在系中存在某些其它包不能完成， 可把其它包的名字列在 **CONFLICTS_BUILD** 量中。 此可以使用 shell 通配符， 如 * 和 ?。 列出其它包的名字需要遵循它在 /var/db/pkg 中出的子。 于 **CONFLICTS_BUILD** 的是在程始之前行的。 的冲突不会在好的包中予以。

5.13.3. CONFLICTS

如果的 port 在某些其它 port 已存在的情况下既不能， 也不能安装， 可把其它包的名字列在 **CONFLICTS** 量中。 此可以使用 shell 通配符， 如 * 和 ?。 列出其它包的名字需要遵循它在 /var/db/pkg 中出的子。 可保 **CONFLICTS** 不会匹配到正制作的个包的名字， 否则， 使用 **FORCE_PKG_REGISTER** 来制安装就没有法行了。 于 **CONFLICTS** 的是在程之后、 安装始之前行的。

5.14. 安装文件

5.14.1. INSTALL_* 宏

一定要使用由 `bsd.port.mk` 提供的宏，以保持在自己的 `*-install` target 中能以自己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` 是安装库文件的命令。
- `INSTALL_KLD` 是用于安装可加载式内核模块的命令。在某些平台上，当内核模块运行 `strip` 之后会导致一些问题，因此使用这个宏而不是 `INSTALL_PROGRAM` 来安装内核模块。
- `INSTALL_DATA` 是安装可共享数据的命令。
- `INSTALL_MAN` 是安装手册和其他文档的命令（注意它并不会执行操作）。

有些宏展开后基本上都是包含适当参数的 `install` 命令。

5.14.2. 可执行文件和库文件做脱模 (strip) 操作

除非不得不行，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脱模，但 `INSTALL_PROGRAM` 宏会在安装的同时对其行脱模（参看下一节的内容）。`INSTALL_LIB` 宏

如果需要某一文件行脱模，但不希望使用 `INSTALL_PROGRAM` 及 `INSTALL_LIB` 宏，可使用 `${STRIP_CMD}` 来写程序。一般而言这在 `post-install` target 中行。例如：

```
post-install:
    ${STRIP_CMD} ${PREFIX}/bin/xd1
```

可以使用 `file(1)` 命令来检查所安装的可执行文件是否行脱模。如果它没有输出 `not stripped` 的提示，则表示已做脱模了。另外，`strip(1)` 不会对已脱模的文件重新脱模，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，例如，将整个目录从 `WRKSRC` 复制到 `PREFIX` 的目录。

有些情况，系统提供了两个宏。使用这些宏，而不是直接使用 `cp` 的原因是它们能保持目录文件的属主和权限。第一个宏，`COPYTREE_BIN` 将所有安装的文件复制到可执行文件，因而符合安装文件到 `PREFIX/bin`。第二个宏，`COPYTREE_SHARE`，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 `port` 指定的安装示例文件的位置。

```
post-install:
    ${MKDIR} ${DATADIR}/summer
    (cd ${WRKSRCDIR}/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。

通过设置 `COPYTREE_*` 宏的第三个参数，你可以用 `find` 指定其他的参数。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 ${WRKSRCDIR}/examples/ && \
        ${COPYTREE_SHARE} \* ${EXAMPLESDIR} "! -name Makefile")
```

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

5.14.4. 安装附加的文档

如果你的软件包含了标准的手册和 info 手册以外的文档，而且它可能会用，把一些文档安装到 PREFIX/shared/doc 下。和前面类似，也可以在 `post-install` target 中完成。

你的 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，如果你不同版本的 port 可能会同时安装，也可以用完整的 `PKGNAME`。

另外，是否安装取决于变量 `NOPORTDOCS` 的设置，你可以就在 /etc/make.conf 中禁止安装它。例如：

```
post-install:
    .if !defined(NOPORTDOCS)
        ${MKDIR} ${DOCSDIR}
        ${INSTALL_MAN} ${WRKSRCDIR}/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** 的那些文件，而不影响标准的联机手册以及 info 手册的安装。安装到 **DATADIR** 和 **EXAMPLESDIR** 的文件相应地受 **NOPORTDATA** 和 **NOPORTEXAMPLES** 控制。

有些量也会被输出到 **PLIST_SUB** 中。只要可能，它的值就将在那里以相对于 **PREFIX** 的路径形式输出。也就是，**share/doc/PORTNAME** 在装箱中默认情况下会替掉 **%DOCSDIR%**，等等。（更多的 pkg-plist 代码可以在 [这里](#) 找到。）

所有非无条件安装的文件和目录，都将在 pkg-plist 输出，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 中逐个列出文件，port 也可以将 **PORTDOCS** 置为一个文件及其 shell glob 模式，通过这种方式来加入到最后的装箱中。这个名字是相对于 **DOCSDIR** 的。因此，使用了 **PORTDOCS**，并将文件安装到非标准位置的 port，相应地置 **DOCSDIR**。如果有在 **PORTDOCS** 中列出目录，或者一个量中的 glob 模式匹配到了目录，整个子目录中的文件和目录，都将被注册到最后的装箱中。如果定义了 **NOPORTDOCS**，**PORTDOCS** 中定义的文件和目录将不被安装或加入装箱。是否安装文件到前面所指的 **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，关于 **DATADIR** 和 **EXAMPLESDIR** 的量分别是 **PORTDATA** 和 **PORTEXAMPLES**。



也可以使用 pkg-message 文件，来在安装时显示一些信息。参看 [关于使用 pkg-message 的章节](#) 以了解更多的情况。需要明确的是，并不需要把 pkg-message 加到 pkg-plist 中。

5.14.5. 子目录

尽可能 port 将它创建的文件，放置到 **PREFIX** 中正的位置。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，这是不好的。另外，许多 port 会把除了可执行文件、配置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，将被放到下列位置之一：**etc**（安装/配置文件）、**libexec**（由系统内部使用的可执行文件）、**sbin**（超用户/管理提供的可执行文件）、**info**（用于 info 阅读器的文件）或 **share**（平台无用的其它文件）。参看 [hier\(7\)](#) 以了解更多的情况；而 **/usr** 的那些目录，同样也用于 **/usr/local**。例外情况是那些需要和 USENET "news" 打交道的 port，它可以采用 **PREFIX/news** 作为文件的目的地。

Chapter 6. 特殊情况

有一些在构建port的特殊情况，我在这里提一下。

6.1. 共享

如果的port安装了一个或多个共享,那确定一个 `USE_LDCONFIG` make 量, 在`post-install`把它注册
共享 冲会去用`bsd.port.mk`去行 `${LDCONFIG} -m`来指向新的安装目录。(通常是 `PREFIX/lib`) 同,
也可以当的在的 `pke-plist`文件 中定一`@exec /sbin/ldconfig -m`和`@unexec /sbin/ldconfig -R`, 那
用可以在安装后上 就能使用, 并且在卸包后系也不会些共享仍然存在。

```
USE_LDCONFIG= yes
```

如果需要把共享安装在缺省的位置之外, 可以通确定 make 量 `USE_LDCONFIG` 来改默认的安装路径,
它包含安装共享的目录列表 例如: 如果的共享安装到 `PREFIX/lib/foo` 和 `PREFIX/lib/bar` directories目,
可以在的 Makefile中置:

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

必仔, 通常是完全不必要的, 或者可以通 `-rpath` 或在接置 `LD_RUN_PATH` 来避免 (参
[lang/moscow_ml](#) 出的例子), 或者用一个 shell 封装程序来在行可行文件之前置 `LD_LIBRARY_PATH`,
似 [www.seamonkey](#) 那。

当在 64-位系上安装 32-位 的函数, 使用 `USE_LDCONFIG32`。

尽量将共享版本号保持 `libfoo.so.0` 的格式。我的行境接器只会主 (第一个) 版本数字。

如果在更新 port 升了其的主版本号, 其它所有接了受影响的 port 的 `PORTREVISION` 都, 以
制它采用新版本的重新。

6.2. Ports 的行限制

多, 并且其中的一些致力于 限制的用程序能被打包, 是否能用于售利等等。



做一名porter有去件的 并且保FreeBSD 目不必通FTP/HTTP 或CD-ROM重新布源或的二制而解 什。如果有任何疑问, 系 [FreeBSD ports 件列表](#)。

于情况, 就可以置以下描述 的量。

6.2.1. NO_PACKAGE (禁止果打包)

个量表示我可能不能生成个用 程序的二制文件。例如, 他的不允许 二制文件的再次
行, 或者他可能禁止从丁的源代打包的行。

不管, port的 `DISTFILES` 可以 随意的像到FTP/HTTP。除非`NO_CDROM` 量也被置, 件包也可以

行在一个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，并且它个用件常要 从源代码。例如：当它个用件在 的时候要在配置信息中指定特定的硬件 代码，可以置`NO_PACKAGE`。

`NO_PACKAGE`置成字符串 来描述它什它个件 不能打包。

6.2.2. `NO_CDROM` (禁止以 `CDROM` 行包)

它个量指出虽然我允 生成二进制包，但也我既不能把 个 件包也不能把port的`DISTFILES` 放在光（或类似的媒介）上售。但不管，二进制包和port的`DISTFILES` 可以从FTP/HTTP上得。

如果它个量和 `NO_PACKAGE`一起被置，那它个port的`DISTFILES` 将只能从FTP/HTTP上得。

`NO_CDROM` 被置成一个字符串 来描述它什它个port不能重新布在CD-ROM上。 例如：如果它个port的 是用于“非商业”，那它个量就能置了。

6.2.3. `NOFETCHFILES` (不自取指定的文件)

在 `NOFETCHFILES` 量中定的文件，不会自 从 `MASTER_SITES` 取。一典型的用例是， 使用来自某个件供 商提供的 CD-ROM 上的文件。

用于在 `MASTER_SITES` 上是否包含了所需文件的工具， 忽略些文件， 而不是告它不存在。

6.2.4. `RESTRICTED` (禁止任何形式的再分)

如果用程序既不允 像其 `DISTFILES`， 也不允 布其 版本的包， 置它就可以了。

`NO_CDROM` 或 `NO_PACKAGE` 不 与 `RESTRICTED` 同置， 因它包含了些情形。

`RESTRICTED` 置一个明 port 何不能布的串。 典型情况可能是由于 port 包含了有的件， 因而用 需要自行下 `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)

当置了 `RESTRICTED` 或 `NO_CDROM` ， 它个量会默置 `${DISTFILES} ${PATCHFILES}`， 否它会空。 如果只有某些源包文件是受限的， 可以用它个量来指明它。

注意， port committer 在 /usr/ports/LEGAL 中 一个源包文件撰写 的目， 并介些限制的原因。

6.3. 机制

6.3.1. Ports 的并行

FreeBSD ports 框架支持使用多个 `make` 子程来行并行， 在 SMP 上可以全面地利用系的 CPU 算能力， 令 port 的 程更快、更有效率。

目前是通向原作者的代码 `make(1)` 参数 `-jX` 来的。 憾的是， 并不是所有的 port 都能很好地理 个。 因此， 必通明地在 Makefile 中指定 `MAKE_JOBS_SAFE=yes` 来用一功能。

从 port 人的角度有一个控制的方法是置 `MAKE_JOBS_UNSAFE=yes` 量。 个量主要是用于已知不能与 `-jX` 配合使用的 port, 即使用在 `/etc/make.conf` 中定了 `FORCE_MAKE_JOBS=yes` 量, 系也不会使用并行。

6.3.2. make、gmake, 以及 imake

如果 port 用到了 GNU make, 置 `USE_GMAKE=yes`。

表 4. 与 gmake 有 的 port 量

量	意
<code>USE_GMAKE</code>	此 port 需要使用 gmake 来完成程。
<code>GMAKE</code>	不在 <code>PATH</code> 中, gmake 的完整路径。

于 X 用程序的 port, 如果它使用 imake 根据 Imakefile 文件来生成 Makefile, 置 `USE_IMAKE=yes`。 会使程中的配置 (configure) 段自行 `xmkmf -a`。 如果 `-a` 志会的 port 来麻, 需置 `XMKMF=xmkmf`。 如果 port 用到了 imake 但并不使用 `install.man` target, 置 `NO_INSTALL_MANPAGES=yes`。

如果 port 源文件的 Makefile 的主 target 是 `all` 以外的名字, 地置 `ALL_TARGET`。 于 `install` 而言, 的量是 `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, 需要置 `GNU_CONFIGURE=yes`。 如果希望外的参数 `configure` 脚本 (默认参数 `--prefix=${PREFIX} --infodir=${PREFIX}/${INFO_PATH} --mandir=${MANPREFIX}/man --build=${CONFIGURE_TARGET}`), 通 `CONFIGURE_ARGS` 来指定些参数。 似地, 可以通 `CONFIGURE_ENV` 量来一些境量。

如果的件包使用 GNU `configure`, 而生成可文件命名方式 "怪" 如 `i386-portbld-freebsd4.7-` 用程序名, 需要更一地通改 `CONFIGURE_TARGET` 量来按照新版本的 `autoconf` 生成的脚本所希望的方式指定 target。 其方法是, 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 的量

量	意
<code>GNU_CONFIGURE</code>	此 port 需要用 <code>configure</code> 脚本来准。
<code>HAS_CONFIGURE</code>	与 <code>GNU_CONFIGURE</code> 似, 但默认的 <code>configure</code> target 并不加入 <code>CONFIGURE_ARGS</code> 。
<code>CONFIGURE_ARGS</code>	希望 <code>configure</code> 脚本的外参数。
<code>CONFIGURE_ENV</code>	希望在行 <code>configure</code> 脚本置的境量。
<code>CONFIGURE_TARGET</code>	替默认的 <code>configure</code> target。 其默认是 <code>\${MACHINE_ARCH}-portbld-freebsd\${OSREL}</code> 。

6.3.4. 使用 `scons`

如果 `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
<code>SCONS_ARGS</code>	当前 <code>port</code> 希望 <code>SCons</code> 环境的参数。
<code>SCONS_BUILDENV</code>	希望在系统环境中设置的变量。
<code>SCONS_ENV</code>	希望在 <code>SCons</code> 环境中设置的变量。
<code>SCONS_TARGET</code>	<code>SCons</code> 的最后一个参数，类似于 <code>MAKE_TARGET</code> 。

如果希望第三方的 `SConstruct` 尊重通 `SCONS_ENV`（其中最重要的是 `CC/CXX/CFLAGS/CXXFLAGS` 配置）`SCons` 的配置，需要 `SConstruct` 行修改，使 `Environment` 按下列方式建立：

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

其后，可以通过 `env.Append` 和 `env.Replace` 来对它行修改。

6.4. 利用 GNU autotools

6.4.1. 入门

多 GNU autotools 提供了一在多重操作系统和机器架之上软件件的抽象机制。在 Ports Collection 中，`port` 可以通过的方法来使用些工具：

```
USE_AUTOTOOLS= 工具:版本[:操作] ...
```

撰写本，工具可以置 `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 `devel/{automake,autoconf,libtool}[0-9]+` 以了解有效的版本号）。

操作 是一个可的扩展，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 `+=` 。

最后，可以使用一个特殊的名 `autotools` 的工具，它会安装全部可用的 autotools 版本，以跨平台的需要。可以通过安装 `devel/autotools` `port` 来达到目的。

6.4.2. `libtool`

使用 GNU 框架的共享通常会使用 `libtool` 来整合共享的和安装，以便与所行的操作系统相匹配。通常的做法是使用用程序所附的 `libtool` 副本。如果需要使用外部的 `libtool`，可以使用 Ports 套件提供的版本：

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

如果不使用外的操作符， `libtool:版本` 表示希望框架使用 `configure` 脚本来系所安装的 `libtool` 行修。会暗含地定 `GNU_CONFIGURE`。更一， 框架会置一系列 `make` 和 `shell` 量用于 `port` 后的操作。参 `bsd.autotools.mk` 了解一的情。

如果指定了 `:env` 操作符， 表示只置境， 而跳其他的操作。

最后， `LIBTOOLFLAGS` 和 `LIBTOOLFILES` 可以用来替最常修改的参数， 以及将被 `libtool` 修的文件。多数 `port` 不需要做。参 `bsd.autotools.mk` 以了解一的。

6.4.3. libltdl

一些 `ports` 会使用 `libltdl`， 后者是 `libtool` 件包的一部分。使用个并不意味着必使用 `libtool` 本身， 因此提供了。

```
USE_AUTOTOOLS= libltdl:版本
```

目前， 置所做的全部工作是将 `LIB_DEPENDS` 置当的 `libltdl` `port`， 并作一方便的功能， 助人消除在 `USE_AUTOTOOLS` 框架以外的， 于 `autotools` `port` 的依。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 `autoconf` 模板的 `configure.ac` 文件。可以用下列置来要求 `autoconf` 建 `configure` 脚本， 并使用 `autoheader` 来 `configure` 脚本建模板文件。

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

以及

```
USE_AUTOTOOLS= autoheader:版本
```

上述置会暗含使用 `autoconf:版本`。

于 `libtool`， 置与前面似。如果指定可的 `:env` 操作符， 表示只置用于后工作的境。如果不指定， 会 `port` 行相的修和重新配置。

其它的可量， 如 `AUTOCONF_ARGS` 和 `AUTOHEADER_ARGS` 可以通 `port` 的 `Makefile` 来式地指定替。似 `libtool`， 多数 `port` 并不需要做。

6.4.5. automake 和 aclocal

某些件包只提供了 `Makefile.am` 文件。些文件必首先用 `automake` Makefile.in 并使用 `configure` 来生成 Makefile。

类似地，偶尔会有一些软件包不提供所需的 `aclocal.m4` 文件。这些文件可以通过使用 `aclocal` 来生成 `configure.ac` 或 `configure.in` 自生成。

`aclocal` 与 `automake` 有和 `autoheader` 与 `autoconf` 在前面一章中所介绍的相似的关系。`aclocal` 会隐含使用 `automake`，因此：

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

和

```
USE_AUTOTOOLS= aclocal:版本
```

也自隐含使用 `automake:版本`。

与 `libtool` 类似，`autoconf` 如果使用了可选项的 `:env` 操作符表示选项置于后使用的环境，如果不选项，则会 `port` 行重新配置。

对于 `autoconf` 和 `autoheader` 而言，`automake` 和 `aclocal` 提供了选项的可选项参数量 `AUTOMAKE_ARGS` 和 `ACLOCAL_ARGS`，如果需要的，可以在 `port` 的 `Makefile` 中指定。

6.5. 使用 GNU `gettext`

6.5.1. 基本用法

如果 `port` 需要使用 `gettext`，只要将 `USE_GETTEXT` 选项置 `yes`，`port` 就会添加 `devel/gettext` 的依赖。`USE_GETTEXT` 也可以指定所需的 `libintl` 的版本，它是 `gettext` 的基本组成部分，尽管如此，强烈建议不要使用该功能：`port` 能与目前版本的 `devel/gettext` 配合工作。

在 `port` 中相当常见的情况下，会需要同使用 `gettext` 和 `configure`。一般而言，GNU `configure` 能自定位到 `gettext`。如果它没有成功地完成工作，可以通过类似下面选项的 `CPPFLAGS` 和 `LD_FLAGS` 将 `gettext` 的位置告诉它：

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

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

当然，不需要选项参数 `configure`，代码可以更简洁：

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

6.5.2. 可选项用法

一些软件包提供了禁用 NLS 的能力，例如，在 `configure` 脚本，指定 `--disable-nls` 参数。如果 port 的软件支持该配置，那么根据 `WITHOUT-NLS` 的设置来有条件地使用 `gettext`。对于比该选项和不太好的 port，可以使用下列脚本：

