映像包管理系统手册页

文件号码 E25755
2012 年 3 月，E25755-03

ORACLE

120411@25097

目录

前言

本文档提供 Oracle Solaris 11 系统安装工具的手册页。

概述

以下内容包含对手册页各部分及其所引用信息的简要说明：

- 第 1 部分介绍了适用于操作系统的各种命令。
- 第 1M 部分介绍了主要用于系统维护和管理的各种命令。
- 第 5 部分包含其他文档，如字符集表。

下面是手册页的通用格式。每个手册的手册页部分通常遵循该顺序，但只包括需要的标题。例如，如果未报告任何已知问题，则不包括“已知问题”部分。通常，可以使用 `man` 命令来查看有关手册页的更多信息。

名称　本部分提供了记录的命令或函数的名称，后跟其用途的简要说明。

用法概要　本部分显示了命令或函数的语法。如果命令或文件不存在于标准路径中，则显示其全路径名。除非要求使用不同的参数顺序，否则选项和参数均按字母顺序排列，首先是单个字母的参数，接下来是带有参数的选项。

本部分使用以下特殊字符：

[ ]　括号。扩在这些括号中的选项或参数是可选的。如果省略括号，则必须指定参数。

. . .　省略号。可以为前一个参数提供多个值，也可以多次指定前一个参数，例如 "*filename* . . ."

|　分隔符。一次只能指定一个由该字符分隔的参数。

{ }　大括号。括在大括号内的选项和/或参数是相互依赖的，因此必须将大括号中的所有内容视为一个单元。

协议　本部分仅在第 3R 子部分出现，用于指示协议说明文件。

说明　本部分定义了服务的功能和行为。因此，它简明地介绍了命令执行哪些操作。它不讨论“选项”或引用“示例”。在“用法”下介绍了交互式命令、子命令、请求、宏和函数。

选项　　本部分列出了各命令选项及每个选项用途的简明摘要。逐个列出各个选项，并以它们在“用法概要”部分显示的顺序排列。在选项下讨论各个选项可能的参数，还提供缺省值（如果适用）。

操作数　　本部分列出了命令操作数，并介绍它们对命令操作的影响。

输出　　本部分介绍了命令所生成的输出（标准输出、标准错误或输出文件）。

返回值　　如果手册页记录返回值的函数，则本部分列出这些值并介绍返回这些值应满足的条件。如果函数只能返回常量值（例如 0 或 –1），则将在标记的段落中列出这些值。否则，会有单个段落介绍每个函数的返回值。声明为 void 的函数不返回值，因此不会在“返回值”中讨论这些函数。

错误　　对于故障，大多数函数将指出它们出现故障的原因的错误代码置于全局变量 `errno` 中。本部分按字母顺序列出了函数可以生成的所有错误代码，并介绍了导致每个错误的条件。如果多个条件可以导致同一错误，则在错误代码下以单独的段落介绍每个条件。

用法　　本部分列出了需要详细说明的特殊规则、功能和命令。此处列出的子部分用于说明内置功能：

命令
修饰符
变量
表达式
输入语法

示例　　本部分提供了用法的示例，或者如何使用命令或函数的示例。尽可能显示包括命令行条目和计算机响应的完整示例。每当给定一个示例，就会出现提示并显示为 `example%`，或者如果用户必须为超级用户，则显示为 `example#`。示例后面跟有说明、变量替换规则或返回值。大部分示例说明了“用法概要”、“说明”、“选项”和“用法”部分的概念。

环境变量　　本部分列出了命令或函数影响的所有环境变量，其后附加了关于影响的简要说明。

退出状态　　本部分列出了命令返回到调用程序或 shell 中的值以及导致返回这些值的条件。通常，返回零表示成功完成，返回非零值表示各种错误条件。

文件　　本部分列出了手册页引用的所有文件名称、相关文件以及命令创建或所需的文件。每个文件名称后面都具有描述性摘要或说明。

属性　　本部分通过定义属性类型及其相应的值列出了命令、实用程序和设备驱动程序的特征。有关更多信息，请参见 `attributes(5)` 手册页。

另请参见　　本部分列出了对其他手册页、内部文档和外部出版物的引用。

诊断　　本部分列出了诊断消息以及导致错误的条件的简要说明。

警告	本部分列出了有关特殊条件的警告，这些条件可能会严重影响您的工作状况。此部分不是诊断列表。
附注	本部分列出了不属于页面任何部分的其他信息。它采用对用户旁白提示的形式，包含用户特别关注的要点。此处不包含关键信息。
已知问题	本部分介绍了已知问题，并尽可能给出解决方法。

参 考 文 档

用户命令

引用名

packagemanager – 映像包管理系统的 GUI

用法概要

```
/usr/bin/packagemanager [options]

/usr/bin/packagemanager [-hiRu] [--help]
    [--info-install file] [--update-all]
    [--image-dir dir]

/usr/bin/packagemanager [file]
```

描述

`packagemanager` 是映像包管理系统软件 `pkg(5)` 的图形用户界面。

使用软件包管理器，您可以执行以下任务：

- 搜索、安装和删除软件包。
- 添加、删除和修改发布者。
- 创建、删除和管理引导环境。

如果指定了 *file* 操作数且其后缀为 `.p5i`，`packagemanager` 将在 Web 安装模式下启动，可添加一个或多个发布者并为每个发布者添加多个软件包。

选项

支持以下选项：

`-h` 或 `--help`
显示用法消息。

`-i` 或 `--info-install` *file*
允许指定 `.p5i` 文件以在 Web 安装模式下运行 `packagemanager`。*file* 必须具有后缀 `.p5i`。

`-R` 或 `--image-dir` *dir*
对根目录为 *dir* 的映像（而不是自动搜索到的映像）执行操作。

`-U` 或 `--update-all`
更新所有具有可用更新的已安装软件包。

注 – 如果 `package/pkg`、`package/pkg/package-manager` 或 `package/pkg/update-manager` 软件包需要更新，则 `packagemanager` 首先更新这些软件包，然后重新启动以完成其余所有更新。

操作数

file Web 安装文件。该文件必须具有后缀 `.p5i`。有关 Web 安装的更多信息，请参见软件包管理器联机帮助。

示例

示例 1 对当前映像执行操作

对当前映像调用 `packagemanager`。

```
$ packagemanager
```

示例 2 对指定映像执行操作

对存储在 /aux0/example_root 中的映像调用 packagemanager。

```
$ packagemanager -R /aux0/example_root
```

示例 3 在 Web 安装模式下调用

在 Web 安装模式下调用 packagemanager。

```
$ packagemanager ~/test.p5i
```

退出状态

将返回以下退出值：

0 一切正常工作。

1 出现错误。

2 指定的命令行选项无效。

文件

因为 pkg(5) 映像可位于任意一个较大的文件系统内，需要使用标记 $IMAGE_ROOT 来区分相对路径。对于典型系统安装，$IMAGE_ROOT 等效于 /。

$IMAGE_ROOT/var/pkg	完整或部分映像的元数据目录。
$IMAGE_ROOT/.org.opensolaris,pkg	用户映像的元数据目录。 在特定映像的元数据中，某些文件和目录包含修复和恢复期间有用的信息。标记 $IMAGE_META 用于指示包含元数据的顶层目录。 $IMAGE_META 通常是以上给出的两个路径之一。
$IMAGE_META/gui-cache	缓存元数据的位置，packagemanager 维护高速缓存元数据，以便加速程序启动及发布者之间的转换。

$IMAGE_META 目录分层结构中的其他路径是专用的，但可以进行更改。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg/package-manager
接口稳定性	Uncommitted（未确定）

另请参见

pm-updatemanager(1)、pkg(1)、pkg(5)