```
GNU_CONFIGURE=    yes

.if !defined(WITHOUT-NLS)
USE_GETTEXT=      yes
PLIST_SUB+=       NLS=""
.else
CONFIGURE_ARGS+=  --disable-nls
PLIST_SUB+=       NLS="@comment "
.endif
```

要做的下一件事是合理地安排装箱文件，使其根据用选项配置来决定是否将消息文件（message catalog）文件放入最合适的装箱。前面已介绍了在 Makefile 中所需的写法，该做法在 [高 pkg-plist 用法](#) 中进行了介绍。因此，在 `pkg-plist` 中输出的 `%%NLS%%` 均会在禁用 NLS 时自替换为“``@comment``”，反之则替换为空串。因此，在最合适的装箱中 `%%NLS%%` 的行，在 NLS 关闭的情况下就会注释掉，反之，则些前就会自删除。在需要做的事情就是把 `%%NLS%%` 到 `pkg-plist` 中的消息文件的那些行，例如：

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

在比该选项的情形中，可能需要使用更高深的技巧，例如 [生成装箱](#) 等。

6.5.3. 处理消息目录

在安装消息文件时有一个需要注意的地方。有些文件会放到 `LOCALBASE/shared/locale` 下与语言的目录中，这些目录一般的 port 不需要创建和删除。最常用的语言的目录已在 `/etc/mtree/BSD.local.dist` 中列出；也就是，它是基本系统的一部分。其他一些语言的目录，则由 `devel/gettext` 控制。最好看一下 `pkg-plist`，以确定是否正在安装某个不常用语言的文件。

6.6. 使用 perl

如果 `MASTER_SITES` 包含 `MASTER_SITE_PERL_CPAN`，那么尽量把 `MASTER_SITE_SUBDIR` 设置成目录的名字。例如，对于 `p5-Module-Name` 而言推荐的名字是 `Module`。可以在 [cpan.org](#) 得到该目录的名字。可以保存在模块的作者生成化，保持 port 可用。

以上有一个例外，即该目录不存在或源包不在这个目录中，允许使用作者的 id 作为 `MASTER_SITE_SUBDIR`。

所有 些 均同 接受 YES 和版本串， 似 5.8.0+ 的写法。使用 YES 表示 port 能 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， 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 量

量	意
USE_PERL5	表示 port 将 perl 5 用于 和 行。
USE_PERL5_BUILD	表示 port 将 perl 5 用于 。
USE_PERL5_RUN	表示 port 将 perl 5 用于 行。
PERL	perl 5 的完整路径，可能是系 自 的，或者从 port 安装，但没有版本号。如果 需要在脚本中替 “#!” 行， 使用 个 量。
PERL_CONFIGURE	采用 Perl 的 MakeMaker 行配置。 一 量 含 置 USE_PERL5。
PERL_MODBUILD	使用 Module::Build 行配置、 并安装。 一 量 含 置 PERL_CONFIGURE。



Perl 模 通常并没有官方网站， 些 port 将 cpan.org 作 其 pkg-descr WWW 行的内容。 推 的 URL 格式 <http://search.cpan.org/dist/Module-Name/>（保留最后的斜 ）。

6.7. 使用 X11

6.7.1. X.Org 件

在 Ports 套件中提供的 X11 是 X.Org。如果 的 用程序用到了 X 件， 将 USE_XORG 所需要的那些 件。目前可用的 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 到。

The Mesa Project 是一个致力于自由的 OpenGL 的 。 可以使用 USE_GL 量来 port 依 其不同的 件。 可用的 包括： glut, glu, glw, glew, gl 和 linux。 了 向前兼容， 当使用 yes 系 会自 将其映射 glu。

例 12. 使用 *USE_XORG* 的例子

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

多个 ports 会定义 *USE_XLIB*，这会导致 port 依赖 50 多个间接依赖。由于它出于 X.org 模块化之前，因此这个数量向前兼容的原因提供，新的 port 不再使用它。

表 8. 用到 X 的 *port* 可以使用的变量

<i>USE_XLIB</i>	此 port 用到了 X 库。已禁用 - 不使用 <i>USE_XORG</i> 变量列出用到的 X.Org 组件，而不是使用这个变量。
<i>USE_IMAKE</i>	此 port 用到了 <i>imake</i> 。
<i>USE_X_PREFIX</i>	已禁用。目前其作用与 <i>USE_XLIB</i> 相同，并可以直接用后者替代。
<i>XMKMF</i>	设置 <i>xmkmf</i> 的完整路径名，如果它不在 <i>PATH</i> 中的。默认是 <i>xmkmf -a</i> 。

表 9. 用于表示 X11 某些组件的依赖关系的变量

<i>X_IMAKE_PORT</i>	用以提供 <i>imake</i> 以及许多其它用于 X11 的工具的 port。
<i>X_LIBRARIES_PORT</i>	用以提供 X11 库的 port。
<i>X_CLIENTS_PORT</i>	用以提供 X 客户端的 port。
<i>X_SERVER_PORT</i>	用以提供 X 服务器的 port。
<i>X_FONTSERVER_PORT</i>	用以提供字体服务的 port。
<i>X_PRINTSERVER_PORT</i>	用以提供打印服务的 port。
<i>X_VFBSERVER_PORT</i>	用以提供在虚拟帧缓存器(virtual framebuffer server)的 port。
<i>X_NESTSERVER_PORT</i>	用以提供嵌套 X 服务器的 port。
<i>X_FONTS_ENCODINGS_PORT</i>	用以提供字体提供者的 port。
<i>X_FONTS_MISC_PORT</i>	用以提供多位字体的 port。
<i>X_FONTS_100DPI_PORT</i>	用以提供 100dpi 位字体的 port。
<i>X_FONTS_75DPI_PORT</i>	用以提供 75dpi 位字体的 port。
<i>X_FONTS_CYRILLIC_PORT</i>	用以提供西里字母位字体的 port。
<i>X_FONTS_TTF_PORT</i>	用以提供 TrueType® 字体的 port。
<i>X_FONTS_TYPE1_PORT</i>	用以提供 Type1 字体的 port。
<i>X_MANUALS_PORT</i>	用以提供面向用户的计算机手册的 port。

```
# 使用某些 X11 并依赖于字体服务和西里字体。
RUN_DEPENDS=    ${LOCALBASE}/bin/xf86:${X_FONTSERVER_PORT} \

${LOCALBASE}/lib/X11/fonts/cyrillic/crox1c.pcf.gz:${X_FONTS_CYRILLIC_PORT}

USE_XORG=       x11 xpm
```

6.7.2. 需要使用 Motif 的 `port`

如果 `port` 需要 Motif，则在 `Makefile` 中定义 `USE_MOTIF`。默认的 Motif 是 `x11-toolkits/openmotif`。用户可以通过置 `WANT_LESSTIF` 变量来用 `x11-toolkits/lesstif` 代替它。

`bsd.port.mk` 会将 `MOTIFLIB` 变量置到合适的 Motif 的引用。使用 `port` 中将 `Makefile` 或 `Imakefile` 提到 Motif 的地方改 `${MOTIFLIB}`。

有以下几种情况：

- 如果 `port` 中将 Motif 在其 `Makefile` 或 `Imakefile` 表 `-lXm`，则将其替换 `${MOTIFLIB}`。
- 如果 `port` 在其 `Imakefile` 中使用 `XmClientLibs`，则将其改 `${MOTIFLIB} ${XTOOLLIB} ${XLIB}`。

注意 `MOTIFLIB` (通常) 会展 `-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 X Window 系统安装字体，将字体放到 `LOCALBASE/lib/X11/fonts/local`。

6.7.4. 通过 `Xvfb` 来获得虚的 `DISPLAY`

某些应用程序必须在有可用的 X11 显示的时候才能成功。当机器没有控制台，则会来。了解这个问题，如果定义了适当的变量，则基本措施会采用虚存的 X server。此时，程序中将会出可用的 `DISPLAY`。

```
USE_DISPLAY=    yes
```

6.7.5. 桌面

通过利用 `DESKTOP_ENTRIES` 变量，可以很容易地在 `port` 中建立桌面 (`Freedesktop` 标准)。这些会在类似 GNOME 或 KDE 的符合一标准的桌面环境中显示在应用程序菜单中。它们会自建、安装 `.desktop` 文件，并将其加入 `pkg-plist`。其法：

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

可以在 [Freedesktop 网站](#) 上找到可用的分桌名称。 `StartupNotify` 表示应用程序在支持该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}/extra/graf/tome-128.png" \
                  "tome -v -g" "Application;Game;RolePlaying;" \
                  false
```

6.8. 使用 GNOME

FreeBSD/GNOME 项目使用自己的变量来定义 port 所使用的 GNOME 软件。 [这些变量的列表](#) 可以在 FreeBSD/GNOME 项目的主页上找到。

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

<code>USE_QT_VER</code>	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，系统会自动从 <code>configure</code> 脚本和 <code>make</code> 命令提供必要的参数。
<code>QT_PREFIX</code>	该变量会自动从 Qt 的安装路径（只读变量）。
<code>MOC</code>	该变量会自动从 <code>moc</code> 的路径（只读变量）。默认与 <code>USE_QT_VER</code> 变量的值有。
<code>QTCPPFLAGS</code>	通过 <code>CONFIGURE_ENV</code> 向 Qt 工具套件的参数。默认配置与 <code>USE_QT_VER</code> 有。
<code>QTCFGLIBS</code>	通过 <code>CONFIGURE_ENV</code> 向 Qt 工具套件的接口。默认配置与 <code>USE_QT_VER</code> 有。
<code>QTNONSTANDARD</code>	禁止系统自动修改 <code>CONFIGURE_ENV</code> 、 <code>CONFIGURE_ARGS</code> 和 <code>MAKE_ENV</code> 。

表 11. 其他用于使用 Qt 4.x 的变量

<code>QT_COMPONENTS</code>	用于指定 Qt4 工具和函数库的依赖。详情后。
<code>UIC</code>	该变量会自动从 <code>uic</code> 的路径（只读变量）。默认与 <code>USE_QT_VER</code> 有。
<code>QMAKE</code>	该变量会自动从 <code>qmake</code> 的路径（只读变量）。其默认与 <code>USE_QT_VER</code> 有。
<code>QMAKESPEC</code>	该变量会自动从 <code>qmake</code> 配置文件的路径（只读变量）。其默认与 <code>USE_QT_VER</code> 有。

当设置了 `USE_QT_VER`，系自会 `configure` 脚本一系列有用的参数：

```
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` 4， 会行下列配置：

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

6.9.2. 件的 (限 Qt 4.x)

当把 `USE_QT_VER` 4， 就可以通 `QT_COMPONENTS` 量来指定 Qt4 工具和函数的依了。 通在 的名称后面添加 `_build` 或 `_run` 的后， 可相地将依系限于或行刻。 在没有指定后， 系 默在和行刻均依件。 通常情况下在指明函数一的件不使用后， 工具件使用 `_build` 后， 而件件， 使用 `_run` 后。 下表中列出了一些最常用的件（全部可用的件， 在 `/usr/ports/Mk/bsd.qt.mk` 中的 `_QT_COMPONENTS_ALL` 列出）：

表 12. 可用的 Qt4 函数件

名字	描述
corelib	核心 (在 port 只使用 corelib 而没有用到其他 可以省略)
gui	形用界面
network	网函数
opengl	OpenGL 函数
qt3support	Qt3 兼容支持函数
qtestlib	元函数
script	脚本函数
sql	SQL 函数
xml	XML 函数

可以通在成功之后， 通在主可行文件上行 `ldd` 来定所需的。

表 13. 可用的 Qt4 工具件

名字	描述
moc	元象器 (几乎所有的 Qt 用程序在 程中都需要它)

名字	描述
<code>qmake</code>	Makefile 生成器 / 工具
<code>rcc</code>	源器 (如果应用程序中包含 <code>.rc</code> 或 <code>.qrc</code> 文件, 就需要它)
<code>uic</code>	界面器 (如果应用程序中包含使用 Qt Designer 建的 <code>*.ui</code> 文件就需要它 - 一般来 Qt 程序都会使用 GUI 的)

表 14. 可用的 Qt4 组件

名字	描述
<code>iconengines</code>	SVG 引擎 (如果程序使用 SVG)
<code>imageformats</code>	用于 GIF、JPEG、MNG 和 SVG 的 imageformat 件 (如果程序使用片文件)

例 14. Qt4 组件

在这个例子中, 我将要移植的程序用到了 Qt4 形用界面函数、 Qt4 核心 (core) 函数、 所有 Qt4 代生成工具以及 Qt4 的 Makefile 生成器。 由于 `gui` 函数会自附核心函数的依, 因此并不需要明确指出需要 `corelib` 的依系。 Qt4 代生成工具 `moc`、 `uic` 和 `rcc` 以及 Makefile 生成器 `qmake` 只在程中才会用到, 因此可以指定 `_build` 后:

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

6.9.3. 其他考

如果程序没有提供 `configure` 文件, 而是了一个 `.pro` 文件, 则:

```
HAS_CONFIGURE= yes

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

注意, 与系提供的 `BUILD.sh` 中的 `qmake` 似。 `CONFIGURE_ENV` 能保 `qmake` 可以看到 `QMAKESPEC` 量, 否它可能无法正常工作。 `qmake` 会生成准的 Makefile, 因此无需自行写 `build target`。

Qt 程序通常会写能跨平台使用, 通常 X11/Unix 并不是它的平台, 有会致一些角角的, 例如:

- 缺少必要的 `includepaths`。 多程序会使用托支持, 但忽略了些或文件需要在 X11 目中。
- 可以通命令行告 `qmake` 将些文件和函数加入到搜索路径中, 例如:

```

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

```

- 有`~/.config`的安装路径。 有`~/.local`， 似`~/.local`或 `~/.desktop` 文件的一些数据， 默`~/.local`情况下没有安装到 `~/.local` XDG-兼容的程序会`~/.local`描的路径中。 [editors/texmaker](#) 就是一个`~/.local`的例子 - 参考`~/.local`个 `port` 的 `files` 目`~/.local`中的 `patch-texmaker.pro`， 以了解如何在 Qmake 工程文件中修正`~/.local`个`~/.local`。

6.10. 使用 KDE

6.10.1. 量定`~/.local` (只用于 KDE 3.x)

表 15. 用于使用 KDE 3.x 的 `port` 的量

<code>USE_KDELIBS_VER</code>	表示 <code>port</code> 用到了 KDE <code>~/.local</code> 。 个 <code>~/.local</code> 量可以指定希望使用的 KDE 主版本号， 如果 <code>~/.local</code> 置了 <code>~/.local</code> 个量， <code>~/.local</code> 系 <code>~/.local</code> 也会将 <code>USE_QT_VER</code> <code>~/.local</code> 当的版本。 量目前唯一有效的 <code>~/.local</code> 是 <code>3</code> 。
<code>USE_KDEBASE_VER</code>	表示 <code>port</code> 用到了 KDE 的基本系 <code>~/.local</code> 。 个 <code>~/.local</code> 量可以指定希望使用的 KDE 主版本号， 如果 <code>~/.local</code> 置了 <code>~/.local</code> 个量， <code>~/.local</code> 系 <code>~/.local</code> 也会将 <code>USE_QT_VER</code> <code>~/.local</code> 当的版本。 量目前唯一有效的 <code>~/.local</code> 是 <code>3</code> 。

6.10.2. 用于 KDE 4 的量定`~/.local`

如果`~/.local`的`~/.local`用程序需要使用 KDE 4.x， `~/.local`将 `USE_KDE4` `~/.local`所需`~/.local`件的列表。 下面列出一些最常用到的`~/.local`件 (最新的`~/.local`件列表位于 `/usr/ports/Mk/bsd.kde4.mk` 中的 `_USE_KDE4_ALL`)：

表 16. 可用的 KDE4 `~/.local`件

名称	明
<code>akonadi</code>	个人信息管理 (PIM)存 <code>~/.local</code> 服 <code>~/.local</code>
<code>automoc4</code>	令 <code>port</code> 使用 <code>automoc4</code> <code>~/.local</code> 工具集
<code>kdebase</code>	基本的 KDE <code>~/.local</code> 用程序 (Konqueror、Dolphin、Konsole)
<code>kdeexp</code>	<code>~/.local</code> 性的 KDE <code>~/.local</code> (包含尚未完全 <code>~/.local</code> 定不 <code>~/.local</code> 的 API)
<code>kdehier</code>	常用的 KDE 目 <code>~/.local</code> 次 <code>~/.local</code>
<code>kdelibs</code>	基本 KDE <code>~/.local</code>
<code>kdeprefix</code>	如果 <code>~/.local</code> 置了 <code>~/.local</code> 个 <code>~/.local</code> ， <code>~/.local</code> <code>port</code> 将安装到 <code>~/.local</code> {KDE4_PREFIX} <code>~/.local</code> 而不是 <code>~/.local</code> {LOCALBASE}
<code>pimlibs</code>	PIM 函数 <code>~/.local</code>
<code>workspace</code>	用于 <code>~/.local</code> 成 <code>~/.local</code> 面的 <code>~/.local</code> 用程序和函数 <code>~/.local</code> (Plasma、KWin)

KDE 4.x `port` 会安装到 `~/.local`{KDE4_PREFIX}`~/.local`， 目前是 `/usr/local/kde4`， 以避免与 KDE 3.x `ports` 冲突。 `~/.local`是通`~/.local`指定 `kdeprefix` `~/.local`件来`~/.local`的， 它表示替`~/.local`默`~/.local`的 `PREFIX`。 不`~/.local`， `port` 仍会遵循通`~/.local` `MAKEFLAGS` `~/.local`境`~/.local`量`~/.local`置的

PREFIX 以及其它 make 参数。

KDE 4.x ports 有可能和 KDE 3.x ports 冲突，因此如果你用了 `kdeprefix` 选项，它会安装到 `${KDE4_PREFIX}`。目前 `KDE4_PREFIX` 的默认是 `/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，它会替 `kdeprefix` 提供的配置。