软件包管理器联机帮助

http://hub.opensolaris.org/bin/view/Project+pkg/

附注 需要使用足够的特权调用 `packagemanager` 以便对映像的文件和目录执行操作。

引用名

pkg – 映像包管理系统的检索客户端

用法概要

```
/usr/bin/pkg [options] command [cmd_options] [operands]

/usr/bin/pkg refresh [--full] [publisher ...]

/usr/bin/pkg install [-nvq] [-g path_or_uri ...] [--accept]
    [--licenses] [--no-be-activate] [--no-index] [--no-refresh]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    [--reject pkg_fmri_pattern ...] pkg_fmri_pattern ...

/usr/bin/pkg uninstall [-nvq] [--no-be-activate] [--no-index]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    pkg_fmri_pattern ...

/usr/bin/pkg update [-fnvq] [-g path_or_uri ...] [--accept]
    [--licenses] [--no-be-activate] [--no-index] [--no-refresh]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    [--reject pkg_fmri_pattern ...] [pkg_fmri_pattern ...]

/usr/bin/pkg list [-Hafnsuv] [-g path_or_uri ...]
    [--no-refresh] [pkg_fmri_pattern ...]

/usr/bin/pkg info [-lr] [-g path_or_uri ...] [--license]
    [pkg_fmri_pattern ...]

/usr/bin/pkg contents [-Hmr] [-a attribute=pattern ...]
    [-g path_or_uri ...] [-o attribute ...] [-s sort_key]
    [-t action_type ...] [pkg_fmri_pattern ...]

/usr/bin/pkg search [-HIaflpr] [-o attribute ...]
    [-s repo_uri] query

/usr/bin/pkg verify [-Hqv] [pkg_fmri_pattern ...]

/usr/bin/pkg fix [--accept] [--licenses] [pkg_fmri_pattern ...]

/usr/bin/pkg revert [-nv] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    (--tagged tag-name ... | path-to-file ...)

/usr/bin/pkg mediator [-aH] [-F format] [mediator ...]

usr/bin/pkg set-mediator [-nv] [-I implementation]
    [-V version] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    mediator ...

/usr/bin/pkg unset-mediator [-nvIV] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
```

```
    [--deny-new-be | --require-new-be] [--be-name name]
    mediator ...

/usr/bin/pkg variant [-H] [variant.variant_name ...]

/usr/bin/pkg change-variant [-nvq] [-g path_or_uri ...]
    [--accept] [--licenses] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    variant_name=value ...

/usr/bin/pkg facet [-H] [facet_name ...]

/usr/bin/pkg change-facet [-nvq] [-g path_or_uri ...]
    [--accept] [--licenses] [--no-be-activate]
    [--no-backup-be | --require-backup-be] [--backup-be-name name]
    [--deny-new-be | --require-new-be] [--be-name name]
    facet_name=[True|False|None] ...

/usr/bin/pkg avoid [pkg_fmri_pattern ...]

/usr/bin/pkg unavoid [pkg_fmri_pattern ...]

/usr/bin/pkg freeze [-n] [-c reason] [pkg_fmri_pattern] ...

/usr/bin/pkg unfreeze [-n] [pkg_name_pattern] ...

/usr/bin/pkg property [-H] [propname ...]

/usr/bin/pkg set-property propname propvalue

/usr/bin/pkg add-property-value propname propvalue

/usr/bin/pkg remove-property-value propname propvalue

/usr/bin/pkg unset-property propname ...

/usr/bin/pkg publisher [-HPn] [publisher ...]

/usr/bin/pkg set-publisher [-Ped] [-k ssl_key] [-c ssl_cert]
    [-g origin_to_add | --add-origin origin_to_add ...]
    [-G origin_to_remove | --remove-origin origin_to_remove ...]
    [-m mirror_to_add | --add-mirror mirror_to_add ...]
    [-M mirror_to_remove | --remove-mirror mirror_to_remove ...]
    [--enable] [--disable] [--no-refresh]
    [--reset-uuid] [--non-sticky] [--sticky]
    [--search-after publisher] [--search-before publisher]
    [--search-first]
    [--approve-ca-cert path_to_CA]
    [--revoke-ca-cert hash_of_CA_to_remove]
    [--unset-ca-cert hash_of_CA_to_remove]
    [--set-property name_of_property=value]
    [--add-property-value name_of_property=value_to_add]
    [--remove-property-value name_of_property=value_to_remove]
    [--unset-property name_of_property_to_delete]
    publisher
```

```
/usr/bin/pkg set-publisher -p repo_uri
    [-Ped] [-k ssl_key] [-c ssl_cert]
    [--non-sticky] [--sticky]
    [--search-after publisher] [--search-before publisher]
    [--search-first]
    [--approve-ca-cert path_to_CA]
    [--revoke-ca-cert hash_of_CA_to_remove]
    [--unset-ca-cert hash_of_CA_to_remove]
    [--set-property name_of_property=value]
    [--add-property-value name_of_property=value_to_add]
    [--remove-property-value name_of_property=value_to_remove]
    [--unset-property name_of_property_to_delete]
    [publisher]

/usr/bin/pkg unset-publisher publisher ...

/usr/bin/pkg history [-Hl] [-t [time | time-time],...]
    [-o column,...] [-n number]

/usr/bin/pkg purge-history

/usr/bin/pkg rebuild-index

/usr/bin/pkg update-format

/usr/bin/pkg version

/usr/bin/pkg help

/usr/bin/pkg image-create [-FPUfz] [--force]
    [--full | --partial | --user] [--zone]
    [-k ssl_key] [-c ssl_cert]
    [--no-refresh] [--variant variant_name=value ...]
    [-g path_or_uri | --origin path_or_uri ...]
    [-m uri | --mirror uri ...]
    [--set-property name_of_property=value]
    [--facet facet_name=(True|False) ...]
    [(-p | --publisher) [name=]repo_uri] dir
```

描述

pkg 是映像包管理系统的检索客户端。在有效配置下，可以调用 pkg 来为要安装的软件包创建位置（称为“映像”），然后将软件包安装到这些映像中。软件包由发布者发布。发布者可使其软件包在一个或多个系统信息库上可用，或者在软件包归档中可用。pkg 从发布者的系统信息库或软件包归档中检索软件包，然后将软件包安装到映像中。

发布者名称将个人、个人组或组织标识为一个或多个软件包的源。为避免发布者名称冲突并有帮于标识发布者，最佳做法是使用代表发布软件包实体的域名作为发布者名称。

系统信息库是客户端可在其中发布和检索软件包内容（该软件包包含的文件，例如程序和文档）和元数据（有关该软件包的信息，例如其名称和描述）的位置。例如，有一个名为 example.org 的发布者，其系统信息库位于 URI http://example.org/repository。

pkg 还可以卸载软件包、刷新发布者元数据（例如可用软件包的列表）、验证映像中的软件包安装，以及查询映像中的各个标记。也可以对 pkg(5) 系统信息库执行这些查询。

映像有三种类型：完整映像：能够提供完整的系统；部分映像：与完整映像（父映像）链接，但本身无法提供完整的系统；用户映像。

选项

支持以下选项：

-R *dir*
对根目录为 *dir* 的映像进行操作。如果未根据环境指定或确定目录，则缺省值为 /。有关更多信息，请参见“环境变量”部分。

--help 或 -?
显示用法消息。

子命令

支持以下子命令：

refresh [--full] [*publisher* ...]
对于每个指定的发布者，更新客户端可用软件包列表和发布者元数据。如果未指定任何发布者，则针对所有发布者执行此操作。