例 15. `USE_KDE4` 示例

下面是一个 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` 引入 KDE 函数，并令 port 在代码段使用 `automoc4`。需要的 KDE 选项，以及其他依赖的选项可以从 `configure` 的日志中可知。 `USE_KDE4` 并不会自动设置 `USE_QT_VER`。如果 port 需要使用某些 Qt4 选项，需要设置 `USE_QT_VER` 并指定所需要的选项。

```
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

如果你的 port 需要 Java™ 包 (JDK™) 来完成、支持运行，甚至完成解包源代码包的工作，就定义 `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。如果你的 port 必须使用其中的某个特定的版本，也可以予以定义。最新的定义版本是 [java/jdk16](#)。

表 17. 用到 Java 的 port 可以使用的变量

变量	意义
<code>USE_JAVA</code>	只有定义它才能使其它变量生效。
<code>JAVA_VERSION</code>	用空格分隔的符合 port 使用的 Java 版本。可选的 "+" 可以用于指定某个版本 (可以用： <code>1.5[+]</code> <code>1.6[+]</code> <code>1.7[+]</code>)。
<code>JAVA_OS</code>	用空格分隔的 port 的 JDK port 操作系统型 (可以用： <code>native linux</code>)。
<code>JAVA_VENDOR</code>	用空格分隔的 port 的 JDK port 供应商 (可以用： <code>freebsd bsdjvm sun openjdk</code>)。
<code>JAVA_BUILD</code>	设置这个变量表示所定义的 JDK port 被列入 port 的依赖系。
<code>JAVA_RUN</code>	设置这个变量表示所定义的 JDK port 被列入 port 的运行环境依赖系。

变量	意义
<code>JAVA_EXTRACT</code>	设置该变量表示所用的 JDK port 被列入 port 的解包支持依赖。

下面是在设置了 `USE_JAVA` 之后，port 能从系中获得的配置：

表 18. 向使用了 *Java* 的 port 提供的变量

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

可以使用 `java-debug make target` 以取得用于 `port` 的信息。大多数前述量的皆会予以呈。

此外，会定下述常量，以保所有的 Java port 均以一致之方式安装：

表 19. 使用 Java 的 `port` 定义的常量

常量	
<code>JAVASHAREDIR</code>	所有 Java 相关料的安装根目。默认： <code>\${PREFIX}/shared/java</code> .
<code>JAVAJARDIR</code>	用以安装 JAR 文件的目。默认： <code>\${JAVASHAREDIR}/classes</code> .
<code>JAVALIBDIR</code>	其它 port 安装的 JAR 文件所在的目。默认： <code>\${LOCALBASE}/shared/java/classes</code> .

相关的也会定在 `PLIST_SUB` (在 根据 `make` 量 `pkg-plist` 行修改 中行介) 和 `SUB_LIST` 中。

6.11.2. 采用 Ant 行

如果 port 采用 Apache Ant 行，需要定 `USE_ANT`。如是，Ant 将作子-make 命令来使用。如果 port 未定 `do-build` target，将默认依 `MAKE_ENV`、`MAKE_ARGS` 和 `ALL_TARGET`。的置行 Ant。似于 机制 中介的于 `USE_GMAKE` 的机制。

6.11.3. 最佳践

如果正移植某个 Java ，的 port 把 JAR 文件安装到 `${JAVAJARDIR}`，而其它文件放在 `${JAVASHAREDIR}/${PORTNAME}` 下 (除了文，参下文)。要少打包文件的尺寸，可以直接在 Makefile 中引用些 JAR 文件，具体做法是使用下面的句 (此的 `myport.jar` 是作 port 一部分安装的 JAR 文件的名字)：

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

移植 Java 用程序，port 通常会希望将所有文件安装到同一目 (包括其依的 JAR)。烈建使用 `${JAVASHAREDIR}/${PORTNAME}`。移植件的入，可以自行决定是否将所依的其它 JAR 安装到此目，或直接使用已装好的那些 (来自 `${JAVAJARDIR}`)。

无正制作一的 port (或者用程序)，附加的文都安装到和其它 port 同的位置。已知道，JavaDoc 会根据 JDK 版本的不同而生不同的文件。于那些不打算制使用某一特定版本 JDK 的 port 而言，无疑提高了制作装箱 (`pkg-plist`) 的度。是什烈建使用 `PORTDOCS` 宏的原因。更一，即使能 `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` 的默认是 `${PREFIX}/shared/${PORTNAME}`。Java port 而言将 `DATADIR` 改 `${JAVASHAREDIR}/${PORTNAME}` 是一个好主意。当然，`DATADIR` 会自加到 `PLIST_SUB` 中 (在 根据 `make` 量 `pkg-plist` 行修改 有所介) 因此可以在 `pkg-plist` 中直接使用 `%DATADIR%`。

撰写本文，是从源代，是直接安装 Java ports 安装包并没有明的定。尽管如此，FreeBSD Java Project 的入仍鼓励移植件的者在不麻的情况下尽可能从源代完成。

本中所介的全部特性，均是在 `bsd.java.mk` 中的。如果感自己的 port 需要更的 Java 支持，

首先参 [bsd.java.mk CVS 日志](#)，因通常撰文介最新特性需要一些。此外，如果所缺少的支持多其它 Java port 亦属有益，在 `freebsd-java` 其行。

在 PR 中的 `java`，主要是用于 FreeBSD Java project 移植 JDK 本身之用。因而，提交的 Java port，入 `ports`，除非正解决的 JDK 本身或 `bsd.java.mk` 的。

似地，参考 [分](#) 中所述的于 `CATEGORIES` 在 Java port 中的使用。

6.12. Web 用，Apache 和 PHP

6.12.1. Apache

表 20. 用到 Apache 的 port 可以使用的量

<code>USE_APACHE</code>	此 port 需要 Apache。可用的： <code>yes</code> (任意可用版本)、 <code>1.3</code> 、 <code>2.0</code> 、 <code>2.2</code> 、 <code>2.0+</code> 、等等。默依的版本是 <code>1.3</code> 。
<code>WITH_APACHE2</code>	此 port 需要 Apache 2.0。如果没有个量，port 将依 Apache 1.3。一量目前已，因而不使用。
<code>APXS</code>	到 <code>apxs</code> 可行文件的完整路径。可以在 port 中替代。
<code>HTTPD</code>	到 <code>httpd</code> 可行文件的完整路径。可以在 port 中替代。
<code>APACHE_VERSION</code>	目前系中安装的 Apache 版本 (只量)。一量只有在引用了 <code>bsd.port.pre.mk</code> 之后才能使用，其可能的： <code>13</code> 、 <code>20</code> 、 <code>22</code> 。
<code>APACHEMODDIR</code>	Apache 模所在的文件。在 <code>pkg-plist</code> 中，一量会自展。
<code>APACHEINCLUDEDIR</code>	Apache 文件所在的文件。在 <code>pkg-plist</code> 中，一量会自展。
<code>APACHEETCDIR</code>	Apache 配置文件所在的文件。在 <code>pkg-plist</code> 中，一量会自展。

表 21. 在移植 Apache 模比有用的量

<code>MODULENAME</code>	模的名称。默 <code>PORTNAME</code> 。例如： <code>mod_hello</code>
<code>SHORTMODNAME</code>	模的略名字。默情况下会自根据 <code>MODULENAME</code> 算，但也可以自行置来替代它。例如： <code>hello</code>
<code>AP_FAST_BUILD</code>	使用 <code>apxs</code> 来和安装模。
<code>AP_GENPLIST</code>	同自建 <code>pkg-plist</code> 。
<code>AP_INC</code>	在程中，将指定的目加入到搜索文件的目中。
<code>AP_LIB</code>	在程中，将指定的目加入到搜索函数目中。
<code>AP_EXTRAS</code>	<code>apxs</code> 的外参数。

6.12.2. Web 应用

Web 应用程序安装到 `PREFIX/www/` 应用程序的名字。为方便起见，该路径在 `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` 的变量。

web 服务器程序所用的用户和组，分别以 `WWWOWN` 和 `WWWGRP` 变量的形式提供，如果需要修改某些文件的属主的。该变量的默认均为 `www`。如果的 `port` 希望使用其他，使用 `WWWOWN?=myuser` 写法，以便应用能更容易地修改它。

除非的 `port` 必需使用 `Apache`，否则不要将其写入依赖系。尊重该行的应用程序的用户 `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

<code>USE_PHP</code>	此 <code>port</code> 需要 <code>PHP</code> 。取 <code>yes</code> 将把 <code>PHP</code> 加入依赖系。此外，可以在此指定将所需要的 <code>PHP</code> 扩展模。例如： <code>pcre xml gettext</code>
<code>DEFAULT_PHP_VER</code>	在没有安装 <code>PHP</code> 自安装的 <code>PHP</code> 主版本。默认是 <code>4</code> 。可 <code>4</code> 、 <code>5</code> 之一。
<code>IGNORE_WITH_PHP</code>	此 <code>port</code> 无法与特定版本的 <code>PHP</code> 一同工作。可 <code>4</code> 、 <code>5</code> 之一。
<code>USE_PHPIZE</code>	此 <code>port</code> 将作 <code>PHP</code> 扩展模运行。
<code>USE_PHPEXT</code>	此 <code>port</code> 将作 <code>PHP</code> 扩展，且需要作扩展模注册。
<code>USE_PHP_BUILD</code>	依赖于 <code>PHP</code> 。
<code>WANT_PHP_CLI</code>	希望使用 <code>CLI</code> (命令行) 版本的 <code>PHP</code> 。
<code>WANT_PHP_CGI</code>	希望使用 <code>CGI</code> 版本的 <code>PHP</code> 。
<code>WANT_PHP_MOD</code>	希望使用 <code>Apache</code> 模版本的 <code>PHP</code> 。
<code>WANT_PHP_SCR</code>	希望使用 <code>CLI</code> 或 <code>CGI</code> 版本的 <code>PHP</code> 。
<code>WANT_PHP_WEB</code>	希望使用 <code>Apache</code> 模或 <code>CGI</code> 版本的 <code>PHP</code> 。

6.12.4. PEAR 模

移植 `PEAR` 模的程序非常。

使用 `FILES`、`TESTS`、`DATA`、`SQLS`、`SCRIPTFILES`、`DOCS` 以及 `EXAMPLES` 些变量来指明希望安装的文件。所有里列出的文件都会自安装到合的位置，并加入 `pkg-plist`。

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

```

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 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，它是通过将脚本中的 `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 要在包名上使用 `pyXY-` 前缀，以便明示它将会配合哪个 Python 版本使用。

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

表 23. 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+` 的形式来指定所希望的版本。除此之外，也可以用横杠来分隔多个版本号，以表示某个版本，例如：`2.1-2.3`

USE_PYDISTUTILS	使用 Python distutils 来完成配置、编译和安装。它包含 setup.py 的 port 而言是必需的。它会覆盖默认的 do-build 以及 do-install 两个 target。如未定义 GNU_CONFIGURE，它会改 do-configure。
PYTHON_PKGNAMEPREFIX	作 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，PYTHON_SITELIBDIR 量会非常有用。
PYTHONPREFIX_SITELIBDIR	去掉了 PREFIX 部分的 PYTHON_SITELIBDIR。尽可能在 pkg-plist 中使用 %PYTHON_SITELIBDIR%。 %PYTHON_SITELIBDIR% 的默认是 lib/python%PYTHON_VERSION%/site-packages
PYTHON_CMD	Python 解释器的命令行，包括版本号。
PYNUMERIC	将数处理扩展模块加入依赖系。
PYNUMPY	新的数计算扩展, numpy的依赖。(PYNUMERIC 目前已被作者淘汰)。
PYXML	将 XML 扩展模块加入依赖系。(由于 Python 2.0 和更高版本不再需要，因为它已成了标准件)。
USE_TWISTED	将 twistedCore 加入依赖系。也可以用个量指定所需的件，例如：web lore pair flow
USE_ZOPE	加入 Zope，一个 web 应用平台的依赖。会把 Python 依赖改 Python 2.3。此外 ZOPEBASEDIR 也会自 Zope 安装目录的位置。

完整的可用量列表，可以在 /usr/ports/Mk/bsd.python.mk 中得到。

6.14. 使用 Tcl/Tk

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

表 24. 用到 Tcl/Tk 的 port 可以使用的量

USE_TCL	表示 port 依赖于 Tcl 函数(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 量中逐个指定。
USE_TCL_BUILD	表示 port 在编译过程中需要使用 Tcl。
USE_TCL_WRAPPER	需要使用 Tcl shell 而不需要特定版本的 tclsh 的 port 可以使用个新量。系中会安装 tclsh wrapper，用可以指定所希望的 tcl shell。

WITH_TCL_VER	由用□定的、希望使用的 Tcl 版本。
UNIQUE_NAME_WITH_TCL_VER	和 WITH_TCL_VER □似，但是□ port 指定的。
USE_TCL_THREADS	需要包含□程支持的 Tcl/Tk。
USE_TK	表示 port 依赖于 Tk □ (不是 wish shell)。它同□会□含将 USE_TCL □置□相同的□。更多的描述，□参考 USE_TCL □量。
USE_TK_BUILD	与 USE_TCL_BUILD □量表□□似的含□。
USE_TK_WRAPPER	与 USE_TCL_WRAPPER □量表□□似的含□。
WITH_TK_VER	表□与 WITH_TCL_VER □量□似的含□，它同□会□含将 WITH_TCL_VER □置□相同的□。

可用的□量的完整列表，可以在 /usr/ports/Mk/bsd.tcl.mk 中□到。

6.15. 使用 Emacs

本□尚有待撰写。

6.16. 使用 Ruby

表 25. 使用 Ruby 的 port 可以使用的□量

□量	□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□量名□置□所用的 setup.rb 的文件名。通常是 install.rb。

下表展示了 ports 系□提供□ port 作者的一些□量。□□使用□些□量，□以便把文件装到合□的位置。□尽可能多地在 pkg-plist 中使用它□。□些□量不□在 port 中重新定□。

表 26. 使用 Ruby 的 port 中的一些可用的只□□量

□量	□明	示□□
RUBY_PKGNAMEPREFIX	作□ 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□□的安装路径。	/usr/local/lib/ruby/site_ruby/1.8
RUBY_SITEARCHLIBDIR	平台相□的□的安装路径。	/usr/local/lib/ruby/site_ruby/1.8/amd64-freebsd6

变量	说明	示例
<code>RUBY_MODDOCDIR</code>	模本文的安装路径。	<code>/usr/local/shared/doc/ruby18/patsy</code>
<code>RUBY_MODEXAMPLESDIR</code>	模用例的安装路径。	<code>/usr/local/shared/examples/ruby18/patsy</code>

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` 可以用于自配置 port 的依赖系，以使用类似 `devel/sdl12` 和 `x11-toolkits/sdl_gui` 些依赖 SDL 的的情形。

目前系能下列 SDL ：

- 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`，的写法将是：

```
USE_SDL=      net mixer
```

同，`net/sdl_net` 和 `audio/sdl_mixer` 所依赖的 `devel/sdl12` 也会被自地加入。

加入使用 `USE_SDL`，它将自地：

- 将于 `sdl12-config` 的依赖系加入到 `BUILD_DEPENDS`
- 将变量 `SDL_CONFIG` 加入到 `CONFIGURE_ENV`
- 将所依赖的，加入到 `LIB_DEPENDS`

要某个特定的 SDL 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

本节介绍了在 ports tree 中的 wxWidgets 的状态，以及它与 ports 系统的集成。

6.18.1. 介绍

许多不同版本的 wxWidgets 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，我们通常是将不同的版本以包含版本号后的名字安装来解决的。

我们做的一个最明显的缺点是，应用程序必须进行修改，才能得到所希望的版本。幸运的是，多数应用程序会用 `wx-config` 脚本来确定需要的库和连接器。这个脚本会随可用的版本不同而有不同的名字。主要的程序都会尊重环境变量的配置，或提供一个 `configure` 参数，用以指定用哪个 `wx-config`。如果不是的话，就需要应用程序打补丁了。

6.18.2. 版本的变量

为了使用指定版本的 wxWidgets，可以定义一个变量（如果只定义了一个，那么一个会取默认值）：

表 27. 用于设置 wxWidgets 版本的变量

变量	说明	默认值
<code>USE_WX</code>	列出哪个 port 能使用的版本	全部版本
<code>USE_WX_NOT</code>	列出哪个 port 不能使用的版本	无

下面是可用的 wxWidgets 版本，以及它们的 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 Unicode 版本，该版本可以通过 slave port 安装，与普通版本相比，它会多一个 `-unicode` 后缀，不过可以通过使用变量来处理（参见 [Unicode](#)）。

在 [用于 wxWidgets 版本的量](#) 中的量， 可以下列或由空格分隔的合：

表 29. wxWidgets 版本

明	例子
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 (版本号小的必在前)	2.4-2.6

除此之外， 有一些用以从可用的本那本中所希望的版本的量。 量也可以一版本， 而前的版本的先更高。

表 30. 用于希望的版本的 wxWidgets versions

量名	用于
WANT_WX_VER	port
WITH_WX_VER	用

6.18.3. 件

也有一些其他用， 尽管它本身并不是 wxWidgets ， 但却与之相。 些用程序可以在 [WX_COMPS](#) 量中使用， 以下是可用的件：

表 31. 可用的 wxWidgets 件

名称	明	版本限制
wx	主	无
contrib	第三方	无
python	wxPython (Python 定)	2.4-2.6
mozilla	wxMozilla	2.4
svg	wxSVG	2.6

可以个依的件， 通冒号分隔的后指定其型。 如果没有指定， 会使用默的依型（参 [默的 wxWidgets 依系型](#)）。下面是可用的型：

表 32. 可用的 wxWidgets 依型

名称	明
build	需要件， 相当于 BUILD_DEPENDS
run	行需要件， 相当于 RUN_DEPENDS
lib	和行均需要件， 相当于 LIB_DEPENDS

件的默依系型， 如下表所示：

表 33. 默的 wxWidgets 依系型

组件	依赖系统型
wx	lib
contrib	lib
python	run
mozilla	lib
svg	lib

例 17. 安装 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 从其 2.5 版开始支持 Unicode 了。在 ports 系统中，每个版本均有提供，并可以通过下列变量来设置：

表 34. 用以在 Unicode 版本的 wxWidgets 的变量

变量	说明	作用
WX_UNICODE	port 只能配合 Unicode 版本使用	port
WANT_UNICODE	port 能与普通版本配合使用，但希望使用 Unicode 版本	port
WITH_UNICODE	令 port 使用 Unicode 版本	用
WITHOUT_UNICODE	令 port 使用普通版本，如果支持的 (即未定义 WX_UNICODE)	用



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

6.18.5. 安装已安装的版本

要安装系统中安装的版本，就需要定义 WANT_WX。如果没有将其置为特定的版本，组件将包含版本后。HAVE_WX 变量在安装完成后会自动填入内容。

下面的片段可以在安装 wxWidgets 的系统中令 port 使用它，反之则作一提供。

```
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安装有 wxPython 支持，或在没有安装则作提供； wxWidgets 也是如此处理，版本皆 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 (之前需要定义用于 wxWidgets 版本的变量中的至少一个变量)。

表 35. 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 库, 系统将自动定义此变量。

6.18.7. 在 bsd.port.pre.mk 中执行处理

如果需要引用了 bsd.port.pre.mk 之后立即一些变量执行处理, 需要定义 WX_PREMK。



如果定义了 `WX_PREMK`，在此之后定义的依赖、条件和变量将不会生效，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

下面的片段以行 `wx-config` 脚本得到完整的版本号，将其放到变量中，并写一个程序示例说明了 `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，无法自动到 `wxWidgets`，而需要使用额外的参数来加以指定。可以使用 `WX_CONF_ARGS` 变量来输出些参数。

表 36. 可用于 `WX_CONF_ARGS` 的

可用	结果
<code>absolute</code>	<code>--with-wx-config=\${WX_CONFIG}</code>
<code>relative</code>	<code>--with-wx=\${LOCALBASE} --with-wx-config=\${WX_CONFIG:T}</code>

6.19. 使用 Lua

描述了在 `ports` 系统中的 Lua 的状态，以及它与 `ports` 系统的集成。

6.19.1. 介绍

许多不同版本的 Lua 和相应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，它们是通过将不同版本的文件以不同的版本号作后缀名解决的。

所做最大的一个问题是，每个程序都需要进行修改才能到它所需要的版本。不过，通过将适当的参数到解释器和连接器很容易解决这个问题。

6.19.2. 变量版本

要变量 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， 另一个会使用默认值)：

表 37. 用于变量 Lua 版本的变量

变量	说明	默认值
USE_LUA	port 能使用的 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 版本和变量的目录：

表 38. 可用的 Lua 版本

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

在 [用于变量 Lua 版本的变量](#) 中的变量， 可以设置下面的版本之一， 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 (低版本必须在前面)	5.0-5.1

除此之外， 也有一些用来从可用版本中推荐版本的其它变量。 这些变量也可以设置一个版本， 而前面的版本必须先高。

表 40. 用于推荐 Lua 版本的变量

变量名	用于
WANT_LUA_VER	port
WITH_LUA_VER	用户

例 20. 变量 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. 组件的组件

也有一些其它的组件， 尽管本身并不是 Lua 组件， 但却与它相关。 这些组件可以通过 `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	主组件	无
tolua	用于编译 C/C++ 代码的组件	4.0-5.0
ruby	Ruby 绑定	4.0-5.0



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）。

每个组件的依赖系统类型可以通过手工添加分隔符冒号的后缀来指定。 如果不指定， 组件会采用默认类型（参见 [默认的 Lua 依赖系统类型](#)）。 以下是可用的依赖系统类型：

表 42. 可用的 Lua 依赖系统类型

名字	说明
build	该组件是程序所必需的， 相当于 <code>BUILD_DEPENDS</code>
run	在运行时需要该组件， 相当于 <code>RUN_DEPENDS</code>
lib	该组件在编译和运行时都需要， 相当于 <code>LIB_DEPENDS</code>

组件的默认依赖系统类型如下：

表 43. 默认的 Lua 依赖系统类型

组件	依赖系统类型
lua	对于 4.0-5.0 是 <code>lib</code> (动态链接) 而对于 5.1 是 <code>build</code> (静态链接)
tolua	<code>build</code> (静态链接)
ruby	<code>lib</code> (动态链接)

例 21. 编译 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. 依赖系统中已安装的版本

要依赖系统中已安装的版本， 必须定义 `WANT_LUA`。 如果没有将其绑定具体的版本， 组件会包含版本后缀。 在

之后， `HAVE_LUA` 变量将指向到的版本。

例 22. 已安装的 *Lua* 版本和文件

下面是一个如果系中有安装 *Lua* 或为了使用它的 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
```

下面的片段 port 在系中已安装， 或用到了 *tolua* 和 *Lua* 支持加以安装， 版本均为 **4.0**。

```
USE_LUA=           4.0
LUA_COMPS=         tolua
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 Lua 版本的变量 中至少一个变量之后)。

表 44. 用到 *Lua* 的 port 定义的变量

变量名	说明
<code>LUA_VER</code>	将要使用的 <i>Lua</i> 版本。(例如, 5.1)
<code>LUA_VER_SH</code>	<i>Lua</i> 接口的主版本 (例如, 1)
<code>LUA_VER_STR</code>	不带点的 <i>Lua</i> 版本 (例如, 51)
<code>LUA_PREFIX</code>	安装 <i>Lua</i> (及其文件) 使用的后缀
<code>LUA_SUBDIR</code>	在 <code>\${PREFIX}/bin</code> 、 <code>\${PREFIX}/share</code> 和 <code>\${PREFIX}/lib</code> 中用于安装 <i>Lua</i> 的子目录
<code>LUA_INCDIR</code>	用以安装 <i>Lua</i> 和 <i>tolua</i> 文件的目录
<code>LUA_LIBDIR</code>	用以安装 <i>Lua</i> 和 <i>tolua</i> 文件的目录
<code>LUA_MODLIBDIR</code>	用以安装 <i>Lua</i> 模块接口 (.so) 的目录

变量名	说明
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* 到什么地方去放 Lua

下面的 *port* 片段展示了如何告诉使用的 *configure* 脚本去什么地方放 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* 中运行处理

如果你需要在引用 *bsd.port.pre.mk* 之后就得到变量，以便将其用于运行一些命令，需要定义 *LUA_PREMK*。



如果你定义了 *LUA_PREMK*，那么在引用 *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生成了。

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，将其作为一个变量，并嵌入程序。

```
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` 变量可以用来配置使用基于 Xfce 的或应用程序，如 `x11-toolkits/libxfce4gui` 和 `x11-wm/xfce4-panel` 的 port 的依赖系。

目前，系能下列 Xfce 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，能使用下列参数：

- configenv：如果的 port 需要使用特殊的 `CONFIGURE_ENV` 来所需的。

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

会加到 `CONFIGURE_ENV` 的 `CPPFLAGS`。

因此，如果 port 有到 `sysutils/xfce4-mcs-manager` 的依赖系，并需要在 `configure` 的境中指定特殊的 `CPPFLAGS`，所用的法：

```
USE_XFCE=          mcsmanager configenv
```

6.21. 使用 Mozilla

表 45. 用到 Mozilla 的 port 使用的量

<code>USE_GECKO</code>	port 支持的 Gecko 后端。可： <code>libxul</code> (<code>libxul.so</code>)、 <code>seamonkey</code> (<code>libgtkembedmoz.so</code> ，，新 port 避免使用)。
<code>USE_FIREFOX</code>	port 需要使用 Firefox 作行境依。可： <code>yes</code> (使用默认版本)、 <code>40</code> 、 <code>36</code> 、 <code>35</code> 。默的依采用的是版本 <code>40</code> 。

USE_FIREFOX_BUILD	port 需要使用 Firefox 作依赖。可参：参USE_FIREFOX。该量会自置 USE_FIREFOX 使用相同的。
USE_SEAMONKEY	port 需要使用 SeaMonkey 作依赖。可参：yes (使用默认版本)、20、11 (，新 port 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 作依赖。可参：参USE_SEAMONKEY。该量会自置 USE_SEAMONKEY 使用相同的。
USE_THUNDERBIRD	port 需要使用 Thunderbird 作依赖。可参：yes (使用默认版本)、31、30 (，新 port 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 作依赖。可参：参USE_THUNDERBIRD。该量会自置 USE_THUNDERBIRD 使用相同的。

可用量的完整列表，参 /usr/ports/Mk/bsd.gecko.mk。

6.22. 使用数据

表 46. ports 中有数据的数据量

Variable	Means
USE_BDB	如果该量 yes，把 databases/db41 列依赖。该量可以被置成的有：40, 41, 42、43、44、46、47、48 或 51。可以声明可接受的，USE_BDB=42+ 将已安装的最高版本，如果没有到退回到 42。
USE_MYSQL	如果该量 yes，把 databases/mysql55-server 列依赖。有一个相关的量，WANT_MYSQL_VER，可以置的有 323, 40, 41, 50, 51, 52, 55 或者 60。
USE_PGSQL	如果置成 yes，把 databases/postgresql82 列依赖。有一个相关的量，WANT_PGSQL_VER，可以置的有 73, 74, 80, 81, 82, 83 或 90。

更多情参 [bsd.database.mk](#)。

6.23. 和停止服务 (rc 脚本)

rc.d 脚本在系用于服务，并管理提供停止、和重新某个服务的准方法。Ports 安装的脚本会集成到系的 rc.d 框架中。于如何使用它的明，可以在 [使用手册的 rc.d 章](#) 到。于可用命令的解，可以在 [rc\(8\)](#) 和 [rc.subr\(8\)](#) 到。最后，可以参 [篇文章](#) 了解撰写 rc.d 脚本的最佳践。

可以安装一或多个 rc.d 脚本：

```
USE_RC_SUBR=    doormand
```

有些脚本必须放到 files 目录，并附加 .in。每个文件中可以使用标准的 SUB_LIST 替换。除此之外，我们强烈推荐使用 %%PREFIX%% 和 %%LOCALBASE%% 替换。关于 SUB_LIST 的介绍可以在本节的[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 需要提供安装基本系统的脚本，或者服务需要在 rc.d 脚本 FILESYSTEMS 之前运行特殊情况，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）会加入基本系统的 rcorder(8)。

以下是一个典型的 rc.d 脚本：

```
#!/bin/sh

# $FreeBSD$
#
# PROVIDE: doormand
# REQUIRE: LOGIN
# KEYWORD: shutdown
#
# 在 /etc/rc.conf.local 或 /etc/rc.conf 中添加下述配置可以启用一服务：
#
# doormand_enable (bool): 默认为 NO。
#                        如果 YES 可以启用 doormand。
# doormand_config (path): 默认为 %%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禁用，所有的 ports 脚本都使用

REQUIRE: LOGIN

。如果服务需要以特定用户（除 root 之外）身份运行，则必须这样做。在前面的例子中，我使用了

KEYWORD: shutdown

以便 mythical port 在系统停机的过程中以正常的方式停止，因为它需要在系统引导过程中服务。如果脚本没有做任何服务，则并不需要这样做。

这里，变量的默认方法采用 "=", 而非 "!=" 的形式。这是因为，前一方法只有在变量未被置位才置默认，而后一方法会在变量没有置位，或者其值都为空时都置默认。用非常可能在其 rc.conf.local 中使用类似

```
doormand_flags=""
```

的置位，而采用 ":" 来运行，它会在不意覆盖用户所希望的置位。



新的脚本均不使用 .sh 后缀。未来，仍然包含一后脚本将被批量改名。

6.23.1. 卸载并停止服务

可以在卸载的过程中自在地停止服务。我建议只有在必要时，例如必须在删除文件之前停止服务的情况下才使用这一功能。通常来，决定是否在卸载并停止服务是系统管理需要考虑的事情。此外要注意，这个功能也会影响升级。

需要可以在 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. 添加用户和组

一些 port 需要在安装的系统构建特定的用户或组。如果有这种情况，则从 50 到 999 之间选择一个尚未使用的 UID，并在 ports/UIDs (用户) 或 ports/GIDs (组) 中予以分配。必须确保没有使用系统中已在其他 ports 中使用的 UID。

如果 port 需要构建新用户或组，则在提交补丁的时候一并提交两个文件的补丁。

接下来，可以在 Makefile 中使用 USERS 和 GROUPS 两个变量，系统会在安装时自动构建用户或组。

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

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。下面是使用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 pkg-plist 用法

7.1. 根据 make 量 pkg-plist 行修改

某些 port，特别是 p5- port，会需要根据配置（或于 p5- port 而言，perl 的版本）来修改它的 pkg-plist。简化工作，在 pkg-plist 中的 `%%OSREL%%`、`%%PERL_VER%%`，以及 `%%PERL_VERSION%%` 将自行相的替。其中，`%%OSREL%%` 的是操作系统以数表示的版本（例如 4.9）。`%%PERL_VERSION%%` 和 `%%PERL_VER%%` 是 perl 的完整版本号（例如 5.8.9）。多其它与 port 文文件有的 `%%量%%` 在 相章 中行了介。

如果需行其它的替，可以通将 `PLIST_SUB` 量置一 `量=` 来。其中，`%%VAR%%` 表示在 pkg-plist 中将被 替的那些文字。

例来，如果 port 需要把很多文件放到和版本有的目中，可以在 Makefile 中按照似下面的例子：

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

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

如果的 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 中列出些文件，在行的加上 `%%TAG%%`，并将 TAG 写到 Makefile 中的 `PLIST_SUB` 量中，根据需要替掉，或替 `@comment`，后者表示打包工具忽略行：

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

与之，在 pkg-plist 中：

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

一替程（以及加入 机手册 的程），会在 `pre-install` 和 `do-install` 个 target 之，通取 `PLIST` 并写入 `TMPPLIST`（默情况下，是： `WRKDIR/.PLIST.mktmp`）来完成。因此，如果的 port 生成 `PLIST`，就需要在 `pre-install` 之前完成。外，如果的 port 需要所生成的文件，需要在 `post-install` 中操作名 `TMPPLIST` 的那个文件。

一可行的修改装箱的方法，是根据 `PLIST_FILES` 和 `PLIST_DIRS` 个量的置来行。它的会作目名同 `PLIST` 的内容一起写入 `TMPPLIST`。在 `PLIST_FILES` 和 `PLIST_DIRS` 中列出的名字，会前面所介的 `%%量%%` 替程。除此之外，在 `PLIST_FILES` 中列出的文件，会不加任何修改第出在最的装箱中，而 `@dirrm` 将作前加到 `PLIST_DIRS` 所列的名字之前。了到目的，`PLIST_FILES` 和 `PLIST_DIRS` 必在写 `TMPPLIST` 之前，也就是在 `pre-install` 或更早的段行置。

7.2. 空目录

7.2.1. 清理空目录

一定要在 port 在卸除前清理空目录。通常，可以通过所有由 port 建立的目录添加的 `@dirrm` 行来。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 `@dirrm` 会由于其它 port 使用了同一个目录而产生。利用 `@dirrmtry` 可以只删除那些空目录，而避免出警告。

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

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

7.2.2. 如何建立空目录

在 port 安装过程中建立的空目录需要特别注意。安装 package 并不会自动建一些目录，是因为 package 只保存文件。要安装 package 会自动建一些空目录，需要在 pkg-plist 中加入与 `@dirrm` 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`，不要盲目地安装它，并将其列入 `pkg-plist`，因为会导致 `pkg_delete(1)` 删除精心设计的文件，而新安装又会把那些文件覆盖。

因此，把配置文件的例子按其它的后来安装（例如 `filename.sample` 就是一个不错的例子）并显示一条消息告诉如何复制并安装个配置文件，以便组件能正常工作。

因此，按其它的后来安装配置文件的例子（`filename.sample` 就是一个不错的例子）。如果配置文件不存在，将其复制成文件的名字。卸除时，如果没用修改配置文件，将其删除。需要在 port 的 Makefile，以及 `pkg-plist`（由于从 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，显示一条 [消息](#) 指出用它在何处制并多少个文件，以便组件能开始正常工作。

7.4. 装箱与静装箱的对比

静装箱是指在 Ports Collection 中以 pkg-plist 文件 (可能包含变量替)，或以 `PLIST_FILES` 和 `PLIST_DIRS` 的形式嵌入到 Makefile 输出的装箱。即使它是由工具或 Makefile 中的某个 target 在由 committer 加入到 Ports Collection 之前自生成的也是如此，因为它可以在不下或源代包的前提下其行。

装箱是指在 port 并安装生成的装箱。在下并所移植的程序的源代之前，或在行了 `make clean` 之后，就无法看其内容了。

尽管使用装箱并不被禁止，但人尽可能使用静装箱，因为它能用使用 [grep\(1\)](#) 来所需的 ports，例如，它是否会安装某个特定文件。列表主要用于的，其装箱随所功能会生巨 (因而使得静装箱不再可行)，或那些随版本而改装箱内容的 port (例如，使用 Javadoc 来生成文的那些 ports)。

我鼓励那些使用装箱的人提供一个能生成 pkg-plist 的 target，以便于用其内容。

7.5. 装箱 (package list) 的自动化制作

首先，已基本上完成了 port 的工作，缺 pkg-plist。

接下来，建立一个用于安装 port 的目录，并在其中安装它所依的所有其他件包：

```
# mkdir /var/tmp/`make -V PORTNAME`
# mtrees -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 `port` 遵循 **PREFIX** (如此) 接下来安装 `port` 并建装箱。

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

此外把新建立的目录加入装箱。

```
# (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；它不是完全自的。手册列入 `port` 的 Makefile 中的 **MAN**，而不是装箱。用配置文件被除，或以 `filename.sample` 的名字来安装。 `info/dir` 文件，也不列入，同时按照 **info 文件** 的明来加一些 `install-info` 行。所有由 `port` 安装的，按照 小中介的方法理。

外，也可以使用 `/usr/ports/Tools/scripts/` 中的 **plist** 脚本来自建 package list。 `plist` 脚本是一个 Ruby 脚本，它能将前面介的手工操作自化。

始的和上面的前三行一，也就是 **mkdir**，**mtree** 并 **make depends**。然后和安装 `port`：

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

然后 **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
```

与前面似，如此生成的装箱也需要手工行一些清理工作。

一个可以用来建最初的 `pkg-plist` 的工具是 **ports-mgmt/genplist**。和其他自化工具似，它生成的 `pkg-plist` 手工并根据需要行修改。

Chapter 8. pkg-* 文件

前面有一些没有提及的关于 pkg-* 文件的技巧，它可以方便地完成多任务。

8.1. pkg-message (安装包的显示的消息文件)

如果需要在安装包显示一条消息用，可以把消息放在 pkg-message 中。一特性通常可以用于在 `pkg_add(1)` 之后显示一些附加的安装，或显示于授的信息。

当需要显示一些或警告，使用 `ECHO_MSG`。pkg-message 文件只是显示安装后的行操作指使用的。似地，需要留意 `ECHO_MSG` 和 `ECHO_CMD` 之的区别。前一个是在屏幕上显示消息性的文字，而后者用于在行命令使用管道。

下面是用到了个宏的例子 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，在用使用 port 而不是 package 来安装包，它并不会被显示出来。因此如果需要的，在 `post-install` target 中指定显示它。

8.2. pkg-install (安装包的运行的脚本文件)

如果的 port 需要在包的安装包通 `pkg_add(1)` 安装行一些命令，通 `pkg-install` 脚本来完成。个脚本会自地加入 package，并被 `pkg_add(1)` 行次：第一次是 `${SH} pkg-install ${PKGNAME} PRE-INSTALL` 而第二次是 `${SH} pkg-install ${PKGNAME} POST-INSTALL`。\$2 可被用来脚本行的模式。境量 `PKG_PREFIX` 将置 package 的安装目。参 `pkg_add(1)` 以了解更一的。



在使用 `make install` 个脚本不会被自行。如果需要行它，必在的 port 中的 Makefile 里明确地予以用，其方法是加入似 `PKG_PREFIX=${PREFIX} ${SH} ${PKGINSTALL} ${PKGNAME} PRE-INSTALL` 的命令。

8.3. pkg-deinstall (卸包的运行的脚本文件)

一脚本将在 package 被卸行。

此脚本会被 `pkg_delete(1)` 行次。第一次是 `${SH} pkg-deinstall ${PKGNAME} DEINSTALL` 而第二次是 `${SH} pkg-deinstall ${PKGNAME} POST-DEINSTALL`。

8.4. pkg-req (安装包是否行操作的脚本文件)

如果的 port 需要定它是否被安装，可以建 pkg-req"requirements" 脚本。它会在安装/卸自自行，以决定操作是否被施。

个脚本会在使用 `pkg_add(1)` 安装以 `pkg-req ${PKGNAME} INSTALL` 的命令行行。卸，它将由 `pkg_delete(1)` 以 `pkg-req ${PKGNAME} DEINSTALL` 的命令行行。

8.5. 改 pkg-* 文件的名字

所有 pkg- 文件的名字，皆系采用量予以定，因此在需要可以在的 Makefile 中加以改。当需要在多个 port 之共享某些 pkg- 文件，或需要写入某些文件就非常有用。 (参在 `WRKDIR` 以外的地方写文件，以了解什直接将更写入 pkg-* 子目是个糟糕的主意)

下面是一量以及它的默认 (PKGDIR 默认情况下是 `${MASTERDIR}`。)

量	默认
DESCR	<code>\${PKGDIR}/pkg-descr</code>
PLIST	<code>\${PKGDIR}/pkg-plist</code>
PKGINSTALL	<code>\${PKGDIR}/pkg-install</code>
PKGDEINSTALL	<code>\${PKGDIR}/pkg-deinstall</code>
PKGREQ	<code>\${PKGDIR}/pkg-req</code>
PKGMESSAGE	<code>\${PKGDIR}/pkg-message</code>

修改些量，而不是直接覆 `PKG_ARGS` 的。如果改了 `PKG_ARGS`，些文件将无法在安装 port 正地制到 `/var/db/pkg` 目。

8.6. 使用 SUB_FILES 和 SUB_LIST

`SUB_FILES` 和 `SUB_LIST` 个量可以用来在 port 文件中使用某些的，例如 `pkg-message` 中的 `installation PREFIX`。

用 `SUB_FILES` 量，可以指定需要自行修改的文件列表。在 `SUB_FILES` 中的一个文件，在 `FILESDIR` 目中都必须有一个的文件.in。修改后的版本将保存在 `WRKDIR`。在 `USE_RC_SUBR` (或已的 `USE_RCORDER`) 中定的文件会自加入到 `SUB_FILES` 中。于 `pkg-message`、`pkg-install`、`pkg-deinstall` and `pkg-req`，的 Makefile 量会被自置，以指向理的版本。

`SUB_LIST` 个量的内容是一系列 `VAR=VALUE` 。 `SUB_FILES` 所列出的文件中所有的 `%VAR%` 都将被替 `VALUE`。系自定了一些常用的替，包括：`PREFIX`、`LOCALBASE`、`DATADIR`、`DOCSDIR`，以及 `EXAMPLESDIR`。替果中所有以 `@comment` 的行，都将在量替之后被去。

下面的例子中，将把 `pkg-message` 中的 `%ARCH%` 替系所行的架名称：

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

注意，在上述例子中，**FILESDIR** 里必须有 pkg-message.in 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. 的 port

9.1. 行 make describe

多 FreeBSD port 工具，例如 [portupgrade\(1\)](#)，会依赖于一个名为 `/usr/ports/INDEX` 的数据的正性，它提供了关于 port 的相关信息，例如依赖系等等。INDEX 是由的 `ports/Makefile` 通过 `make index` 来建立的，这个命令会入一个 port 的子目，并在那里行 `make describe`。因此，如果某个 port 的 `make describe` 失，就没有人能生成 INDEX，人很快会得不高。



无在 `make.conf` 中置了什，个文件都能正地生成。因此，避免在（例如）某个依赖系无法足使用 `.error`。（参 [避免使用 .error](#)。）

如果 `make describe` 只是生一个字符串，而不是信息，可能就没什。参 `bsd.port.mk` 以了解所生成的串的意思。

最后要明的是，新版本的 `portlint`（在下一中将行介）将会自地行 `make describe`。

9.1.1. Portlint

在提交或 commit 之前，使用 `portlint` 来行。 `portlint` 会常的、包括功能上的和格式上的出警告。于新的（或在 repocopy 代中制的）port，`portlint -A` 可以完成全面；于存的 port，`portlint -C` 一般就足了。

由于 `portlint` 采用式方法来，有它会生警。外，有由于 port 框架的限制可能没有办法修正它指出的。如果有疑，写信 [FreeBSD ports 件列表](#)。

9.1.2. 使用 Port Tools 来完成

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

`port` 是一个能助化工具的前端脚本。如果希望新的 port 或更新 port 行，可以用 `port test` 来完成些工作，也包含了 `portlint`。这个命令会并列出没有在 `pkg-plist` 中列出的文件。具体用法参下面的例子：

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

9.1.3. PREFIX (安装的目的目名) 和 DESTDIR

`PREFIX` 能决定 port 安装的目的位置。一般情况下个位置是 `/usr/local` 或 `/opt`，但也可以其它的任意。的 port 必遵循个量。