使用 --full 时，会强制完全检索所有发布者元数据（而不是尝试增量更新），并请求操作期间使用的所有代理忽略高速缓存的数据。此选项用于进行故障排除，正常情况下不应使用。

install [-nvq] [-g *path_or_uri* ...] [--accept] [--licenses] [--no-be-activate] [--no-index] [--no-refresh] [--no-backup-be | --require-backup-be] [--backup-be-name *name*] [--deny-new-be | --require-new-be] [--be-name *name*] [--reject *pkg_fmri_pattern* ...] *pkg_fmri_pattern* ...
安装与映像中安装的软件包所允许的 *pkg_fmri_pattern* 匹配的软件包，并将其更新到最新版本。要显式请求安装软件包的最新版本，请在 *pkg_fmri_pattern* 的版本部分使用 latest。例如，指定 vim@latest。

安装过程中，某些配置文件可能被重命名或替换。有关软件包系统如何确定要保留的文件，以及在软件包操作期间如何保留这些文件的更多信息，请参见 pkg(5) 手册页中的“文件操作”。

如果某个软件包在避免列表中，则安装该软件包会将它从该列表中删除。

使用 -g 时，会暂时将指定的软件包系统信息库或归档添加到从中检索软件包数据的映像中的源列表内。如果也可以通过映像中配置的发布者使用指定源中的软件包，客户端将仅从指定的源检索这些软件包的内容。确定要使用的软件包版本

时，优先选择映像中配置的、但在给定源中找不到的发布者。安装或更新后，在映像中找不到的发布者提供的任何软件包将添加到映像配置中，且添加的软件包没有源。可以多次指定此选项。

使用 -n 时，会试运行操作而不进行软件包更改。

使用 -q 时，会在执行请求的操作期间隐藏进度消息。

使用 -v 时，会在执行请求的操作期间发出详细进度消息，并显示详细的规划信息（例如更改侧面、中介者和变体）。可以多次指定此选项，以增加显示的规划信息量。

使用 --accept 时，表示同意并接受所更新或安装的软件包的许可证条款。如果不提供此选项，且任何软件包许可证都要求接受，则安装操作将失败。

使用 --licenses 时，会在此操作过程中显示已安装或更新的软件包的所有许可证。

使用 --no-backup-be 时，不会创建备份引导环境。

使用 --no-be-activate 时，如果创建了一个引导环境 (boot environment, BE)，则下次引导时，不会将该 BE 设置为活动 BE。有关更多信息，请参见 beadm(1M)。

使用 --no-index 时，不会在操作成功完成后更新搜索索引。

使用 --no-refresh 时，不会尝试联系系统信息库来让映像发布者检索最新的可用软件包列表和其他元数据。

使用 --backup-be-name 时，使用给定参数命名创建的备份引导环境。使用 --backup-be-name 意味着 --require-backup-be。另请参见 beadm(1M)。

使用 --be-name 时，会将新创建的引导环境重命名为给定的参数。使用 --be-name 意味着 --require-new-be。另请参见 beadm(1M)。

使用 --require-backup-be 时，如果不创建新的引导环境，则始终创建一个备份引导环境。如果不使用该选项，则根据映像策略创建备份引导环境。有关何时自动创建备份引导环境的说明，请参见下面“映像属性”中的 be-policy。

使用 --require-new-be 时，始终创建新的引导环境。如果不使用该选项，则根据映像策略创建引导环境。有关何时自动创建引导环境的说明，请参见下列“映像属性”中的 be-policy。该选项不能与 -require-backup-be 一起使用。

使用 --deny-new-be 时，不会创建新的引导环境。如果需要新的引导环境，则不执行此操作。

使用 --reject 时，会阻止安装其名称与给定模式匹配的软件包。如果已安装匹配的软件包，则在此操作过程会将其删除。作为组依赖性目标的被拒绝软件包将放置在避免列表中。

`uninstall [-nvq] [--no-be-activate ] [--no-index] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | - -require-new-be] [--be-name` *name*`]` *pkg_fmri_pattern* `...`

删除与 *pkg_fmri_pattern* 匹配的已安装软件包。

如果某个软件包是组依赖性的主体，则卸载该软件包会将它放置在避免列表中。请参见下面的 `avoid` 子命令。

对于其他所有选项，请参阅上面的 `install` 命令，以了解其用法及效果。

`update [-fnvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-index] [--no-refresh ] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`] [--reject` *pkg_fmri_pattern* `...] [`*pkg_fmri_pattern* `...]`

如果不使用任何参数，或者星号 (*) 是提供的模式之一，则将当前映像中安装的所有软件包更新到由安装的软件包和发布者配置强加于系统的约束所允许的最新版本。要显式请求安装软件包的最新版本，请在 *pkg_fmri_pattern* 的版本部分使用 `latest`。例如，指定 `vim@latest`。

如果提供了 *pkg_fmri_pattern*，`update` 将与 *pkg_fmri_pattern* 匹配的已安装软件包替换为由安装的软件包和发布者配置强加于系统的模式和约束所允许的最新版本。可以指定比已安装版本更高或更低的版本，以便对特定软件包执行就地降级或升级。不支持跨越软件包重命名边界或过时边界更新特定的软件包。

作为将要由 `update` 降级的软件包的一部分、自安装原始版本以来已更改的任何保留配置文件，将会使用扩展名 `.update` 进行重命名。有关软件包系统如何确定要保留的文件，以及在软件包升级期间如何保留这些文件的更多信息，请参见 `pkg`(5) 手册页中的“文件操作”。

使用 `-f` 选项时，不会在更新已安装的所有软件包时执行客户端最新状态检查。

对于其他所有选项，请参阅上面的 `install` 命令，以了解其用法及效果。

`list [-Hafnsuv] [-g` *path_or_uri* `...][--no-refresh] [`*pkg_fmri_pattern* `...]`

未提供参数时，显示当前映像中软件包的列表，包括版本和安装状态等信息。提供参数时，显示指定软件包的信息。缺省情况下，会排除不同体系结构或区域类型的软件包变体。输出通常包括三个列：

```
NAME (PUBLISHER)            VERSION          IFO
system/core-os              0.5.11-0.169     i--
x11/wm/fvwm (fvwm.org)      2.6.1-3          i--
```

第一列包含软件包的名称。如果安装（或者提供）该软件包的发布者在发布者搜索顺序上不是第一个，则该发布者名称将列在软件包名称的后面，并括在括号中。第二列包含软件包的发行版本和分支版本。有关发行版本和分支版本以及变量的信息，请参见 `pkg`(5) 手册页。

最后一列包含一组标志，用于显示软件包的状态：

- `I` 列中的 `i` 表明软件包已安装。
- `F` 列中的 `f` 表明软件包已冻结。
- `O` 列中的 `o` 表明软件包已过时。`O` 列中的 `r` 表明软件包已重命名（一种形式的过时）。

使用 `-H` 时，将忽略列表的标题。

使用 `-a` 时，会列出已安装的软件包，以及可用于安装的软件包的最新版本。如果软件包得到已安装 incorporation 和映像变体的允许，则认为这些软件包可用于安装。如果指定了一个或多个模式，则会列出与指定模式匹配的、得到任何已安装 incorporation 和映像变体允许的最新版本。如果不使用 `-a`，则仅列出已安装的软件包。

使用 `-f` 和 `-a` 时，会列出所有变体的所有软件包的所有版本，而不管 incorporation 约束或安装状态如何。使用这些选项时，要显式列出某个软件包的最新版本，请为 *pkg_fmri_pattern* 的版本部分使用 `latest`。例如，指定 `vim@latest`。