除此之外，如果用配置了 `DESTDIR`，表示希望将 port 安装到一个境，通常是 `jail` 或在 / 以外的其他位置挂接的系中。上，port 会安装到 `DESTDIR/PREFIX`，并注册到位于 `DESTDIR/var/db/pkg` 的包数据中。由于 `DESTDIR` 是由 ports 框架藉由 [chroot\(8\)](#) 来的，在撰写符合 `DESTDIR` 的 ports 并不需要什么外的工作。

一般而言 `PREFIX` 会 `LOCALBASE_REL`（默是 `/usr/local`）。如果置了 `USE_LINUX_PREFIX`，`PREFIX` 会

`LINUXBASE_REL` (默认是 `/compat/linux`)。

避免将 `/usr/local` 或 `/usr/X11R6` 硬编码到源代码中，能大大提高 port 的灵活性，并满足不同环境的需要。对于使用 `imake` 的 X port，这一工作是由它完成的；其他情况下，通常可以明智地将 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`) 替换为取 `${PREFIX}` 变量就能达到目的了，因为这个变量在编译和安装的过程中，会自行向下调整。

一定要避免将 port 在 `/usr/local` 而不是正好的 `PREFIX` 中安装文件。正确的方法是：

```
# 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，打包程序将抱怨找不到这些文件。

它并不能帮助内部引用，或正在引用其它 port 中的文件所使用的 `LOCALBASE`。它需要在 `/var/tmp/make -V PORTNAME` 中安装好的文件，才能达到目的。

它可以在自己的 Makefile 中改变 `PREFIX` 变量的值，也可以通过用环境变量来影响它。然而，一般情况下决不可以在 Makefile 中明确设置它的值。

此外，引用其它 port 中的文件时，使用前面介绍的变量，而不要直接指定它的路径名。例如，如果该 port 需要使用 `PAGER` 这个宏来指明 `less` 的完整路径，使用下面的代码：

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

而非 `-DPAGER="/usr/local/bin/less"`。这种方法能增加在系统管理时把整个 `/usr/local` 目录移到其它位置安装成功的机会。

9.1.4. Tinderbox

如果你是非常核心的 ports 参与者，你可以看看 Tinderbox。它是一个大的用于管理和维护 ports 的系统，它基于 [Pointyhat](#) 的脚本。你可以使用 `ports-mgmt/tinderbox` port 来安装 Tinderbox。你一定仔细读随它安装的文档，因为配置并不简单。

看看 [Tinderbox 网站](#) 以了解它的一切。

Chapter 10. 升一个 port

如果某个 port 和原作者所布的版本已不同，首先需要的是该 port 是最新的。可以在 FreeBSD FTP 镜像的 `ports/ports-current` 目录中找到它。但是，如果正在使用很多的 port，可能使用 CVSup 来保持 Ports Collection 最新更新，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。

下一步是是否已有在等待的更新。要完成这项工作，可以采用下列方法之一。有一个用于搜索 [FreeBSD 公告 \(PR\) 数据库](#) (也被称作 GNATS)。在下拉框中输入 `ports`，然后输入 port 的名字。

但是，有些人可能会将避免混淆的 port 的名字放到 Synopsis 字段中。这时候，可以看看 [FreeBSD Ports 系统](#) (也被叫做 `portsmon`)。该系统会按照 port 的名字来执行分析。要搜索和某个特定 port 有关的 PR，可以使用 [port 概览](#)。

如果没有相关的 PR，下一步是该 port 的维护者写信，可以通过运行 `make maintainer` 看到。个人可能正在执行升级工作，或者由于某些理由没有升级（例如，新版本有稳定性问题）；一般不希望重他的工作。注意没有维护者的 port 的维护者会显示 `ports@FreeBSD.org`，是一般性 port 的邮件列表，因此邮件它一般没什么意义。

如果维护者要求去完成升级，或者没有维护者，就有机会通过自行完成升级来帮助 FreeBSD 了！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，将需要修改的文件制成 `something.orig`，并将改动放入 `something`，接着生成补丁：

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

如果不是这样的话，可以使用 `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)` 比较两个目录在目录中生成的输出结果（例如，如果修改后的 port 目录的名字是 `superedit` 而原始文件的目录是 `superedit.bak`，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 port committer 会更喜欢一致式 diff。注意这里使用的 `-N`，它的目的是强制 diff 正确地处理出新文件，或老文件被删除的情形。在把 diff 发我之前，再次输出，以便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，可以使用 `/usr/ports/Tools/scripts/patchtool.py`。使用之前，首先看 `/usr/ports/Tools/scripts/README.patchtool`。

如果 port 目前无人维护，而且自己经常使用它，考虑自任它的维护者。FreeBSD 有超过 4000 个没有维护者的 port，而这正是最需要志愿者的领域。（要了解关于志愿者的任何描述，参看 [手册中的相关部分](#)。）

将 diff 发送给我的最佳方式是通过 `send-pr(1)` (category 一写 `ports`)。如果正是那个 port，必须在 synopsis 的末尾写上 `[maintainer update]`，并将该 PR 的 "Class" 置为 `maintainer-update`。反之，该 PR 的 "Class" 就应该是 `change-request`。在信中逐个提及一个删除或添加的文件，因为有些都必须明确地在使用 `cvs(1)` 执行 commit 时明确地指定。如果 diff 超过了 20K，考虑将其进行 uuencode；否则，直接地将其原样加入 PR 即可。

在 `send-pr(1)` 之前，再次看 Problem Reports 一文中的 [如何撰写公告](#) 小节；它出了丰富的

关于如何撰写更好的公告的介绍。



如果包的更新是由于安全考虑，或修复已 commit 的 port 中的漏洞，通知 Ports 管理 <portmgr@FreeBSD.org> 来申请立即重建和分发的 port 的 package。否则，不建议的使用 `pkg_add(1)` 的用法，可能会在未来数周之内通过使用 `pkg_add -r` 安装旧版本。



再次提醒，使用 `diff(1)` 而非 `shar(1)` 来发送有 port 的更新！可以帮助 ports committer 理解需要修改的内容。

在您已经了解了所需的所有操作，可能会像要在 [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，提交 `cvsv(1)` diff；该情形要比直接比较“新、旧”目录要容易处理。此外，该方法也更容易看出到底改了什么，并在其他人更新了 Ports Collection 容易合并一些改动，在提交之前，可以减少补丁所需的工作。

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

- ① 当然，可以是任意目录；该 port 并不局限于 /usr/ports/ 的子目录。
- ② R_CVSRROOT 是任何一个公共的 cvs 服务器，可以在 [FreeBSD 使用手册](#) 中挑一个。
- ③ pdnsd 是 port 的模块名字；通常来它和 port 的名字一样，不过也有些例外，特别是那些本地化（[german/selfhtml](#) 的模块名字是 `de-selfhtml`）；可以通过 [cvsweb 界面](#) 查看，或者也可以指定完整路径，例如在我这个例子中是 `ports/dns/pdnsd`。

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

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

反过来，`port` 列出的事项并使用 [用 portlint 来检查 port](#) 进行。

```
% cvs status
% cvs update ①
```

- ① 会合并 CVS 中其他人做的改动和你的补丁；在这个过程中，需要仔细观察输出。文件名前面的那个字母会指示做了什么，参看 [cvs update 文件名前字母前缀的含义](#) 中输出的说明。

表 47. `cvs update` 文件名前字母前缀的含义

U	文件更新无冲突。
P	文件更新无冲突（通常只有在使用程序代换时才会看到）。

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。

如果在运行 `cv update` 某些文件出了 **C**， 说明有其他人在 CVS 中做了修改， 而 `cv(1)` 无法将这些改与本地的改进行合并。 不过， 无论如何， 最好都看一下合并的结果， 因为 `cv` 并不知道 port 是什么东西， 因此它所做的合并无论是否产生了冲突， 都有可能 (并且并不可能) 产生没有意义的结果。

最后一项是以 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， 因为它保证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， 这些文件会被清空， 所以最好在 PR 中提醒 committer 删除它。

根据 [升一个 port](#) 的指引提交你的补丁。

10.2. UPDATING 和 MOVED 文件

如果在升 port 可能需要修改配置文件或运行特定的程序的特性， 请在 `/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/port名字
AUTHOR: 你的名字 <你的电子邮件地址>

所需行的特性
```

如果你需要在内文中加入具体的 `portmaster` 或 `portupgrade` 的说明， 确保所用的 `shell` 命令使用了正确的符号。

如果 port 被删除或改名， 请在 `/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，安全漏洞是最危险的。从技术角度看，有些漏洞可以通过消除导致它的 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 通常只影响那些用了某些能触到它的组合的用户。用户最会在发布没有那个的新版之后出一个补丁来修正它，而用户中的主体并不会立即升级，因为他并没有因存在而感到痛苦。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 通常会被很快修复。

安全漏洞完全不同。第一，它可能存在数年而不被发现，因为它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授权的权限，并利用这些权限掉或修改敏感数据；而更糟糕的情况是用户可能根本注意不到损害已产生。第三，暴露出安全漏洞的系统，往往能协助攻击者入侵其它之前不可能入侵的系统。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，让他就能评估，并采取适当的措施。

11.2. 修复安全漏洞

当引起 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。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，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迟，要将 port 标记为 FORBIDDEN，要在 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，都是需要按照 [批准的提交流程](#) 提交，除非有直接在 ports tree 上 commit 的权限。



作为 ports committer 并不能随便 commit 所有 port。注意通常 port 都有所有者，而他应得到尊重。

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 的用户就能看到他们需要升级。另外，通过 FTP 和 WWW 像发布，以取代有漏洞的版本。注意要增加 PORTREVISION 数字，除非在修正时 PORTVERSION 发生了变化。一般来说，如果在 port 中增加了补丁文件，就增加 PORTREVISION，但例外的例子是已经将软件升级到了最新版，因此已经改掉 PORTVERSION 了。参考 [相关小](#) 以了解更多信息。

11.3. 通知整个用户群体

11.3.1. VuXML 数据

当有了安全漏洞的一个重要而迫切的问题，就是使用 port 的用户群了解其危险。通知有重要目的。首先，如果危害真的很严重，可能理性的办法就是立即用一些缓解措施，例如，停止受到影响的服务器，甚至完全删除 port，直到被修正为止。其次，多用只是偶尔升级所安装的软件包，通过通知，他能知道已到了必须更新软件的时候，因为他已经有了修正某些版本的版本了。

由于有的 port 数量其大，没有一个都能发布安全公告，毫无疑问地会发表和狼来了一堆多的安全公告，并大受在真的产生严重的忽略的可能。因此，在 port 中的安全漏洞，会在 [FreeBSD VuXML 数据](#)

中。安全官可能会持续地追踪个数据的修改，以了解需要他注意的内容。

如果你是 committer，你可以自行更新 VuXML 数据。，你就能同助安全官，并尽早将至重要的信息用群体。然而，如果你不是 committer，或者你相信自己了一个常重的漏洞，不要犹豫，按照 [FreeBSD 安全信息](#) 面上的方法系安全官。

正如其名称所暗示的那，VuXML 数据是一个 XML 文。其源文件 vuln.xml 被保存在 [security/vuxml](#) port 的目中。所以，它的全名是 PORTSDIR/security/vuxml/vuln.xml。当 port 中的安全漏洞，把新的加入到那个文件中。在熟悉 VuXML 之前，最好先看看是否有似的其它的，并制它作模板。

11.3.2. VuXML 介

XML 是一个的言，它超越了本的。不，只需了解的命名，就能 VuXML 的有一个大体的了解了。XML 的名字出在一尖括号之。一个 <tag> 必有一个的 </tag>。可以嵌套，如果嵌套的，内的必在外之前束。就形成了一个的次，也就是于它之如何嵌套的。听起来很像 HTML，是不是？最主要的区在于，XML 是可展的 (eXtensible)，例如通定新的等等。由于其的内在性，XML 能予无的数据新的形。VuXML 是描述安全漏洞的言。

在我来察一个的 VuXML ：

```

<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&m=20388660782
        5605</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>

```

包的名字都是显而易见的，下面我来介绍一下需要由包填写的字段：

- ① 包是 VuXML 包的 XML tag。它有一个控制性的字段，`vid`，用于此包（它包含的部分）指定一个全局唯一标识符 (UUID)。包生成一个新的 vuXML 生成新的 UUID（而且别忘了要把模板中的 UUID 改成新的，如果不是从包始的）。包可以使用 `uuidgen(1)` 来生成 VuXML UUID。
- ② 包于包的一句描述。
- ③ 此包出受到影响的 package 的名字。可以列出多个名字，因为可能有多件包基于同一个 master port 或件品。包可能包括固定和分支、本地化版本，以及提供了不同的包的 slave port。
- ④ 受影响的 package 版本，可以使用 `<lt>`, `<le>`, `<eq>`, `<ge>`, 和 `<gt>` 表成一个或多个版本及其。注意包出的版本不能存在重。在描述包的时候，`*`（星号）表最小的版本。更具体地，`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>` 可以性地匹配一个 `2.x` 的版本，而 `<ge>2.0</ge><lt>3.0</lt>` 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 受到影响的一个 package（本上是 ports）可以列在 `<affected>` 小中。如果多个件品都采用了同的基代，（比如 `FooBar`、`FreeBar` 和 `OpenBar`）而且包含同的 bug 或漏洞。注意列出多个名字，包在一个 `<package>` 小中完成。
- ⑥ 如果可能，版本的包包括 `PORTEPOCH` 和 `PORTREVISION`。包必注意，根据加，包有非零 `PORTEPOCH` 的版本，系会包比没有 `PORTEPOCH` 的版本高，例如 `3.0,1` 高于 `3.1` 甚至 `8.9`。
- ⑦ 包于包的摘要性信息。此包使用 XHTML。包必要成包使用 `<p>` 和 `</p>`。可以使用包的包，但包限于有助于包信息更准和明了的修：包不要分地美化。
- ⑧ 包部分包含了相的可供参考的文。包尽可能多提供参考文献。
- ⑨ 指定 [FreeBSD 安全公告](#)。
- ⑩ 指定 [FreeBSD 公告](#)。
- 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 漏洞明](#)。
- ⑮ 指定 [US-CERT 计算机安全警](#)。
- ⑯ 指定 [US-CERT 技术性计算机安全警](#)。
- ⑰ 指向件列表存的 URL。属性 `msgid` 是可，用以指定某一封信的 message ID。
- ⑱ 一般的 URL。只有在没有其它更合的参考文献，才使用它。
- ⑲ 包被全面披露的日期 (YYYY-MM-DD)。
- ⑳ 包加入到数据中的日期 (YYYY-MM-DD)。

包最后一次被修改的日期 (YYYY-MM-DD)。新包不能包括个字段。只有在修改时才加入它。

11.3.3. 包 VuXML 数据包所作的修改

假定打算撰写，或已写好了一个于 package `clamav` 的描述，并且，已知 `0.65_7` 版本修正了个。

需要做的工作，是安装一个新版本的 `ports-ports-mgmt/portaudit` 程序、`ports-ports-mgmt/portaudit-db`，以及 `security/vuxml`。



要运行 `packaudit`，必须有其 `DATABASEDIR`，通常是 `/var/db/portaudit` 的写入权限。

可以通过 `DATABASEDIR` 环境变量来指定一个不同的位置。

如果的工作目录是 `${PORTSDIR}/security/vuxml` 以外的其它地方，使用环境变量 `VUXMLDIR` 来指明 `vuln.xml` 的位置。

首先，看一下是否已经有了关于漏洞的描述。如果已经有了的，那它将匹配早版本的 `package`, `0.65_6`：

```
% packaudit
% portaudit clamav-0.65_6
```

如果什么都没有，就可以考虑写一个新的来描述漏洞了。在可以生成一个新的 UUID（假设它是 `74a9541d-5d6c-11d8-80e3-0020ed76ef5a`），然后将新的加入到 `VuXML` 数据中。接下来，用下面的命令来验证它是否符合：

```
% cd ${PORTSDIR}/security/vuxml && make validate
```



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 `portaudit` 数据：

```
% packaudit
```

要重新加入的 `<affected>` 小段能正确地匹配希望的 `package`，可以使用下面的命令：

```
% portaudit -f /usr/ports/INDEX -r 74a9541d-5d6c-11d8-80e3-0020ed76ef5a
```



参 `portaudit(1)` 以了解关于命令的更多。

相信新添加的不会在输出中匹配不匹配的。

在添加的所匹配的版本是否正：

```
% 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.
```

当然，前一个版本会匹配，而后一个不会。

最后，从 VuXML 数据中能正确地得到期的网页效果：

```
% mkdir -p ~/public_html/portaudit
% packaudit
% lynx ~/public_html/portaudit/74a9541d-5d6c-11d8-80e3-0020ed76ef5a.html
```

Chapter 12. 做什么和不做什么

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。按照这个列表自己的 port，同时，也可以借助 PR 数据中由其他人提交的 port。按照在 [公告和一般性注](#) 中介绍的方法提交我的看法。借助 PR 数据中的 ports 即能助我更快地 commit 它，也能明确清楚地了解如何完成些工作。

12.2. WRKDIR (要使用的目录)

任何时候都不要 **WRKDIR** 以外的位置写文件。**WRKDIR** 是在 port 目录中唯一的一个一定可写的地方（参看[如何从 CDROM 安装 port](#) 以了解从只读的目录中和安装 port 的例子）。如果需要改 pkg-* 文件，按照[重新定义某个变量](#) 介绍的方法，而不是覆写它来。

12.3. WRKDIRPREFIX (用于目录的目录的父目录名)

一定要保证的 port 尊重 **WRKDIRPREFIX** 的设置。大多数 port 并不需要担心这个。具体来，当引用其它 port 的 **WRKDIR** 时，需要注意正的位置是 **WRKDIRPREFIX**PORTSDIR/subdir/name/work 而不是 PORTSDIR/subdir/name/work 或 .CURDIR/../../subdir/name/work，或别的什么。

另外，如果自行定义了 **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 下可能需要代换行一些修改或加少代码，才能正确地编译和运行。如果需要根据一些条件来代换行修改，尽可能做些修改通用，那么，我就能将些代码移植回更早的 FreeBSD 系，并交叉移植到其它 BSD 系，例如来自 CSRG 的 4.4BSD，BSD/386，386BSD，NetBSD 和 OpenBSD。

推荐的得 4.3BSD/Reno (1990) 以及更新版本 BSD 代换版本号的方式，是使用 [sys/param.h](#) 中所定义的 BSD 宏的。一般来这个文件已被引用了；如果没有的，加下述代码：

```
#if (defined(__unix__) || defined(unix)) && !defined(USG)
#include <sys/param.h>
#endif
```

到 .c 文件中合的地方。我相信所有定义了这些符号的系中，都提供了 sys/param.h。如果确实没有做的系，通致信 [FreeBSD ports 邮件列表](#) 我了解一情况。

另一方法是使用 GNU Autoconf 风格的方式：

```
#ifdef HAVE_SYS_PARAM_H
#include <sys/param.h>
#endif
```


采用这种方法，不要忘了把 `-DHAVE_SYS_PARAM_H` 加到 Makefile 中的 `CFLAGS` 里。

一旦引用了 `sys/param.h`，就可以使用：

```
#if (defined(BSD) && (BSD >= 199103))
```

来检测是否正在 4.3 Net2 代基，或更新的系上（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))
```

来检测是否正在 4.4 或更新的系（例如 FreeBSD 2.x，4.4，NetBSD 1.0、BSD/386 2.0 或更高版本）。

于 4.4BSD-Lite2 代系来，`BSD` 宏的值是 199506。这里只是作信息提供，并不使用它来区分基于 4.4-Lite 的 FreeBSD 和基于 4.4-Lite2 的版本。些情况下，使用 `{freebsd}` 宏。

保守地使用：

- `{freebsd}` 在所有版本的 FreeBSD 中皆有定义。如果正行的修改只影响 FreeBSD，使用个宏。似 `sys_errlist[]` 之于 `strerror()` 的移植是伯克利代系公用的，而并非 FreeBSD 所特有。
- 在 FreeBSD 2.x 中，`{freebsd}` 定义为 2。更早版本中，它曾是 1。新的版本都会在主要的版本号化更它。
- 如果需要区分 FreeBSD 1.x 系和 FreeBSD 2.x 及更高版本的区，通常使用前述的 `BSD` 宏来行。如果事上需要一个 FreeBSD 有的修改（例如，在使用 `ld` 需要特殊的共享），可以用 `{freebsd}` 和 `#if {freebsd} > 1` 来 FreeBSD 2.x 和新系上的化。如果需要更粒度地 FreeBSD 2.0-RELEASE 之后版本的化，可以使用：

```
#if __FreeBSD__ >= 2
#include <osreldate.h>
#   if __FreeBSD_version >= 199504
/* 用于 2.0.5+ 版本的代 */
#   endif
#endif
```

在已有的数百个 port 中，只有一使用 `{freebsd}`。早期的 port 在不当的地方使用了它并引，并不意味着也必定如此。

12.5. `__FreeBSD_version`

下面是在 `sys/param.h` 中定义的及其意义的列表，里出以方便：

表 48. `__FreeBSD_version`

□	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
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□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

□	日期	版本
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□化之后的 3.0-CURRENT
300004	September 3, 1998	ELF □□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
310002	April 14, 1999	C++ □建/析□函数□序□化之后的 3.1-STABLE
320000		3.2-RELEASE
320001	May 8, 1999	3.2-STABLE
320002	August 29, 1999	二□制不兼容的 IPFW 和 socket □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□□□接器□理方式之后的 4.0-CURRENT
400002	March 13, 1999	C++ □建/析□函数□序□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)

□	日期	版本
400005	April 27, 1999	suser(9) API □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□之后的 4.0-CURRENT
400007	June 17, 1999	加入了 socket □凭据的 so_cred 之后的 4.0-CURRENT
400008	June 20, 1999	在 libc_r 中加入 poll 系□□用接口之后的 4.0-CURRENT
400009	July 20, 1999	将内核中 dev_t □型改□ struct specinfo 指□之后的 4.0-CURRENT
400010	September 25, 1999	修正了一□ jail(2) 漏洞之后的 4.0-CURRENT
400011	September 29, 1999	sigset_t 数据□型改□之后的 4.0-CURRENT
400012	November 15, 1999	切□到 GCC 2.95.2 □□器之后的 4.0-CURRENT
400013	December 4, 1999	加入了可□的 linux 模式 ioctl □理程序后的 4.0-CURRENT
400014	January 18, 2000	引入 OpenSSL 之后的 4.0-CURRENT
400015	January 27, 2000	GCC 2.95.2 中 ABI 默□从 -fvtable -thunks 改□ -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□校□和之后的 4.0-STABLE
400020	June 4, 2000	将 libxpg4 的代□并入 libc 之后的 4.0-STABLE
400021	July 8, 2000	Binutils 升□到 2.10.0 之后的 4.0-STABLE, ELF □志□化, 以及将 tcsh 引入基本系□
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

□	日期	版本
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□的 GCC □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 □源状□ 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 □□改□之后的 4.4-STABLE (影□文件系□ kld)
440003	December 18, 2001	用□□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 □□元素改名之后的 4.5-STABLE
450004	April 16, 2002	在 rc.conf(5) □量 sendmail_enable 默□□改□ NONE 之后的 4.5-STABLE
450005	April 27, 2002	默□将 XFree86 4 用于□□□包 □□之后的 4.5-STABLE
450006	May 1, 2002	accept □□器修正了安全 □□并且不再会□易被 DoS 之后的 4.5-STABLE
460000	June 21, 2002	4.6-RELEASE
460001	June 21, 2002	修正了 sendfile(2) 以吻合文□, 而不再根据□出的□□算□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 □多 pkg_install 新特性之后的 4.6-STABLE

□	日期	版本
470000	October 8, 2002	4.7-RELEASE
470100	October 9, 2002	4.7-STABLE
470101	November 10, 2002	□始生成 <code>std{in,out,err}p</code> 引用, 而不是 <code>sF</code> 。□将 <code>std{in,out,err}</code> 从□□□表□式□成了□行□□。
470102	January 23, 2003	MFC mbuf 相□的将 <code>m_aux mbuf</code> 改□ <code>m_tag</code> 的修改之后的 4.7-STABLE
470103	February 14, 2003	OpenSSL 升□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) □□□程安全的之后的 4.8-STABLE
480102	August 10, 2003	□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	<code>kinfo_eproc</code> 中加入 <code>e_sid</code> 之后的 4.9-STABLE
490102	February 4, 2004	MFC <code>rtld</code> 的 <code>libmap</code> 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 □□工具之后的 4.10-STABLE
491102	November 16, 2004	修正了 VM 当解除 <code>wire</code> 不存在□面□的□□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 <code>libdata/ldconfig</code> 目□加入 <code>mtree</code> 文件之后的 4.11-STABLE。
500000	March 13, 2000	5.0-CURRENT
500001	April 18, 2000	加入 ELF □字段, 并改□我□的 ELF □行文件□□方式之后的 5.0-CURRENT

□	日期	版本
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□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□到 5.6.0 之后的 5.0-CURRENT
500008	July 7, 2000	KAME 代□升□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□原先的默□□, 并使用 -L 来跟随符号□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 □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□ <sys/selinfo.h> 之后的 5.0-CURRENT
500015	January 10, 2001	libgcc.a 和 libgcc_r.a 以及 GCC □接方式□□之后的 5.0-CURRENT
500016	January 24, 2001	修改以允□ libc 和 libc_r □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□到 struct xucred 以便使内核□ mountd 等程序□出的 API □定下来之后的 5.0-CURRENT
500018	February 24, 2001	加入 CPUTYPE 用于 CPU □用的 □化的 make □量之后的 5.0-CURRENT

□	日期	版本
500019	June 9, 2001	machine/ioctl_fd.h 改□ sys/fdcio.h 之后的 5.0-CURRENT
500020	June 15, 2001	locale 名称改□之后的 5.0-CURRENT
500021	June 22, 2001	引入 bzip2 之后的 5.0-CURRENT, 同□也代表□去了 S/Key
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□ UUCP 被移入 ports
500025	October 4, 2001	64-位平台上的描述符和 creds API □化之后的 5.0-CURRENT
500026	October 9, 2001	采用 XFree86 4 作□默□的□□□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□□的 smbfs □件之后的 5.0-CURRENT
(未予□加)		加入了新的 C99 指定位□整形 □量之后的 5.0-CURRENT
500029	January 29, 2002	修改了 sendfile(2) 的返回□之后的 5.0-CURRENT
500030	February 15, 2002	引入□合表□文件□志的 fflags_t □型之后的 5.0-
500031	February 24, 2002	usb □□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) □量 sendmail_enable 默□改□ NONE 之后的 5.0-CURRENT
500034	April 30, 2002	mtx_init() □加了第三个参数之后的 5.0-CURRENT

编号	日期	版本
500035	May 13, 2002	包含 Gcc 3.1 的 5.0-CURRENT
500036	May 17, 2002	在 /usr/src 中移除了 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 的成对修改, 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 <i>BSD_FOO_T</i> 而开始使用 <i>_FOO_T_DECLARED</i> 。这个头文件可以用于作一个包含使用 bzip2(1) 的包支持的日期点。
500040	September 20, 2002	以去掉 disklabel 内容的依赖为名, 磁盘相关的函数进行了多次修改之后的 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 模型, 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> 成了一个空文件, 不再被引用
500102	February 25, 2003	修改 d_mmap_t 接口之后的 5.0-CURRENT
500103	February 26, 2003	taskqueue_swi 以无全局的方式运行之后的 5.0-CURRENT, 同时加入了使用全局的 taskqueue_swi_giant
500104	February 27, 2003	去掉了 cdevsw_add() 和 cdevsw_remove() 输出 MAJOR_AUTO 分配机制

□	日期	版本
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；□参□ sys/sys/param.h 1.149
500108	March 15, 2003	改□了 Token-Ring 接口
500109	March 25, 2003	加入 vm_paddr_t
500110	March 28, 2003	将 realpath(3) 改□程安全之后的 5.0-CURRENT
500111	April 9, 2003	usbhid(3) 与 NetBSD 同□之后的 5.0-CURRENT
500112	April 17, 2003	加入新的 NSS □□，以及 POSIX.1 getpw*_r, getgr*_r 函数之后的 5.0-CURRENT
500113	May 2, 2003	□去旧式 rc 系□之后的 5.0-CURRENT
501000	June 4, 2003	5.1-RELEASE.
501100	June 2, 2003	□建 RELENG_5_1 分支之后的 5.1-CURRENT
501101	June 29, 2003	改正 sigtimedwait(2) 和 sigwaitinfo(2) □□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 □化之后的 5.1-CURRENT
501105	August 17, 2003	允□□□接 /bin 和 /sbin, 以及将某些□移□到 /lib 之后的 5.1-CURRENT
501106	September 8, 2003	□加内核□ Coda 6.x 支持之后的 5.1-CURRENT

ID	日期	版本
501107	September 17, 2003	将 16550 UART 常量从 <dev/sio/sioreg.h> 移到 <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	添加 kiconv(3) 之后的 5.1- CURRENT
501110	September 28, 2003	默认 cdevsw open 和 close 操作国际化之后的 5.1-CURRENT
501111	October 16, 2003	cdevsw 的布局国际化之后的 5.1- CURRENT
501112	October 16, 2003	添加 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 改回链接之后的 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 函数之后的 5.2-CURRENT
502102	January 30, 2004	默认线程从 libc_r 改为 libpthread 之后的 5.2-CURRENT
502103	February 21, 2004	API 大模翻修之后的 5.2- CURRENT
502104	February 25, 2004	添加 getopt_long_only() 之后的 5.2-CURRENT
502105	March 5, 2004	C 的 NULL 定义改为 ((void *)0) 之后的 5.2-CURRENT，会产生更多的警告
502106	March 8, 2004	pf 输入和安装程序之后的 5.2- CURRENT
502107	March 10, 2004	在 sparc64 上将 time_t 改为 64-位 之后的 5.2-CURRENT

□	日期	版本
502108	March 12, 2004	在一些□文件修改以支持 Intel C/C++ □□器， 以及□ execve(2) 更□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	□去 getvfsent(3) API 之后的 5.2-CURRENT
502112	April 13, 2004	□ make(1) □加 .warning □句之后的 5.2-CURRENT
502113	June 4, 2004	所有串口□□都□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□返回 0， 失□□返回非 0 的□□代□之后的 5.2-CURRENT
502116	June 16, 2004	将内核 dev_t 改□指向 struct cdev * 的指□之后的 5.2-CURRENT
502117	June 17, 2004	将内核 udev_t 改□ dev_t 之后的 5.2-CURRENT
502118	June 17, 2004	□ clock_gettime(2) 和 clock_getres(2) □加 CLOCK_VIRTUAL 和 CLOCK_PROF 支持之后的 5.2-CURRENT
502119	June 22, 2004	□网□接口□制□行全面修改之后的 5.2-CURRENT
502120	July 2, 2004	package 工具升□ 20040629 之后的 5.2-CURRENT
502121	July 9, 2004	不再将□牙代□□□□ i386 □用之后的 5.2-CURRENT
502122	July 11, 2004	引入 KDB □□器框架之后的 5.2-CURRENT。 同□□引入了 DDB 作□后台， 以及 GDB 后台。

□	日期	版本
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 □ 加了一个用 □ 数据指□。同□, 默□的 X □□切□□ xorg
502124	July 24, 2004	将使用 rc.d 和□□脚本的 port 分□□□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	□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 □□建函数和初始化失□之后的 5.2-CURRENT
502128	August 8, 2004	vfs_mount □ 名和全局替□ suser(9) API 的 PRISON_ROOT □ 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	□建 RELENG_5_3 分支之后的 5.3-STABLE
503101	December 3, 2004	加入了 glibc □格的 strftime(3) 填充□□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++ □□器的修正之后的 5.4-PRERELEASE
503104	February 28, 2005	MFC 了将 ifi_epoch 由 wall □□□□改□ uptime 之后的 5.4-PRERELEASE
503105	March 2, 2005	MFC 了 vswprintf(3) 中的 EOVERFLOW □□的 5.4-PRERELEASE
504000	April 3, 2005	5.4-RELEASE.

ID	日期	版本
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
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 之后的 6.0-CURRENT。 此后不久即被撤回。 不要使用它。
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 无全局的文件系统引入 MNTK_MPSAFE 标志之后的 6.0-CURRENT
600009	February 4, 2005	加入 cpufreq 框架和之后的 6.0-CURRENT
600010	February 6, 2005	引入 OpenBSD 的 nc(1) 之后的 6.0-CURRENT

□	日期	版本
600011	February 12, 2005	□去并不存在的 SVID2 <code>matherr()</code> 支持之后的 6.0-CURRENT
600012	February 15, 2005	□大默□□程堆□尺寸之后的 6.0-CURRENT
600013	February 19, 2005	□加了□□ <code><src/include/stdbool.h></code> 和 <code><src/sys/i386/include/_types.h></code> 的用于 Intel C/C++ □□器的 GCC-兼容性修正。
600014	February 21, 2005	修正了 <code>vswprintf(3)</code> 的 EOVERFLOW □□之后的 6.0-CURRENT
600015	February 25, 2005	将 <code>struct if_data</code> 成□ <code>ifi_epoch</code> 从 <code>wall</code> □□□□改□ <code>uptime</code> 之后的 6.0-CURRENT
600016	February 26, 2005	修改 <code>LC_CTYPE</code> 磁□格式之后的 6.0-CURRENT
600017	February 27, 2005	修改 <code>NLS</code> □□磁□格式之后的 6.0-CURRENT
600018	February 27, 2005	修改 <code>LC_COLLATE</code> 磁□格式之后的 6.0-CURRENT
600019	February 28, 2005	将 <code>acpica</code> □文件安装到 <code>/usr/include</code>
600020	March 9, 2005	□ <code>send(2)</code> API 加入了 <code>MSG_NOSIGNAL</code>
600021	March 17, 2005	在 <code>cdevsw</code> 上□加了一些字段
600022	March 21, 2005	基本系□中□去了 <code>gtar</code>
600023	April 13, 2005	<code>unix(4)</code> 中加入了 <code>LOCAL_CREDS</code> , <code>LOCAL_CONNWAIT</code> □个 <code>socket</code> □□
600024	April 19, 2005	加入了 hwpmc(4) 及其相□工具之后的 6.0-CURRENT
600025	April 26, 2005	加入 <code>struct icmphdr</code> 之后的 6.0-CURRENT
600026	May 3, 2005	<code>pf</code> 更新到了 3.7
600027	May 6, 2005	引入了内核 <code>libalias</code> 和 <code>ng_nat</code>
600028	May 13, 2005	将 <code>ttynam_r(3)</code> 接口改□符合 POSIX □准, 并通□ <code>unistd.h</code> 和 <code>libc</code>

□	日期	版本
600029	May 29, 2005	将 libpcap 升□ 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□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□的共享□的版本□之后的 6.0-STABLE。
600034	August 13, 2005	□ dev_clone 事件□理函数□加身□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□中的脚本集成到基本系□的 rcorder(8) 之后的 6.0-STABLE。
600102	December 30, 2005	更新 ELF □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□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。
601103	July 17, 2006	将域名解析函数更新至 BIND9, 并□出了可重入版本的 netdb 函数之后的 6.1-STABLE。
601104	August 8, 2006	在 OpenSSL 中□用了 DSO (□□共享□) 支持之后的 6.1-STABLE。

□	日期	版本
601105	September 2, 2006	由于 802.11 修正了 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	添加 pci_find_extcap() 之后的 6.2-STABLE。
602103	January 16, 2007	MFC 了 dlsym 行修改, 使其在指定 dso 及其暗指的依赖符号之后的 6.2-STABLE。
602104	January 28, 2007	MFC 了 netgraph 点 ng_deflate(4) 和 ng_pred1(4) 以及用于 ng_ppp(4) 点的新及加密模式之后的 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 了包含字符支持的 ncurses 5.6 之后的 6.2-STABLE。
602108	April 11, 2007	MFC 了 Linux SCSI SG 直通 API 子集的 CAM 'SG' 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 致文件系统模 KBI 化之后的 6.2-STABLE。

□	日期	版本
602112	September 21, 2007	一系列 libutil(3) MFC 之后的 6.2-STABLE。
602113	October 25, 2007	MFC 了 □ 字符和 □ 字 □ ctype 函数分拆之后的 6.2-STABLE。 新 □ 的引用了 ctype.h 的可 □ 行文件，可能会需要一个在旧系 □ 上不存在的 □ 符号 __mb_sb_limit。
602114	October 30, 2007	恢 □ 了 ctype ABI 向前兼容性之后的 6.2-STABLE。
602115	November 21, 2007	回退了 □ 字符和 □ 字 □ 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 □ 支持之后的 6.3-STABLE。
603102	April 24, 2008	□ flock □ 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	□ make(1) MFC :u □ 量修 □ 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 □ 的共享 □ 的版本 □ 之后的 7.0-CURRENT。
700002	August 13, 2005	□ dev_clone 事件 □ 理函数中 □ 加身 □ 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 □ 接受一 backlog 参数之后的 7.0-CURRENT。
700005	November 11, 2005	将 IFP2ENADDR() 改 □ 返回一 IF_LLADDR() 指 □ 之后的 7.0-CURRENT。

□	日期	版本
700006	November 11, 2005	在 <code>struct ifnet</code> 中加 <code>if_addr</code> 成□, 并□除 <code>IFP2ENADDR()</code> 之后的 7.0-CURRENT。
700007	December 2, 2005	将 <code>local_startup</code> 目□中的脚本集成到基本系□的 <code>rcorder(8)</code> 之后的 7.0-CURRENT。
700008	December 5, 2005	去掉 <code>MNT_NODEV</code> 挂接□□之后的 7.0-CURRENT。
700009	December 19, 2005	□ <code>ELF-64</code> □型和符号版本□行□更之后的 7.0-CURRENT。
700010	December 20, 2005	□加 <code>hostb</code> 和 <code>vgapci</code> □□、 <code>pci_find_extcap()</code> , 并将 <code>AGP</code> □□改□不再影射 <code>aperature</code> 之后的 7.0-CURRENT。
700011	December 31, 2005	除 Alpha 之外的所有平台上 <code>tv_sec</code> 改□ <code>time_t</code> 之后的 7.0-CURRENT。
700012	January 8, 2006	修改 <code>ldconfig_local_dirs</code> 之后的 7.0-CURRENT。
700013	January 12, 2006	在修改了 <code>/etc/rc.d/abi</code> 以支持 <code>/compat/linux/etc/ld.so.cache</code> 以某只□文件系□上的符号□接形式存在之后的 7.0-CURRENT。
700014	January 26, 2006	引入 <code>pts</code> 之后的 7.0-CURRENT。
700015	March 26, 2006	在引入 <code>hwpmc(4)</code> 的第 2 版 ABI 之后的 7.0-CURRENT。
700016	April 22, 2006	在 <code>libc</code> 中加入了 <code>fcloseall(3)</code> 之后的 7.0-CURRENT。
700017	May 13, 2006	□去 <code>ip6fw</code> 之后的 7.0-CURRENT。
700018	July 15, 2006	引入了 <code>snd_emu10kx</code> 之后的 7.0-CURRENT。
700019	July 29, 2006	引入了 <code>OpenSSL 0.9.8b</code> 之后的 7.0-CURRENT。
700020	September 3, 2006	□加了 <code>bus_dma_get_tag</code> 函数之后的 7.0-CURRENT。
700021	September 4, 2006	在引入了 <code>libpcap 0.9.4</code> 和 <code>tcpdump 3.9.4</code> 之后的 7.0-CURRENT。

ID	日期	版本
700022	September 9, 2006	在 <code>dlsym</code> 行修改，使其在指定 <code>dso</code> 及其暗指的依赖中 符号之后的 7.0-CURRENT。
700023	September 23, 2006	<code>OSSv4</code> 混音器 API 加入新的声音 IOCTL 之后的 7.0-CURRENT。
700024	September 28, 2006	加入 OpenSSL 0.9.8d 之后的 7.0- CURRENT。
700025	November 11, 2006	加入了 <code>libelf</code> 之后的 7.0- CURRENT。
700026	November 26, 2006	音效相关的 <code>sysctl</code> 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 <code>libc</code> 中加入 <code>sctp</code> 用之后的 7.0- CURRENT。
700029	January 26, 2007	将 GNU gzip(1) 替换从 NetBSD 移植的采用 BSD 授权版本之后的 7.0-CURRENT。
700030	February 7, 2007	在 IPv4 多播代码中去了 IPIP 隧道封装 (<code>VIFF_TUNNEL</code>) 之后的 7.0-CURRENT。
700031	February 23, 2007	修改了 <code>bus_setup_intr()</code> (<code>newbus</code>) 之后的 7.0-CURRENT。
700032	March 2, 2007	引入了 <code>ipw(4)</code> 和 <code>iwi(4)</code> 固件之后的 7.0-CURRENT。
700033	March 9, 2007	在 <code>ncurses</code> 中引入了 字符支持之后的 7.0-CURRENT。