使用 `-g` 时，会使用指定的软件包系统信息库或归档作为操作的软件包数据源。可以多次指定此选项。如果未指定 `-n`，则可使用 `-g` 表示 `-a`。

使用 `-n` 时，会显示所有已知软件包的最新版本，而不管安装状态如何。

使用 `-s` 时，会显示单行短格式，用于提供软件包名称和摘要。此选项可与 `-a`、`-n`、`-u` 或 `-v` 一起使用。

使用 `-u` 时，仅列出其更高版本可用的软件包。此选项不能与 `-g` 一起使用。

使用 `-v` 时，会在第一列中显示完整的软件包 FMRI，包括发布者和完整版本（VERSION 列将会消失）。此选项可与 `-a`、`-n` 或 `-u` 一起使用。

使用 `--no-refresh` 时，不会尝试联系系统信息库来让映像发布者检索最新的可用软件包列表。

`info [-lr] [-g` *path_or_uri* `...][--license] [`*pkg_fmri_pattern* `...]`
以用户可读的格式显示有关软件包的信息。可以指定多个 FMRI 模式。如果未指定模式，则显示映像中所有已安装软件包的相关信息。

使用 `-g` 时，会使用指定的软件包系统信息库或归档作为操作的软件包数据源。可以多次指定此选项。可使用 `-g` 表示 `-r`。

使用 `-l` 时，仅显示已安装的软件包的信息。这是缺省值。

使用 `-r` 时，会根据可用的最新版本匹配软件包，并从映像的已配置发布者系统信息库中检索当前未安装的软件包的相关信息（如有必要）。使用此选项时，必须至少指定一个软件包。如果不指定 `-r`，则缺省情况下仅显示已安装的软件包。

使用 `--license` 时，会显示软件包的许可证文本。此选项可与 `-l` 或 `-r` 结合使用。

`contents [-Hmr] [-a` *attribute=pattern* `...][-g` *path_or_uri* `...][-o`*attribute*`,...] [-s`*sort_key*`] [-t` *action_type* `...][`*pkg_fmri_pattern* `...]`

显示软件包的内容（操作属性）。未指定选项或操作数时，显示安装在当前映像中的操作的 `path` 属性值，属性值按字母顺序排列。有关操作及其属性的信息，请参见 `pkg(5)` 手册页中的“操作”。另请参见以下伪属性名称列表。

使用 `-a` 时，会将输出限制为在选项参数中带有某个命名属性的那些操作，该属性的值与选项参数中的 (glob) 模式匹配（在属性名称后面添加一个等号）。如果指定了多个 `-a` 选项，则会显示与其中任一选项匹配的操作。

使用 `-g` 时，将显示指定软件包系统信息库或归档文件中可以安装在此映像中的软件包的信息。可以安装的软件包包含当前已安装的软件包以及其他满足此映像安装条件（例如变量和侧面限制）的软件包。可以多次指定此选项。可使用 `-g` 表示 `-r`。

使用 `-m` 时，将显示指定软件包中所有操作的所有属性，其中包括无法安装在此映像中的操作。

使用 `-o` 时，将按列出的第一个属性的值对列出的属性排序。可以多次指定 `-o` 选项；也可以通过使用逗号分隔属性名称，将多个属性指定为一个 `-o` 选项的参数。仅显示包含所请求属性的操作。

使用 `-r` 时，将显示此映像中配置的发布者系统信息库中可以安装在此映像中的最新可用软件包版本的信息。可以安装的软件包包含当前已安装的软件包以及其他满足此映像安装条件（例如变量和侧面限制）的软件包。使用此选项时，必须至少指定一个软件包。

使用 `-s` 时，会按指定的操作属性对操作进行排序。如果未提供此选项，则缺省设置是按照路径或者按照 `-o` 选项指定的第一个属性进行排序。可以多次指定 `-s` 选项。

使用 `-t` 时，仅列出指定类型的操作。可以在一个逗号分隔的列表中指定多个类型。可以多次指定此选项。

使用 `-H` 时，将忽略列表的标题。

使用 *pkg_fmri_pattern* 时，仅显示那些指定软件包的信息。

为方便起见，可以使用多个特殊的伪属性名称：

`action.hash`	操作的散列值（如果该操作承载了有效负荷）。
`action.key`	该操作的关键属性值。例如，对于 `file` 操作，关键属性是文件的路径。某些操作不具备关键属性。
`action.name`	操作的名称。例如，对于某个文件操作，该项为 `file`。
`action.raw`	匹配操作的所有属性。
`pkg.fmri`	包含操作的软件包的完整 FMRI，例如 `pkg://solaris/web/amp@0.5.11,5.11-0.169:20110705T153434Z`。

`pkg.name`	包含操作的软件包的名称，例如 `web/amp`。
`pkg.publisher`	包含操作的软件包的发布者，例如 `solaris`。
`pkg.shortfmri`	包含操作的软件包的短格式 FMRI，例如 `pkg://solaris/web/amp@0.5.11,5.11-0.169`。

`contents` 和 `search` 子命令是彼此相关的：两者都可在系统中查询软件包的内容。`contents` 子命令显示一个或多个已安装或可安装软件包中的操作，根据指定的选项过滤输出。`search` 子命令从另一个方向处理查询，显示包含用户提供的标记的所有软件包的名称。

每个子命令都能够表达另一个子命令也能表达的某些查询。应该谨慎选择所需的子命令，因为使用其中一个子命令来表达某个给定查询，可能比使用另一个子命令更为自然。

`search [-HIaflpr] [-o` *attribute*`,...][-s`*repo_uri*`]` *query*

搜索 *query* 的匹配项并显示结果。要为哪些标记建立索引是与操作相关的，不过可以包括内容散列和路径名称。有关操作及其属性的信息，请参见 `pkg(5)` 手册页中的“操作”。另请参见上面的 `pkg contents` 和下面的 `-o` 中的伪属性名称的列表。

缺省情况下，会将查询解释为要精确匹配的一系列条件。可将 `?` 和 `*` 字符用作 `glob(3C)` 式通配符，以更灵活地获取查询匹配项。

使用 `-H` 时，会省略标题。

使用 `-I` 时，会使用区分大小写的搜索。

缺省情况下，使用 `-a` 时，会执行搜索并显示有关匹配操作的信息。

缺省情况下，`search` 会删改低于当前安装版本的软件包中的结果，以及当前 incorporation 排除的软件包版本中的结果。使用 `-f` 可显示所有结果，而不管软件包版本如何。

使用 `-l` 时，会搜索映像的已安装软件包。

使用 `-o` 时，可以控制结果的各列。可以多次指定 `-o` 选项；也可以通过使用逗号分隔属性名称，将多个属性指定为一个 `-o` 选项的参数。除了上面概述的伪属性外，还为搜索结果定义了下列属性：

`search.match`	对应于与搜索查询匹配的字符串。
`search.match_type`	对应于包含与搜索查询匹配的字符串的属性。

使用 `-p` 时，会显示其某些操作与每个查询词匹配的软件包。使用此选项相当于在查询中的每个词两侧添加尖括号 (`<>`)。有关 `<>` 运算符的更多描述，请参见下文。

缺省情况下，使用 `-r` 时，会搜索对应于映像发布者的系统信息库。

使用 -s 时，会搜索位于给定 URI 的 pkg(5) 系统信息库。可以多次指定此选项。不支持软件包归档。

可以同时指定 -l 和 -r（或 -s），在此情况下，将会同时执行本地搜索和远程搜索。

除了支持简单的标记匹配和通配符搜索外，还支持更复杂的查询语言。可通过使用单引号或双引号（' 或 "）来搜索短语。请务必考虑所用的 shell，使 pkg 能够真正识别 ' 或 "。

支持使用 AND 和 OR 的布尔搜索。支持字段（或结构化）查询。这些命令的语法为 *pkg_name*:*action_type*:*key*:*token*。对缺少的字段使用隐式通配符。搜索 :basename:pkg 将会匹配包含关键字 basename 且与标记 pkg 匹配的所有软件包中的所有操作类型。pkg_name 和 token 字段中支持显式通配符。action_type 和 key 必须完全匹配。