700034	March 19, 2007	修改了 <code>insmntque()</code> 、 <code>getnewvnode()</code> 以及 <code>vfs_hash_insert()</code> 工作方式之后的 7.0-CURRENT。
700035	March 26, 2007	增加 CPU 率通知机制之后的 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	新加了 Linux SCSI SG 直通 API 子集的 CAM 'SG' 之后的 7.0-CURRENT。

□	日期	版本
700038	April 30, 2007	将 getenv(3) 、 putenv(3) 、 setenv(3) 和 unsetenv(3) 改□符合 POSIX 之后的 7.0-CURRENT。
700039	May 1, 2007	回退了 700038 中的□□之后的 7.0-CURRENT。
700040	May 10, 2007	在 libutil 中□加了 flopen(3) 之后的 7.0-CURRENT。
700041	May 13, 2007	□用了符号版本，并将 libthr 改□默□程□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□版本的共享□版本□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□下□改□ 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□号管理功能而非身□□□功能之后的 7.0-CURRENT。
700046	June 11, 2007	更新 802.11 无□支持之后的 7.0-CURRENT。
700047	June 11, 2007	□加 TCP LRO 网□接口能力之后的 7.0-CURRENT。
700048	June 12, 2007	在 IPv4 □□□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□被□去；0.0.0.0/8 不再能□用于指定接口索引下□，而□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)		□ FAST_IPSEC □加 IPv6 支持，□去 KAME IPSEC，并将 FAST_IPSEC 更名□ IPSEC 之后的 7.0-CURRENT。(未□□)
700050	July 4, 2007	将 setenv/putenv/等等□用，从□□BSD 改□ POSIX □准之后的 7.0-CURRENT。

□	日期	版本
700051	July 4, 2007	□加新的 mmap/lseek/等等□些系□□用之后的 7.0-CURRENT。
700052	July 6, 2007	将 I4B □文件移□到 include/i4b 之后的 7.0-CURRENT。
700053	September 30, 2007	□加了 PCI domain 支持之后的 7.0-CURRENT。
700054	October 25, 2007	MFC 了□字符和□字□字符 ctype 分拆之后的 7.0-CURRENT。
700055	October 28, 2007	7.0-RELEASE, 以及 MFC 了恢□□ FreeBSD 4/5/6 版本的 PCIOCGETCONF、PCIOCREAD 和 PCIOCWRITE IOCTL ABI 向下兼容之后的 7.0-CURRENT, □一□□致 PCIOCGETCONF IOCTL 的 ABI 再次□生□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	□ flock □□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	□ psm(4) MFC write(2) 支持之后的 7.0-STABLE。
700107	April 20, 2008	□ fcntl(2) MFC F_DUP2FD 之后的 7.0-STABLE。
700108	May 5, 2008	□ lockmgr(9) 做了一些修改之后的 7.0-STABLE, 在使用 lockmgr(9) □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□端之后的 7.0-STABLE。
700111	August 20, 2008	加入了□物理□巨□支持之后的 7.0-STABLE。

□	日期	版本
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 <code>strndup</code> 之后的 7.1-STABLE。
701102	January 17, 2009	加入了 <code>cpuctl(4)</code> 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 <code>struct mount</code> 中保存了挂起属主，以及在 <code>struct vfsops</code> 中引入了 <code>vfs_susp_clean</code> 方法之后的 7.1-STABLE。
701105	March 12, 2009	□ <code>kern.ipc.shmsegs</code> <code>sysctl</code> □量不兼容的修改，以允□在 64bit □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□ 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	<code>ichsmb(4)</code> 改□使用左□接 □□址来保持与其它 SMBus 控制器□□一致性之后的 7.2-STABLE。
702102	May 28, 2009	MFC 了 <code>fdopendir</code> 函数之后的 7.2-STABLE。
702103	June 06, 2009	MFC 了 <code>PmcTools</code> 之后的 7.2-STABLE。
702104	July 14, 2009	MFC 了 <code>closefrom</code> 系□□用之后的 7.2-STABLE。
702105	July 31, 2009	MFC 了 SYSVIPC ABI 改□之后的 7.2-STABLE。

□	日期	版本
702106	September 14, 2009	MFC 了 x86 PAT □□, 并新□了 d_mmap_single() 以及 scatter/gather 型 VM □象 □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□字符和 □字□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□ kproc_create() 之后的 8.0-CURRENT。
800003	October 24, 2007	恢□了□ FreeBSD 4/5/6 版本的 PCIOCGETCONF、PCIOCREAD 和 PCIOCWRITE IOCTL ABI 向下兼容之后的 8.0-CURRENT, □一□□致 PCIOCGETCONF IOCTL 的 ABI 再次□生□化。
800004	November 12, 2007	将 agp(4) □□从 src/sys/pci □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□ hwpmc(4) 加入了 callgraph 捕捉功能后的 8.0-CURRENT
800007	December 25, 2007	kdb_enter() □加 "why" 参数之后的 8.0-CURRENT。
800008	December 28, 2007	在去除 LK_EXCLUPGRADE □□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。

□	日期	版本
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□除了 BUF_REFCNT() 之后的 8.0-CURRENT。
800013	January 23, 2008	引入 "ASCII" □□之后的 8.0-CURRENT。
800014	January 24, 2008	修改 lockmgr(9) 并□除了 lockcount() 和 LOCKMGR_ASSERT() 之后的 8.0-CURRENT。
800015	January 26, 2008	□展了 fts(3) 数据□□之后的 8.0-CURRENT。
800016	February 1, 2008	□ MEXTADD(9) □加了一个参数之后的 8.0-CURRENT。
800017	February 6, 2008	□ lockmgr(9) 引入 LK_NODUP 和 LK_NOWITNESS □□后的 8.0-CURRENT。
800018	February 8, 2008	引入 m_collapse 之后的 8.0-CURRENT。
800019	February 9, 2008	□ sysctl □量 kern.proc.filedesc 加入 当前工作目□, root 目□和 jail 目□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 □志之后的 8.0-CURRENT。
800022	(backed out)	把 BSD ar(1) 作□系□默□的 ar 之后的 8.0-CURRENT。
800023	February 25, 2008	修改了 lockstatus(9) 和 VOP_ISLOCKED(9) ; 原型, 特□□去掉 struct thread 参数之后的 8.0-CURRENT。

ID	日期	版本
800024	March 1, 2008	移除了 <code>lockwaiters</code> 和 <code>BUF_LOCKWAITERS</code> 函数, <code>bre1vp</code> 的返回类型从 <code>void</code> 修改成 <code>int</code> , 并引入 <code>lockinit(9)</code> 新标志之后的 8.0-CURRENT。
800025	March 8, 2008	通过 <code>fcntl(2)</code> 引入 <code>F_DUP2FD</code> 之后的 8.0-CURRENT。
800026	March 12, 2008	修改了 <code>cv_broadcastpri</code> 标志参数之后的 8.0-CURRENT, 比如 0 表示无标志。
800027	March 24, 2008	修改了 <code>bpf</code> 的 ABI, 加入了 <code>zerocopy bpf buffer</code> 之后的 8.0-CURRENT。
800028	March 26, 2008	通过 <code>flock</code> 标志加了 <code>l_sysid</code> 之后的 8.0-CURRENT。
800029	March 28, 2008	重新整合了 <code>BUF_LOCKWAITERS</code> 函数并加入 <code>lockmgr_waiters(9)</code> 之后的 8.0-CURRENT。
800030	April 1, 2008	引入 <code>rw_try_rlock(9)</code> 和 <code>rw_try_wlock(9)</code> 之后的 8.0-CURRENT。
800031	April 6, 2008	引入 <code>lockmgr_rw</code> 和 <code>lockmgr_args_rw</code> 函数之后的 8.0-CURRENT。
800032	April 8, 2008	增加了 <code>openat</code> 和相关的系统调用, 通过 <code>open(2)</code> 引入了 <code>O_EXEC</code> 标志, 和提供了相关的 <code>linux</code> 兼容的系统调用之后的 8.0-CURRENT。
800033	April 8, 2008	通过 <code>psm(4)</code> 增加了原生的 <code>write(2)</code> 支持之后的 8.0-CURRENT。 可以在任意命令可写入 <code>/dev/psm%d</code> 并输出状态。
800034	April 10, 2008	引入 <code>memrchr</code> 函数之后的 8.0-CURRENT。
800035	April 16, 2008	引入 <code>fdopendir</code> 函数之后的 8.0-CURRENT
800036	April 20, 2008	无标志部分向 <code>multi-bss</code> (也叫做 <code>vaps</code>) 支持之后的 8.0-CURRENT。

□	日期	版本
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	□去了 netatm 和 ISDN4BSD 后的 8.0-CURRENT。□个版本也表示 □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	□加了内核 □ NFS lockd 客 □端的 8.0-CURRENT。
800041	July 22, 2008	□加了 arc4random_buf(3) 和 arc4random_uniform(3) 之后的 8.0-CURRENT。
800042	August 8, 2008	□加了 cpuctl(4) 之后的 8.0-CURRENT。
800043	August 13, 2008	修改 bpf(4) 使用 □一的 □□点而不是克隆之后的 8.0-CURRENT。
800044	August 17, 2008	在提交了 vimage □目第一 □之后的 8.0-CURRENT。把全局 □量重命名 □虚 □化 □ V_ 前 □并用宏映射到原来的全局名称。
800045	August 20, 2008	引入 MPSAFE TTY □之后的 8.0-CURRENT, 包括 □相 □□□和工具的修改。
800046	September 8, 2008	将 amd64 架 □上 GDT 拆分到不同 CPU 之后的 8.0-CURRENT。
800047	September 10, 2008	□去了 VSVTX、VSGID 和 VSUID 之后的 8.0-CURRENT。
800048	September 16, 2008	将内核中 NFS 挂接部分的代 □改 □能 □通 □ nmount() iovec, 而不再是大的 nfs_args □□体作 □参数之后的 8.0-CURRENT。
800049	September 17, 2008	□去了 suser(9) 和 suser_cred(9) 之后的 8.0-CURRENT。
800050	October 20, 2008	修改了 □冲存 □器 API 之后的 8.0-CURRENT。
800051	October 23, 2008	□去了 MALLOC(9) 和 FREE(9) 宏之后的 8.0-CURRENT。

□	日期	版本
800052	October 28, 2008	引入了 accmode_t 和重新命名 VOP_ACCES 'a_mode' □ 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 □志之后的 8.0-CURRENT。
800054	November 22, 2008	□加了 buf_ring、内存□以及 ifnet 函数， 以方便撰写支持多硬件 □□□列的□□， 以及无□□形□冲□□的 □□程序， 并更高效地管理包 □列功能之后的 8.0-CURRENT。
800055	November 27, 2008	引入了 hwpmc(4) □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□使用源代□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	□去了 minor()、 minor2unit()、 unit2minor() 等之后的 8.0-CURRENT。
800063	February 18, 2009	在 GENERIC 配置中改□使用 USB2 □之后的 8.0-CURRENT； □个数□同□也□志新□了 fdevname(3)。
800064	February 23, 2009	将 USB2 □移□并替□ dev/usb 之后的 8.0-CURRENT。
800065	February 26, 2009	在□ libmp(3) 中所有函数更名之后的 8.0-CURRENT。

□	日期	版本
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(), strlen(), wcsnlen(), wcscasecmp(), 和 wcsncasecmp() 之后的 8.0-CURRENT。
800068	March 2, 2009	在 usbbus devclass 更名□ uhub 之后的 8.0-CURRENT。
800069	March 9, 2009	重命名 libusb20.so.1 □ libusb.so.1 之后的 8.0-CURRENT。
800070	March 9, 2009	合并 IGMPv3 和 Source-Specific Multicast (SSM) 入 IPv4 □之后的 8.0-CURRENT。
800071	March 14, 2009	□ gcc 打上了在 c99 和 gnu99 模式中使用 C99 inline □□丁之后的 8.0-CURRENT。
800072	March 15, 2009	移除了 IFF_NEEDSGIANT □志; 不再支持非□程安全的网□□□□□之后的 8.0-CURRENT。
800073	March 18, 2009	□□了 rpath □□字符替□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 □□布局之后的 8.0-CURRENT。
800076	April 9, 2009	□ dummynet 新□了延 □□估工具之后的 8.0-CURRENT。
800077	April 14, 2009	□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, □致其□□生 □化之后的 8.0-CURRENT。此后 RTM_VERSION □加, 但又回退了。

□	日期	版本
800079	April 15, 2009	在 struct route 和 struct_in6 中添加了 struct llentry 指□之后的 8.0-CURRENT。
800080	April 15, 2009	改□了 struct inpcb 布局之后的 8.0-CURRENT。
800081	April 19, 2009	改□了 malloc_type 布局之后的 8.0-CURRENT。
800082	April 21, 2009	改□了 struct ifnet 布局， 并□加了 if_ref() 和 if_rele() 引用□数□□功能之后的 8.0-CURRENT。
800083	April 22, 2009	□□了底□□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	□用了包括一个活□映像的 VIMAGE 内核支持之后的 8.0-CURRENT。
800086	May 8, 2009	□ patch(1) □加任意□□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□去了□程参数。VFS_* 函数并不需要□些上下文信息， 因□它□是与 curthread 相□。在某些特殊情况中，□保留了原先的行□。
800088	May 20, 2009	□ net80211 □□模式□行□整之后的 8.0-CURRENT。
800089	May 23, 2009	□加了 UDP 控制□支持之后的 8.0-CURRENT。
800090	May 23, 2009	将网□接口克隆虚□化之后的 8.0-CURRENT。
800091	May 27, 2009	□加了□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□的 SX_NOADAPTIVE □志□表□相反□□。

ID	日期	版本
800093	May 29, 2009	添加 struct mount 中的 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 KPI (polling KPI) 之后的 8.0-CURRENT。网络驱动程序会返回处理包的包的数量。新增的 IFCAP_POLLING_NOCOUNT 表示返回不重要，并跳数。
800096	June 1, 2009	新的 netisr 进行了改，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 析挂和相基施之后的 8.0-CURRENT。
800097	June 11, 2009	引入了 netgraph 出到入路径用和排机制，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 并将路由虚拟化到它自己的 VImage 模之后的 8.0-CURRENT。
800100	June 24, 2009	修改了 SYSVIPIC ABI 之后的 8.0-CURRENT。
800101	June 29, 2009	去了与网接口一一的 /dev/net/* 字符之后的 8.0-CURRENT。
800102	July 12, 2009	在 struct sackhint、struct tcpcb 以及 struct tcpstat 上加占位元素之后的 8.0-CURRENT。
800103	July 13, 2009	将 TOE 接口中的 struct tcptopt 替换 TCP syncache 中的 struct toept 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	了所有未使用符号版本的接口版本之后的 8.0-CURRENT。

ID	日期	版本
800106	July 24, 2009	引入 VM 象型 OBJT_SG 之后的 8.0-CURRENT。
800107	August 2, 2009	通加入 newbus sxlock 使 newbus 子系不再使用 Giant, 以及 8.0-RELEASE。
800108	November 21, 2009	了 EVFILT_USER kevent 器之后的 8.0-STABLE。
800500	January 7, 2010	令 <code>pkg_add -r</code> 使用 packages-8-stable 的 <code>__FreeBSD_version</code> 版本化的 8.0-STABLE。
800501	January 24, 2010	整了 <code>scandir(3)</code> 和 <code>alphasort(3)</code> 函数原型, 使其符合 SUSv4 之后的 8.0-STABLE。
800502	January 31, 2010	新了 <code>sigpause(3)</code> 之后的 8.0-STABLE。
800503	February 25, 2010	新了用于管理网接口明的 SIOCGIFDESCR 和 SIOCSIFDESCR ioctl 之后的 8.0-STABLE。接口受到了 OpenBSD 的。
800504	March 1, 2010	MFC 了 x86emu, 来自 OpenBSD 的 x86 CPU 模式模拟器之后的 8.0-STABLE。
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, 包含用跟踪支持之后的 8.2-STABLE。

□	日期	版本
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	引入模块化阻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 □□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	□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	□ <code>rfork(2)</code> 添加了 RFTSIGZMB □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 □模式模拟器之后的 9.0-CURRENT。
900002	September 23, 2009	□□了 EVFILT_USER kevent □□器之后的 9.0-CURRENT。
900003	December 2, 2009	新□了 <code>sigpause(3)</code> 以及 csu 的 PIE 支持之后的 9.0-CURRENT。
900004	December 6, 2009	新□了 libulog 及其 libutempter 兼容接口之后的 9.0-CURRENT。
900005	December 12, 2009	新□了用于□□指定休眠□列上等待者数量的 <code>sleepq_sleepcnt()</code> 函数之后的 9.0-CURRENT。
900006	January 4, 2010	□整了 <code>scandir(3)</code> 和 <code>alphasort(3)</code> 函数原型, 使其符合 SUSv4 之后的 9.0-CURRENT。
900007	January 13, 2010	□去了 utmp(5) 并□加了 utmpx (参□ <code>getutxent(3)</code>) 以改善用□登 □日志和系□事件支持之后的 9.0-CURRENT。

ID	日期	版本
900008	January 20, 2010	9.0-CURRENT 引入了 BSD 授权的 bc/dc 并将 GNU bc/dc 注释到之后的 9.0-CURRENT。
900009	January 26, 2010	新增了用于管理网络接口明确的 SIOCGIFDESCR 和 SIOCSIFDESCR ioctl 之后的 9.0-CURRENT。该接口受到了 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 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 并添加 WITH_BSD_GREP 选项之后的 9.0-CURRENT。
900018	August 24, 2010	将 pthread_kill(3) 产生的信号在 si_code 中改回使用 SI_LWP 选项之后的 9.0-CURRENT。之前, si_code 选项的标志为 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 功能并改回了 struct sbuf 布局之后的 9.0-CURRENT。
900021	September 13, 2010	DTrace 增加了对跟踪支持之后的 9.0-CURRENT。

□	日期	版本
900022	October 2, 2010	新□了 BSDL man 工具，并淘汰 GNU/GPL man 工具之后的 9.0-CURRENT。
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□ libcompiler_rt.a 之后的 9.0-CURRENT。
900025	November 12, 2010	引入了模□化□塞控制之后的 9.0-CURRENT。
900026	November 30, 2010	引入串行管理□□ (SMP) 直通，以及与之□□的 CAM CCB XPT_SMP_IO 和 XPT_GDEV_ADVINFO 之后的 9.0-CURRENT。
900027	December 5, 2010	在 libm 中□加 log2 之后的 9.0-CURRENT。
900028	December 21, 2010	添加了 Hhook (Helper Hook)、Khhelp (Kernel Helpers) 和 Object Specific Data (OSD) KPI 之后的 9.0-CURRENT。
900029	December 28, 2010	修改 TCP □□□使其允□ Khhelp 模□通□ helper hook 指□，与 TCP 控制□交互并保存□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	□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□中 gcc 和 libstdc++ 升□至最后的 GPLv2 授□版本之后的 9.0-CURRENT。

□	日期	版本
900036	April 18, 2011	在基本系□中□去了 libobjc 和 Objective-C 支持之后的 9.0-CURRENT。
900037	May 13, 2011	在基本系□中引入了 libprocstat(3) 函数□以及 fuser(1) 工具之后的 9.0-CURRENT。
900038	May 22, 2011	□ VFS_FHTOVP(9) 添加 □□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。
900040	July 19, 2011	将 amd64 和 ia64 平台上的 MAXCPU 提高到 64, 并把 XLP (mips) 上的□提高到 128 之后的 9.0-CURRENT。
900041	August 13, 2011	□□了 Capsicum capabilities 之后的 9.0-CURRENT。fget(9) 新□了□限参数。
900042	August 28, 2011	提高修改□ ABI 的□□□接 □版本号之后的 9.0-CURRENT。
900043	September 2, 2011	□加了□不支持 SCSI 快取□存同 □功能的 USB 大容量存□□□自 □□□功能之后的 9.0-CURRENT。
900044	September 10, 2011	重□了 auto-quirk 之后的 9.0-CURRENT。
900045	Oct 13, 2011	将非兼容性系□□用入口点全部□加 sys_ 前□之后的 9.0-CURRENT。



□注意, 2.2.5-RELEASE 之后有一段□□的 2.2-STABLE 会声称自己是 "2.2.5-STABLE"。□□模式的版本号表示的是年月。但随后, 我□决定, 从 2.2 □始, 将它改□更□□□的主/次版本号的形式来命名版本。□是因□并行地在多个分支上□行□□, □使得通□□□的 □布日期来区分不同的版本□得不再□□。如果□正在做新的 port, □□不需要担心□早的 -CURRENT ; 在此列出□供参考。

12.6. 在 `bsd.port.mk` 之后写一些内容

不要在 `.include <bsd.port.mk>` □行之后□加任何内容。□通常可以通□在□的 Makefile 中□的某□引用 `bsd.port.pre.mk`, 并在□尾的地方引用 `bsd.port.post.mk` 来避免。



只能□采用 `bsd.port.pre.mk/bsd.port.post.mk` 或 `bsd.port.mk` □□写法之一 ; 任何 □候都不要同□使用□□写法。

`bsd.port.pre.mk` 只定□了很少的□量, 它□可以在 Makefile 中用于□行一些□□, 而 `bsd.port.post.mk` □定

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（并不是一完整的列表，可以得到全部变量的名字）。

变量	描述
<code>ARCH</code>	由 <code>uname -m</code> 输出得到的硬件架构的名字 (例如, <code>i386</code>)
<code>OPSYS</code>	由 <code>uname -s</code> 返回的操作系统类型 (例如, <code>FreeBSD</code>)
<code>OSREL</code>	操作系统的版本号 (例如 <code>2.1.5</code> 或 <code>2.2.7</code>)
<code>OSVERSION</code>	操作系统的版本号的数字形式；它等于 <code>__FreeBSD_version</code> 。
<code>PORTOBJFORMAT</code>	系统默认的归档文件格式 (<code>elf</code> 或 <code>aout</code> ；注意，"取代的" <code>FreeBSD</code> 版本中, <code>aout</code> 已在淘汰之列。)
<code>LOCALBASE</code>	"local" 目录的根 (例如, <code>/usr/local/</code>)
<code>PREFIX</code>	<code>port</code> 被安装到哪里 (参看 关于 PREFIX 的更多说明)。



如果需要定义 `USE_IMAKE`, `USE_X_PREFIX`, 或 `MASTERDIR` 这些变量，请在引用 `bsd.port.pre.mk` 之前完成。

下面是一些在引用 `bsd.port.pre.mk` 之后可以执行的判断：