要将操作转换为包含这些操作的软件包，请使用 <>。使用 -a 选项时，搜索 token 会导致返回与 token 匹配的操作的相关信息，而搜索 <token> 会导致返回一个软件包列表，这些软件包包含与 token 匹配的操作。

verify [-Hqv] [*pkg_fmri_pattern* ...]
验证当前映像中的软件包安装。如果相关发布者的当前签名策略不是 ignore，则会根据策略验证每个软件包的签名。有关如何应用签名策略的说明，请参见下面的“映像属性”中的 signature-policy。

使用 -H 时，会省略验证输出中的标题。

使用 -q 时，如果出现任何致命错误，则不会显示任何信息，但会返回失败消息。

使用 -v 时，会包括有关软件包的提示性消息。

fix [--accept] [--licenses] [*pkg_fmri_pattern* ...]
修复 pkg verify 报告的任何错误。已安装软件包内容的验证基于定制的内容分析，该分析可能会返回与其他程序不同的结果。

使用 --accept 时，表示同意并接受所更新或安装的软件包的许可证条款。如果不提供此选项，且任何软件包许可证都要求接受，则操作将失败。

使用 --licenses 时，会在此操作过程中显示要安装或更新的软件包的所有许可证。

revert [-nv] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name *name*] [--deny-new-be | --require-new-be] [--be-name *name*] (--tagged *tag-name* ... | *path-to-file* ...)
将文件恢复为其交付时的状态。可以使用特定值标记所有文件，也可以恢复单个文件。文件所有权和保护也会得到恢复。

注意 – 将某些可编辑文件恢复为其缺省值可能会使系统无法引导，或导致其他故障。

对于其他所有选项，请参阅上面的 `install` 命令，以了解其用法及效果。

`mediator [-aH] [-F` *format*`] [`*mediator* `...]`
显示所有中介者的当前选定版本和/或实现；如果提供了参数，则仅显示指定中介者的当前选定版本和/或实现。

使用 `-a` 时，会列出可针对当前安装的软件包设置的中介。

使用 `-F` 时，指定备用输出格式。当前，只有 `tsv`（Tab Separated Values，制表符分隔值）有效。

使用 `-H` 时，将忽略列表的标题。

`set-mediator [-nv] [-I` *implementation*`] [-V` *version*`] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | - -require-new-be] [--be-name` *name*`]` *mediator* `...`
设置当前映像中指定中介者的版本和/或实现。

使用 `-I` 时，会设置要使用的中介接口的实现。缺省情况下，如果未指定版本，则允许所有实现版本。要指定一个没有版本的实现，请附加 @ 符号。

使用 `-V` 时，会设置要使用的中介接口的版本。

如果指定的中介者版本和/或实现当前不可用，则会删除使用指定中介者的任何链接。

对于其他所有选项，请参阅上面的 `install` 命令，以了解其用法及效果。

`unset-mediator [-nvIV] [--no-be-activate ] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | --require-new-be] [--be-name` *name*`]` *mediator* `...`
将指定中介者的版本和/或实现恢复为系统缺省值。

使用 `-I` 时，仅恢复中介接口的实现。

使用 `-V` 时，仅恢复中介接口的版本。

对于其他所有选项，请参阅上面的 `install` 命令，以了解其用法及效果。

`variant [-H] [variant.`*variant_name* `...]`
未提供参数时，显示在此映像中设置的所有变量的当前值。提供参数时，显示在此映像中设置的每个指定变量 (`variant.`*variant_name*) 的值。

使用 `-H` 时，将忽略列表的标题。

有关变量的更多信息，请参见 `pkg`(5) 手册页中的“侧面和变量”。

`change-variant [-nvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | - -require-new-be] [--be-name` *name*`]` *variant_name*=*value* `...`

更改在当前映像中设置的指定变量的值。

有关选项的用法和效果，请参阅上面的 `install` 命令。

更改变量的值可能会导致删除、更新或安装某些软件包内容。更改变量值还可能会导致安装、更新或删除整个软件包，以满足新的映像配置。有关变量的更多信息，请参见 pkg(5) 手册页中的“侧面和变量”。

`facet [-H] [`*facet_name* `...]`

未提供参数时，显示使用 `pkg change-facet` 命令在此映像中显式设置的所有侧面的当前值。提供参数时，显示在此映像中设置的每个指定侧面 (*facet_name*) 的值。

使用 `-H` 时，将忽略列表的标题。

有关侧面的更多信息，请参见 pkg(5) 手册页中的“侧面和变量”。

`change-facet [-nvq] [-g` *path_or_uri* `...] [--accept] [--licenses] [--no-be-activate] [--no-backup-be | --require-backup-be] [--backup-be-name` *name*`] [--deny-new-be | - -require-new-be] [--be-name` *name*`]` *facet_name*`=[True|False|None] ...`

更改当前映像中设置的指定侧面的值。

可以将侧面设置为 `True` 或 `False`。将侧面设置为 `None` 时，系统会将缺省值 `True` 应用于该侧面；因此将安装任何由此侧面约束的操作。有关操作的信息，请参见 pkg(5) 手册页中的“操作”。

有关选项的用法和效果，请参阅上面的 `install` 命令。

更改侧面的值可能会导致删除、更新或安装某些软件包内容。更改侧面值还可能会导致安装、更新或删除整个软件包，以满足新的映像配置。有关侧面的更多信息，请参见 pkg(5) 手册页中的“侧面和变量”。

`avoid [`*pkg_fmri_pattern* `...]`

如果指定的软件包是某个组依赖性的目标，则会通过将匹配指定模式的软件包名称放置在避免列表中，来避免安装这些软件包。只能避免安装当前尚未安装的软件包。如果某个软件包当前是某个 group 依赖性的目标，则卸载该软件包会将它放置在免除列表中。

如果不提供任何参数，则会显示每个避免安装的软件包，以及与该软件包存在组依赖性的任何软件包。

如果需要满足 require 依赖性，则会安装免除列表中的软件包。如果删除了该依赖性，则会卸载相应的软件包。

`unavoid [`*pkg_fmri_pattern* `...]`
从避免列表中删除指定的软件包。使用此子命令无法删除避免列表中与某个已安装软件包的组依赖性相匹配的软件包。要从免除列表中删除与某个 group 依赖性相匹配的软件包，请安装该软件包。

如果不提供任何参数，则会显示已避免安装的软件包的列表。

`freeze [-n] [-c` *reason*`] [`*pkg_fmri_pattern*`] ...`
将指定的软件包冻结到指定的版本。如果未提供版本，则必须安装该软件包，然后将它冻结在该已安装版本。如果安装或更新冻结的软件包，则其最终版本必须与冻结时使用的版本匹配。例如，如果某个软件包在冻结时的版本为 1.2，则可以将它更新到 1.2.1、1.2.9、1.2.0.0.1，等等。但该软件包的最终版本不能为 1.3 或 1.1。*pkg_fmri_pattern* 中指定的发布者用于查找匹配的软件包。但是，在冻结过程中不会记录发布者信息。软件包只针对其版本（而不是发布者）进行冻结。冻结一个已经冻结的软件包会将冻结版本替换为新指定的版本。

如果未提供任何软件包，则会显示有关当前已冻结软件包的信息：软件包名称、版本、冻结时间以及任何相关原因。

冻结某个软件包不会阻止删除该软件包。删除软件包时不会显示警告。

使用 `-c` 时，会记录冻结软件包的 *reason*。当安装或更新因冻结而不能继续进行时，将会显示该原因。

使用 `-n` 时，会试运行操作，并显示要冻结的软件包的列表，但不冻结任何软件包。

`unfreeze [-n] [`*pkg_name_pattern* `] ...`
从指定的软件包中删除冻结操作施加的约束。将会忽略提供的版本。

使用 `-n` 时，会试运行解冻，并显示要解冻的软件包的列表，但不解冻任何软件包。

`property [-H] [`*propname* `...]`
显示映像属性信息。如果不提供参数，则会显示所有映像属性的名称和值。如果请求了特定的属性名称列表，则显示这些属性的名称和值。有关映像属性的说明，请参见下面的“映像属性”。

使用 `-H` 时，将忽略列表的标题。

`set-property` *propname propvalue*
更新现有映像属性或添加新的映像属性。

`add-property-value` *propname propvalue*
向现有映像属性添加值，或添加新的映像属性。

`remove-property-value` *propname propvalue*
从现有映像属性中删除值。

`unset-property` *propname* `...`
删除一个或多个现有映像属性。

`publisher [-HPn] [`*publisher* `...]`
显示发布者信息。如果不提供参数，则会按搜索优先顺序显示所有发布者、其源 URI 和镜像的列表。如果请求了特定的发布者，则显示这些发布者的详细配置。

使用 `-H` 时，将忽略列表的标题。

使用 `-P` 时，仅按发布者搜索顺序显示第一个发布者。

使用 `-n` 时，仅显示已启用的发布者。

`set-publisher [-Ped] [-k` *ssl_key*`] [-c` *ssl_cert*`] [-g` *origin_to_add* `| --add-origin` *origin_to_add* `...] [-G` *origin_to_remove* `| --remove-origin` *origin_to_remove* `...] [-m` *mirror_to_add* `| - -add-mirror` *mirror_to_add* `...] [-M` *mirror_to_remove* `| --remove-mirror` *mirror_to_remove* `...] [--enable ] [--disable] [--no-refresh] [--reset-uuid ] [--non-sticky] [--sticky] [--search-after` *publisher*`] [--search-before` *publisher*`] [--search-first] [--approve-ca-cert` *path_to_CA*`] [--revoke-ca-cert` *hash_of_CA_to_remove*`] [--unset-ca-cert` *hash_of_CA_to_remove*`] [--set-property` *name_of_property*`=`*value*`] [--add-property-value` *name_of_property*`=`*value_to_add*`] [--remove-property-value` *name_of_property*`=` *value_to_remove*`] [--unset-property` *name_of_property_to_delete* `] `*publisher*
更新现有发布者，或添加软件包发布者。如果未指定任何影响搜索顺序的选项，则会将新发布者附加到搜索顺序，因此，会最后搜索这些新发布者。

使用 `-P` 或 `--search-first` 时，会将指定的发布者设置在搜索顺序的首位。安装新软件包时，将首先搜索此发布者。对已安装软件包的更新将来自最初提供该软件包的同一发布者，前提是该发布者保持粘滞。将 `-P` 或 `--search-first` 与 `-p` 一起使用时，仅将添加的发布者放置在搜索顺序的首位。

使用 `--non-sticky` 时，会指定级别比此发布者更高的发布者可以为最初通过此发布者安装的软件包提供更新。

使用 `--sticky` 时，会指定对通过此发布者安装的软件包的更新必须也来自此发布者。这是缺省行为。

使用 `--search-before` 时，会更改发布者搜索顺序，以便在指定的发布者之前搜索被修改的发布者。与 `-p` 一起使用时，`--search-before` 仅适用于添加的发布者。

使用 `--search-after` 时，会更改发布者搜索顺序，以便在指定的发布者之后搜索被修改的发布者。与 `-p` 一起使用时，`--search-after` 仅适用于添加的发布者。

使用 `--approve-ca-cert` 时，会将给定的证书添加为可信的 CA 证书。`pkg publisher` 命令的详细输出中列出了用户批准的 CA 证书的 PEM 表示法散列。

使用 `--revoke-ca-cert` 时，会将具有其 PEM 表示法的给定散列的证书视为已撤销。`pkg publisher` 命令的详细输出中列出了用户撤销的 CA 证书的散列。

使用 --unset-ca-cert 时，会从已批准证书列表和已撤销证书列表中删除具有给定散列的证书。

使用 --set-property 时，会更新现有的发布者属性，或添加新的发布者属性。

使用 --add-property-value 时，会向现有的发布者属性添加值，或添加新的发布者属性。

使用 --remove-property-value 时，会从现有的发布者属性中删除某个值。

使用 --unset-property 时，会删除某个现有的发布者属性。

使用 -c 和 -k 时，会分别指定客户端 SSL 证书和密钥。

使用 -g (--add-origin) 时，会将指定的 URI 或路径添加为给定发布者的源。这应该是软件包系统信息库或归档的位置。

使用 -G (--remove-origin) 时，会从给定发布者的源列表中删除 URI 或路径。可以使用特殊值 * 来删除所有源。

使用 --no-refresh 时，不会尝试联系系统信息库来让映像发布者检索最新的可用软件包列表和其他元数据。

使用 --reset-uuid 时，会选择一个新的唯一标识符，用于向其发布者标识此映像。

使用 -m (--add-mirror) 时，会将 URI 添加为给定发布者的镜像。

使用 -M (--remove-mirror) 时，会从给定发布者的镜像列表中删除 URI。可以使用特殊值 * 来删除所有镜像。

使用 -p 时，会从指定的系统信息库 URI 中检索发布者配置信息。如果指定了发布者，则仅添加或更新匹配的发布者。如果未指定发布者，则根据需要添加或更新所有发布者。此选项不能与 -g、--add-origin、--G、--remove-origin、-m、--add-mirror、-M、--remove-mirror、--disable、--enable、--no-refresh 或 --reset-uuid 选项结合使用。

使用 -e (--enable) 时，会启用发布者。使用 -d (--disable) 时，会禁用发布者。填充软件包列表时，或者执行某些软件包操作（安装、卸载和更新）时，将不使用被禁用的发布者。但是，仍可以设置和查看被禁用的发布者的属性。如果只存在一个发布者，则不能将其禁用。

`/usr/bin/pkg set-publisher -p` *repo_uri* `[-Ped] [-k` *ssl_key*`] [-c` *ssl_cert*`] [--non-sticky] [--sticky] [--search-after` *publisher*`] [--search-before` *publisher*`] [--search-first] [--approve-ca-cert` *path_to_CA*`] [--revoke-ca-cert` *hash_of_CA_to_remove*`] [--unset-ca-cert` *hash_of_CA_to_remove*`] [--set-property` *name_of_property*`=` *value*`] [--add-property-value` *name_of_property* `=`*value_to_add*`] [--remove-property-value` *name_of_property*`=`*value_to_remove* `] [--unset-property` *name_of_property_to_delete* `] [`*publisher*`]`

使用 `-p` 时，会从指定的系统信息库 URI 中检索发布者配置信息。如果指定了发布者，则仅添加或更新匹配的发布者。如果未指定发布者，则根据需要添加或更新所有发布者。有关可以与 `-p` 选项结合使用的其他选项的说明，请参见上面介绍的 `pkg set-publisher`。`-p` 选项不能与以下选项结合使用：`-g`、`--add-origin`、`-G`、`--remove-origin`、`-m`、`--add-mirror`、`-M`、`--remove-mirror`、`--disable`、`--enable`、`--no-refresh` 或 `--reset-uuid`。

`unset-publisher` *publisher* `...`

删除与一个或多个给定发布者相关联的配置。

`history [-Hl] [-t [` *time* | *time-time*`],...][-o`*column*`,...][-n` *number*`]`