```
# 如果 perl5 已在系统中提供，不必安装 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

# 组件会自行 ELF 构建符号链接，但 a.out 需要自行构建
post-install:
.if ${PORTOBJFORMAT} == "aout"
${LN} -sf liblinpack.so.1.0 ${PREFIX}/lib/liblinpack.so
.endif
```

记得在 `BROKEN=` 和 `TCL_LIB_FILE=` 后面使用制表符，而不是空格，对吗？:-)

12.7. 在 wrapper 脚本中使用 `exec` 语句

如果 `port` 安装了用以运行其他程序的脚本，并且运行其他程序是这些脚本的最后一操作，则必须使用 `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` 语句表示行指定的程序来取代 shell 进程。如果省略了 `exec`，shell 进程会一直在内存中，从而不必要地消耗了外的系源。

12.8. 理性行事

任何 Makefile 都并理性地行事。如果能其中的条目更和易，一定要做。例如，使用 make 提供的 `.if`，而不要使用 shell 的 `if`，只要能重定 `EXTRACT*` 就不要重 `do-extract`，尽量使用 `GNU_CONFIGURE` 而不是 `CONFIGURE_ARGS += --prefix=${PREFIX}`。

如果在做什么事情的时候不得不写大量的代码，回来一下 `bsd.port.mk`，看看是否有正打算做的事情的成。尽管起来可能很，但有很多貌似很的，在 `bsd.port.mk` 中都出了十分便的解决方案。

12.9. 遵循 CC 和 CXX 置

port 遵循 `CC` 和 `CXX` 量的置。也就是，port 不使用的方式来置个量的，而罔已存在的置；与此相反，它在其后加入需要的其它。，就可以置全局的，令其影所有的 port 程了。

如果在无法做，在 Makefile 中加入 `NO_PACKAGE=ignores cflags`。

下面的 Makefile 例出了如何遵循 `CC` 和 `CXX` 量的置。注意里用到的 `?:=`：

```
CC?= gcc
```

```
CXX?= g++
```

下面是没有遵循 `CC` 和 `CXX` 的例子：

```
CC= gcc
```

```
CXX= g++
```

在 FreeBSD 系中，`CC` 和 `CXX` 个量都可以在 `/etc/make.conf` 中自行定。第一个例子只有在 `/etc/make.conf` 中没有定才量行定，从而保持了系配置。而第二个例子会覆任何的配置。

12.10. 遵循 CFLAGS

的 port 遵循 `CFLAGS` 量的置。也就是，port 不使用的方式来置个量的，而罔已存在的

置；与此相反，它可以在其后加入需要的其它，，就可以置全局的，令其影响所有的 port 程序了。

如果在无法做，在 Makefile 中加入 `NO_PACKAGE=ignores cflags`。

下面的 Makefile 例子，可以帮助我理解如何遵循 `CFLAGS` 的置。注意所用的 `+=`：

```
CFLAGS+= -Wall -Werror
```

下面是一个未能遵循 `CFLAGS` 置的例子：

```
CFLAGS= -Wall -Werror
```

一般来，`CFLAGS` 在 FreeBSD 系中是在 `/etc/make.conf` 里配置的。第一个例子在 `CFLAGS` 量中加了一些参数，并保持了所有系定义的标志。而第二个例子，会覆盖掉任何先前定义参数。

从第三方件的 Makefile 中去掉特殊的化置。系的 `CFLAGS` 出了全系内的化置参数。下面是一个未修改的 Makefile 例：

```
CFLAGS= -O3 -funroll-loops -DHAVE_SOUND
```

如果使用系的化参数，在 Makefile 中的置应如下面：

```
CFLAGS+= -DHAVE_SOUND
```

12.11. 程序

在 FreeBSD 上，程序必须通特殊的连接器参数 `-pthread` 接到可执行文件。如果 port 一定要直接接 `-lpthread` 或 `-lc_r`，应将其改使用由 ports 框架提供的 `PTHREAD_LIBS`。个量的通常是 `-pthread`，但在某些特定平台上的 FreeBSD 版本中，它可能是其它，因此，不要将 `-pthread` 硬到代码中，而使用 `PTHREAD_LIBS` 量。



如果置了 `PTHREAD_LIBS`，而在代码出 `unrecognized option '-pthread'` 的，可能需要通将 `CONFIGURE_ENV` 中 `LD=${CC}` 来使用 `gcc` 作连接器。 `-pthread` 一并不 `ld` 所直接支持。

12.12. 反

如果行了一些很好的修改和，一定要把它回原作者，或者，以便在下一版本的代码中包含它。会在件布新版本的候得松一些。

12.13. README.html

不要包含 README.html 文件。该文件并非 CVS 代碼中的一部分，它是由 `make readme` 命令生成的。

12.14. 使用 **BROKEN**、**FORBIDDEN** 或 **IGNORE** 阻止用 `port` 安装 `port`

某些时候会需要阻止用 `port` 安装某个 `port`。想要告诉用 `port` 某个 `port` 不被安装，有多可以在 `port` 的 Makefile 中使用的 `make` 变量。下列 `make` 的变量，将是在用 `port` 安装 `port` 得到的提示信息。使用正值的 `make` 变量，因一个都表达了截然不同的意思，而且多自自动化系，例如 `port` 集群、`FreshPorts`，以及 `portsmon`，都依赖于 Makefile 的正值性。

12.14.1. 变量

- **BROKEN** 用于表示目前无法正、安装或卸。如果是性的，可以使用它。

如果行了相的配置，集群仍将它，以致致的深是否已被解决。（不，一般情况下，集群并不会做。）

例如，当 `port` 生下述情况，使用 **BROKEN**：

- 无法 (does not compile)
- 无法正行配置或安装操作
- 在 `${LOCALBASE}` 以外的地方安装文件
- 卸无法除所安装的全部文件（不，留下用改的文件可接受的，因可能希望作）
- **FORBIDDEN** 用于表示 `ports` 中包含安全漏洞，或者可能会安装了个 `port` 的 FreeBSD 系来重的安全隐患（例如：一个很不安全的程序，或包含了能被易攻陷的服的文件）。如果了安全漏洞，而其作者没有布升版本，立即把那个 `port` 置 **FORBIDDEN**。理想情况下，包含安全漏洞的 `port` 被尽快升，以便少包含漏洞的 FreeBSD 主机的数量（我希望保持良好的安全），然而，有在安全漏洞的披露和件更新之可能会有一个隔，此予以明。除了安全之外，不要以任何其它理由将 `port` 置 **FORBIDDEN**。
- **IGNORE** 用来表示 `port` 由于某些其它原因不予以。如果生了性的，使用它。任何情况下，集群都不会置 **IGNORE** 的 `port`。以下是使用 **IGNORE** 的一些例子：
 - 能但无法正常行
 - 无法与行的 FreeBSD 版本一同工作
 - 需要 FreeBSD 内核的源代码，但用没有安装它
 - 由于授原因，必手工下 `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 `www/apache13`）



如果 `port` 与某个已安装的 `port` 冲突（例如，它在同一位置安装同名但功能不同的文件），使用 **CONFLICTS** 来它。 **CONFLICTS** 将自地置 **IGNORE**。

- 如果 `port` 只在某些平台上置 **IGNORE**，有外个方便使用的 **IGNORE** 变量可供：**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，
可以使用 `ONLY_FOR_ARCHS_REASONARCH_` 和 `NOT_FOR_ARCHS_REASONARCH_` 来分指定与具体平台有
的信息。

- 如果 `port` 会下并安装用于 `i386` 的二进制文件，设置 `IA32_BINARY_PORT`。如果置了个量，系
会是否已在 `/usr/lib32` 目中安装了 `IA32` 版本的函数，以及内核是否提供了 `IA32` 兼容支持。
如果些依条件不足，会自置 `IGNORE`。

12.14.2. 说明

些字串不使用引号括起来。此外，由于示用的方式不同，些字串的措辞也有所不同。例如：

```
BROKEN= this port is unsupported on FreeBSD 5.x
```

```
IGNORE= is unsupported on FreeBSD 5.x
```

它分会在 `make describe` 生下面的出：

```
==> 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` 将被除

一定要得 `BROKEN` 和 `FORBIDDEN` 只作当某个 `port` 无法正常工作解决方案。永久性地坏掉了的 `port`
被从 `ports tree` 中完全除。

需要可以使用 `DEPRECATED` 和 `EXPIRATION_DATE` 来通知用某个 `port` 不被使用，并即将被除。前一个
量用来表什除 `port`；而后一个是是一个 `ISO 8601` 格式的日期 (`YYYY-MM-DD`)。者都会向用呈。

也可以置 `DEPRECATED` 而不出 `EXPIRATION_DATE` (例如，建使用某个新版本的 `port`)，但反之没有意。

目前没有切的于需要出多少通知的政策。当前的践是，于与安全有的一个月，而与有的
的个月。也有有趣的 `committer` 能有一点来修正。

12.16. 避免使用 `.error`

在 Makefile 中输出信号，表示由于某个外界因素（例如，用指定了无效的）而无法安装的方法是将量 `IGNORE` 一非空。个将被格式化，并在用行 `make install` 是出提示。

用 `.error` 一目的是一常的。做的多是，多在 ports 上行的自化工具会因此而失。最常的情况于 `/usr/ports/INDEX` 的程（参行 [make describe](#)）。然而，即使十分普通的命令，例如 `make maintainer`，在情况下也会失。是不可接受的。

例 25. 避免使用 `.error`

考有人在 `make.conf` 中置了

```
USE_POINTYHAT=yes
```

的情形。接下来的例子中，第一个 Makefile 中的将致 `make index` 失，而第二个不会：

```
.if USE_POINTYHAT
.error "POINTYHAT is not supported"
.endif
```

```
.if USE_POINTYHAT
IGNORE=POINTYHAT is not supported
.endif
```

12.17. 于 `sysctl` 的使用

除了在 target 中之外，是不鼓励使用 `sysctl` 的。是因算 `makevar`，例如在 `make index` 中所行的那，都不得不行一条命令，会使一操作得更慢。

在使用 [sysctl\(8\)](#)，必通 `SYSCTL` 量来行，因此量将展成命令的完整路径，并且用可以根据需要行指定。

12.18. 重新布的 `distfiles`

有，一些件的作者会修改已布的 `distfile` 的内容，而并不修改文件名。情况下，需要些是来自件作者的官方改。在去，曾生下服务器上的 `distfile` 被悄悄成注入意代的版本，并用安全造成威或害的事情。

保留一旧的 `distfile`，并下一新的，分展，用 [diff\(1\)](#) 来比其内容。如果没有可疑的，就可以更新 `distinfo` 了。必在的 PR 或 commit log 中些差行描述，以便人了解已仔比比，并没有了。

除此之外，也可以系件的作者，以些修改是否是他做的。

12.19. 注意

需要仔细地反查 `pkg-descr` 和 `pkg-plist` 两个文件。如果你正在做一个 `port`，并修改这两个文件，你一定要这么做。

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！不要让我宣布没有得到合法授权的软件！

Chapter 13. 示例的 Makefile

这里是一个可以在建立新 port 时参考的 Makefile。你必须删除不必要的那些注释 (方括号中的文字)！

建议按照下面示例的格式 (数量顺序, 小写字母的空行等) 来编写。这个格式的作用是便于重要的信息。我们建议使用 [portlint](#) 来检查 Makefile。

```
[这部... 主要是让我更容易地分辨不同的 port。]
# New ports collection makefile for:  xdvi
[版本行, 只有在 PORTVERSION 量不足以描述 port 时才需要]
# Date created:                26 May 1995
[是最初将软件移植到 FreeBSD 上的日期, 一般来是建立 Makefile 的日期。]
[注意不要在之后再次修改这个日期。]
# Whom:                        Satoshi Asami <asami@FreeBSD.org>
#
# $FreeBSD$
[ ^^^^^^^^^ 是 CVS 在文件 commit 到我代码时, 自行替换的 RCS ID。]
[如果正在升级 port, 不要把它改回 "$FreeBSD$".]
[CVS 会自行处理。]
#

[这个小描述 port 本身以及主要下站点 - PORTNAME 和 PORTVERSION]
[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, 接下来指定]
[PKGNAMEPREFIX 和 PKGNAMESUFFIX。随后是 DISTNAME, EXTRACT_SUFX,]
[以及 DISTFILES, EXTRACT_ONLY, 如果需要的。]
PORTNAME=      xdvi
PORTVERSION=   18.2
CATEGORIES=    print
[如果不想使用 MASTER_SITE_* 宏, 一定不要忘记尾的斜杠 ("/")!]
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, 就需要置这个]
EXTRACT_SUFX=  .tar.Z

[分散的补丁 -- 可以空]
PATCH_SITES=   ftp://ftp.sra.co.jp/pub/X11/japanese/
PATCHFILES=    xdvi-18.patch1.gz xdvi-18.patch2.gz

[维护人(maintainer); *必须有*! 是某个源维护 port 更新、丢失,]
[以及回答或直接提或 bug 的人。为了保 Ports Collection]
[有尽可能高的品质, 我不再接受指定 "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 -- 可以空]
RUN_DEPENDS=    gs:${PORTSDIR}/print/ghostscript
LIB_DEPENDS=    Xpm.5:${PORTSDIR}/graphics/xpm
```

```

[[]是其它不[]合上几[]的[]准 bsd.port.mk []量]
[如果需要在 configure、 build 或 install []程中提[]...]
IS_INTERACTIVE=      yes
[如果解[]到 ${DISTNAME} 以外的目[]...]
WRKSRC=               ${WRKDIR}/xdvi-new
[如果作者[]布的[]丁不是相[]于 ${WRKSRC} 的, 可能需要[]整[]个]
PATCH_DIST_STRIP=    -p1
[如果需要[]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[]...]
USE_GMAKE=            yes
[如果是一个 X []用程序, 并使用 "xmkmf -a" 来[]行...]
USE_IMAKE=            yes
[et cetera.]

[将在接下来的部分使用的非[]准的[]量]
MY_FAVORITE_RESPONSE= "yeah, right"

[接下来是特殊[], 按[]用[]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

[[]]
.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

最普遍的了解已被 commit 到 ports 中的更新的方法，是 [FreshPorts](#)。它可以同步多个 ports 并使其运行。强烈建议每个人同步它，这样就不会接收到他自己所做的修改，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，虽然一般来说会在同步的 commit 之前收到一个礼貌性的通知，但有可能会有人没有注意到需要同步做，或者同步做很困难。另外，有些时候通知的修改也可能是微不足道的。我希望一个人能正确地判断。）

如果想使用 FreshPorts，首先需要建立一个账号。如果注册的邮件地址是 [@FreeBSD.org](#)，将会看到 web 页面右侧的 opt-in 链接。如果已注册了 FreshPorts 账号，但没有使用 [@FreeBSD.org](#) 邮件地址，只需把邮件地址改为 [@FreeBSD.org](#)，重新同步，并将其改回。

FreshPorts 也会同步一个 FreeBSD ports tree 上的 commit 行为的合法性。如果同步了服务，如果同步了，就会收到来自 FreshPorts 的通知。

14.2. 代理的 Web 界面

可以通过 web 界面来同步源代码中的文件。影响整个 ports 系统的修改，都会在 [CHANGES](#) 文件中注明。影响某一个 port 的修改，在 [UPDATING](#) 文件中注明。尽管如此，所有最权威的答案，毫无疑问是 [bsd.port.mk](#) 的源代码，以及相关的文件。

14.3. FreeBSD Ports 文件列表

如果同步了某个或某一些 ports，可以参考 [FreeBSD ports 文件列表](#)。关于 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 集群

FreeBSD 的一个最不为人所知的是，它有一个用于持续 Ports Collection 的集群，这个集群会同步所有主要的 OS 版本在一个 Tier-1 架构上的 package。可以在 [package](#) 和 [日志](#) 看到其结果。

一个 port 都会被同步，除非是 [IGNORE](#)。同步了 [BROKEN](#) 的 port 仍然会被同步，以了解是否某些依赖系的同步解决了其问题（是通过 port 的 Makefile 的 [TRYBROKEN](#) 参数来完成的）。

14.5. FreeBSD 的 Ports Distfile 扫描器

集群是一组用于同步所有 port 最新版本的机器，其上已同步了所有的 distfiles。然而，由于 Internet 在持续地进化，distfile 可能很快就消失了。[FreeBSD Ports distfile 扫描器](#) 同步一个 port 的所有下游站点，以期找出哪些文件是否依然存在。扫描者会规律性地同步一些公告，不会提高同步的速度，同时也避免了浪费那些像全部 distfile 的志愿者的时间。

14.6. FreeBSD 的 Ports 追踪系

一个非常方便的源，就是 [FreeBSD Ports 追踪系](#)（也被称作 [portsmon](#)）。这个系包含了一个管理若干信息来源的数据，并提供了一个可以通过 web 方式的界面。目前，它利用到了和 ports 有关的公告 (PR)、来自集群的日志，以及来自 Ports Collection 的文件所提供的信息。未来，它会它进行一系列的扩展，从而提供包括 distfile 普及，以及其它来源在内的更多信息。

要使用这个工具，可以从[看](#)于某一个 port 的全部资料的 [Port 的](#)开始。

本文撰写时，它是唯一一个能将 GNATS PR，同它的 port 名字映射起来的源。（提交 PR 的用户，有并不在 Synopsis（概要）中指明 port 的名字，尽管我希望他这样做）。因此，[portsmon](#) 在想要是否有人提交某个存在的 port 的 PR，以及它的是否出了；或在新建新的 port 之前想要一下是否已有人提交，就非常有用。