显示适用映像的命令历史记录。

使用 `-H` 时，将忽略列表的标题。

使用 `-t` 时，会显示时间戳（格式为 `%Y-%m-%dT%H:%M:%S`）逗号分隔列表的日志记录（请参见 `strftime`(3C)）。要指定时间范围，请在开始和完成时间戳之间使用连字符 (-)。可以将关键字 `now` 用作当前时间的别名。如果指定的时间戳包含重复的时间戳或重叠的日期范围，则每个重复的历史记录事件仅显示一次。

使用 `-l` 时，会以长格式显示日志记录，也就是说，除了显示标准格式的内容外，还包括命令的结果、命令完成时间、所用客户端的版本和名称、执行操作的用户名，以及执行命令时遇到的任何错误。

使用 `-n` 时，仅显示指定数目的最近条目。

使用 `-o` 时，会使用指定的列名逗号分隔列表显示输出。有效的列名为：

`be`	在其上启动此操作的引导环境的名称。
`be_uuid`	在其上启动此操作的引导环境的 `uuid`。
`client`	客户端的名称。
`client_ver`	客户端的版本。
`command`	用于此操作的命令行。
`finish`	完成此操作时的时间。
`id`	启动此操作的用户 ID。
`new_be`	此操作创建的新引导环境。

new_be_uuid	此操作创建的新引导环境的 uuid。
operation	操作的名称。
outcome	此操作的结果摘要。
reason	有关此操作的结果的其他信息。
snapshot	执行此操作期间创建的快照。仅当成功完成操作后未自动删除快照时，才会记录此信息。
start	启动此操作时的时间。
time	执行此操作所用的总时间。对于用时不到一秒的操作，将显示 0:00:00。
user	启动此操作的用户名。

如果指定了 command 或 reason 列，则它们必须是 -o 列表中的最后一项，这样才能让输出字段彼此分隔。这两列不能显示在同一个 history 命令中。

如果系统上不再存在该引导环境，则在 be 或 new_be 值的后面会显示一个星号 (*)。

可通过使用 be_uuid 或 new_be_uuid 字段查找当前引导环境名称，来获取 be 和 new_be 的值。如果某个引导环境后来已重命名，随后又被删除，则显示的 be 和 new_be 值是执行 pkg 操作时记录的值。

purge-history
删除所有现有历史记录信息。

rebuild-index
重建由 pkg search 使用的索引。这是一项恢复操作，不适用于一般用途。

update-format
将映像格式更新到当前版本。完成此操作后，该映像无法再在早期版本的 pkg(5) 系统上使用。

version
显示一个用于标识 pkg(1) 版本的唯一字符串。不保证在不同版本中，此字符串在任何方面都具有类似性。

image-create [-FPUfz] [--force] [--full | --partial | --user] [--zone] [-k *ssl_key*] [-c *ssl_cert*] [--no-refresh] [--variant *variant_name*=*value* ...] [-g *path_or_uri* | --origin *path_or_uri* ...] [-m *uri* | - -mirror *uri* ...] [--set-property *name_of_property* =*value*][--facet *facet_name*=(True|False) ...] [(-p | --publisher) [*name*=] *repo_uri*] *dir*
在 *dir* 指定的位置，创建一个适合于软件包操作的映像。缺省的映像类型为“用户”，即 -U (--user) 选项指定的类型。可以将映像类型设置为完整映像（--F 或 --full），或者设置为与完整映像（包括给定的 *dir* 路径）链接的部分映像（-P 或 --partial）。可以使用 -g 或 --origin 指定其他源。可以使用 --m 或 --mirror 指定其他镜像。

必须使用 -p 或 --publisher 选项提供软件包系统信息库 URI。如果还提供了某个发布者名称，则创建映像时仅添加该发布者。如果未提供发布者名称，则会将指定的系统信息库已知的所有发布者添加到映像。完成初始创建操作后，将会尝试检索与此发布者关联的目录。

对于使用客户端 SSL 验证的发布者，可以通过 -c 和 -k 选项注册客户端密钥和客户端证书。此密钥和证书用于映像创建期间添加的所有发布者。

如果要在非全局区域上下文中运行映像，则可以使用 -z (--zone) 选项设置相应的变体。

使用 -f (--force) 时，会基于现有映像强制创建一个映像。请慎用此选项。

使用 --no-refresh 时，不会尝试联系系统信息库来让映像发布者检索最新的可用软件包列表和其他元数据。

使用 --variant 时，会将指定的变体设置为指示值。有关变量的更多信息，请参见 pkg(5) 手册页中的“侧面和变量”。

使用 --facet 时，会将指定的侧面设置为指示值。有关侧面的更多信息，请参见 pkg(5) 手册页中的“侧面和变量”。

使用 --set-property 时，会将指定的映像属性设置为指示值。有关映像属性的说明，请参见下面的“映像属性”。

映像属性

以下属性可定义映像的特征。这些属性存储有关映像的用途、内容和行为的信息。要查看映像中这些属性的当前值，请使用 pkg property 命令。要修改这些属性的值，请使用 pkg set-property 和 pkg unset-property 命令。

be-policy
（字符串）指定在打包操作期间何时创建引导环境。允许使用以下值：

default　应用缺省 BE 创建策略 create-backup。

always-new　所有软件包操作均需要重新引导：在下次引导时在设为活动状态的新 BE 中执行这些操作。除非显式请求，否则不会创建备份 BE。

该策略最为安全，但是它比大多数站点的需要更为严格，因为在不重新引导的情况下无法添加任何软件包。

create-backup
对于需要重新引导的软件包操作，在下一次引导时，将创建一个新的 BE 并将其设置为处于活动状态。如果修改了软件包或安装了可能影响内核的内容，并且该操作影响实时 BE，将创建备份 BE，但不会将其设置为活动状态。也可以显式请求创建备份 BE。

仅当新安装的软件导致系统不稳定时（有可能发生，但比较少见），该策略才可能存在风险。

`when-required`
对于需要重新引导的软件包操作，在下一次引导时，将创建一个新的 BE 并将其设置为处于活动状态。除非显式请求，否则不会创建备份 BE。

该策略的风险性最高，因为如果对实时 BE 的打包更改使得以后不可再进行更改，则可能没有可回退到的最近 BE。

`ca-path`
（字符串）一个路径名称，指向为执行 SSL 操作而将 CA 证书保存到的目录。此目录的格式特定于底层 SSL 实现。要对可信 CA 证书使用替代位置，请将此值更改为指向另一个目录。有关 CA 目录的要求，请参见 `SSL_CTX_load_verify_locations`(3openssl) 的 `CApath` 部分。

缺省值：`/etc/openssl/certs`

`check-certificate-revocation`
（布尔型）如果此属性设置为 `True`，则软件包客户机将尝试访问用于签名验证的证书中的任何 CRL 分发点，以确定证书自颁发以来是否已被撤销。

缺省值：`False`

`flush-content-cache-on-success`
（布尔型）如果此属性设置为 `True`，则完成安装或更新操作后，软件包客户机将删除其内容高速缓存中的文件。对于更新操作，仅从源 BE 中删除内容。如果随后目标 BE 中发生了打包操作，并且此选项未发生更改，则软件包客户机将刷新其内容高速缓存。

在磁盘空间有限的系统上，可以使用此属性使内容高速缓存保持为较小的大小。此属性可能会导致花费更长的时间来完成操作。

缺省值：`True`

`mirror-discovery`
（布尔型）此属性通知客户端使用 mDNS 和 DNS-SD 发现本地链路内容镜像。如果此属性设置为 `True`，则客户机尝试从其动态发现的镜像中下载软件包内容。要运行一个通过 mDNS 通告其内容的镜像，请参见 `pkg.depotd`(1M)。

缺省值：`False`

`send-uuid`
（布尔型）执行网络操作时发送映像的通用唯一标识符 (Universally Unique Identifier, UUID)。尽管用户可以禁用此选项，但是某些网络系统信息库可能会拒绝与不提供 UUID 的客户机通信。

缺省值：`True`

`signature-policy`
（字符串）确定在映像中安装、更新、修改或验证软件包时，要对清单执行哪些检查。应用于软件包的最终策略取决于映像策略和发布者策略的组合。该策略组合的

严格程度至少相当于这两个策略单独执行时较严格的那一个。缺省情况下，软件包客户机不检查证书是否已撤销。要启用这些检查（可能需要客户机访问外部 Web 站点），请将 check-certificate-revocation 映像属性设置为 True。允许使用以下值：

ignore	忽略所有清单的签名。
verify	验证所有具有签名的清单的签名是否有效，但不要求签名所有安装的软件包。这是缺省值。
require-signatures	要求所有新安装的软件包至少具有一个有效签名。如果安装的软件包不具备有效签名，pkg fix 和 pkg verify 命令也会发出警告。
require-names	与 require-signatures 遵循相同的要求，但还要求 signature-required-names 属性中列出的字符串显示为用于验证签名信任链的证书的通用名称。

signature-required-names
（字符串列表）在验证软件包签名时必须视为证书通用名称的名称列表。

trust-anchor-directory
（字符串）包含映像信任锚的目录的路径名称。此路径是映像的相对路径。缺省值为 ignore。

use-system-repo
（布尔型）此属性指示映像是否应使用系统信息库作为映像和发布者配置的源，以及作为与提供的发布者通信的代理。缺省值为 False。有关系统信息库的信息，请参见 pkg.sysrepo(1M)。

发布者属性

以下属性定义了特定发布者的签名策略。具有相同名称的映像属性定义了该映像的签名策略。要查看特定发布者的这些属性的当前值，请使用 pkg publisher *publisher_name* 命令。要修改发布者的这些签名策略属性的值，请使用 pkg set-publisher 命令的 --set-property 和 --unset-property 选项。

signature-policy
（字符串）此属性的作用与同名映像属性的功能相同，不过它仅适用于来自特定发布者的软件包。

signature-required-names
（字符串列表）此属性的作用与同名映像属性的功能相同，不过它仅适用于来自特定发布者的软件包。

示例

示例 1 在配置了发布者的情况下创建映像

使用 /aux0/example_root 中存储的发布者 example.com 创建一个新的完整映像。

```
$ pkg image-create -F -p example.com=http://pkg.example.com:10000 \
/aux0/example_root
```

示例 2 创建一个映像并指定附加源和镜像

使用发布者 `example.com` 创建一个新的完整映像。该映像还有一个附加镜像、两个附加源，并存储在 `/aux0/example_root` 中。

```
$ pkg image-create -F -p example.com=http://pkg.example.com:10000 \
-g http://alternate1.example.com:10000/ \
-g http://alternate2.example.com:10000/ \
-m http://mirror.example.com:10000/ \
/aux0/example_root
```

示例 3 在未配置发布者的情况下创建映像

在未配置发布者的情况下，在 `/aux0/example_root` 中创建一个新的完整映像。

```
$ pkg image-create -F /aux0/example_root
```

示例 4 安装软件包

在当前映像中安装最新版本的 `widget` 软件包。

```
$ pkg install application/widget
```

示例 5 列出软件包的指定内容

列出 `system/file-system/zfs` 软件包的内容。显示操作名称、文件模式（如果已定义）、大小（如果已定义）、路径和目标（如果为链接）。将操作限制为类型 `dir`、`file`、`link` 和 `hardlink`，因为指定可用于所有操作的 `action.name` 属性将显示一个列出了所有操作的行，而此处并不需要显示它。

```
$ pkg contents -t dir,file,link,hardlink \
-o action.name,mode,pkg.size,path,target system/file-system/zfs
ACTION.NAME MODE PKG.SIZE PATH                 TARGET
dir         0755          etc
dir         0755          etc/fs
dir         0755          etc/fs/zfs
link                      etc/fs/zfs/mount     ../../../usr/sbin/zfs
link                      etc/fs/zfs/umount    ../../../usr/sbin/zfs
dir         0755          etc/zfs
dir         0755          kernel
dir         0755          kernel/drv
dir         0755          kernel/drv/amd64
file        0755  1706744 kernel/drv/amd64/zfs
file        0644      980 kernel/drv/zfs.conf
dir         0755          kernel/fs
dir         0755          kernel/fs/amd64
hardlink                  kernel/fs/amd64/zfs  ../../../kernel/drv/amd64/zfs
...
```

示例 6 列出两个软件包的指定内容

列出 web/browser/firefox 和 mail/thunderbird 的内容，将显示的内容仅限于软件包名称，以及其 path 属性以 .desktop 或 .png 结尾的操作的路径属性。

```
$ pkg contents -o pkg.name,path -a path=\*.desktop \
-a path=\*.png web/browser/firefox mail/thunderbird
PKG.NAME            PATH
web/browser/firefox usr/share/applications/firefox.desktop
mail/thunderbird    usr/share/applications/thunderbird.desktop
web/browser/firefox usr/share/pixmaps/firefox-icon.png
mail/thunderbird    usr/share/pixmaps/thunderbird-icon.png
...
```

示例 7 搜索软件包

在软件包数据库中搜索标记 bge。

```
$ pkg search bge
INDEX       ACTION VALUE                     PACKAGE
driver_name driver bge                       pkg:/driver/network/bge@0.5.11-0.169
basename    file   kernel/drv/sparcv9/bge    pkg:/driver/network/bge@0.5.11-0.169
basename    file   kernel/drv/amd64/bge      pkg:/driver/network/bge@0.5.11-0.169
pkg.fmri    set    solaris/driver/network/bge pkg:/driver/network/bge@0.5.11-0.169
```

该标记在软件包 driver/network/bge 中，既用作代表 /kernel/drv/ *arch*/bge 的文件操作的根基名称，又用作驱动程序名称。

示例 8 搜索依赖于指定软件包的软件包

搜索依赖于 package/pkg 的已安装软件包。

```
$ pkg search -l 'depend::package/pkg'
INDEX       ACTION VALUE                    PACKAGE
incorporate depend package/pkg@0.5.11-0.169 pkg:/consolidation/ips/ips-incorporation@0.5.11-0.169
require     depend package/pkg@0.5.11-0.169 pkg:/system/install@0.5.11-0.169
require     depend package/pkg@0.5.11-0.169 pkg:/package/pkg/system-repository@0.5.11-0.169
```

示例 9 搜索依赖性

在安装的软件包中搜索所有 incorporate 依赖性。

```
$ pkg search -l 'depend:incorporate:'
INDEX       ACTION VALUE                           PACKAGE
incorporate depend pkg:/BRCMbnx@0.5.11,5.11-0.133  pkg:/consolidation/osnet/osnet-incorporation@0.5
incorporate depend pkg:/BRCMbnxe@0.5.11,5.11-0.133 pkg:/consolidation/osnet/osnet-incorporation@0.5
...
```

示例 10 添加发布者

添加新的发布者 example.com，该发布者的系统信息库位于 http://www.example.com/repo。

示例 10 添加发布者 （续）

```
$ pkg set-publisher -g http://www.example.com/repo example.com
```

示例 11 添加具有密钥和证书的发布者

添加新的发布者 example.com，该发布者的安全系统信息库位于 https://secure.example.com/repo，其密钥和证书存储在目录 /root/creds 中。

```
$ pkg set-publisher -k /root/creds/example.key \
-c /root/creds/example.cert -g https://secure.example.com/repo \
example.com
```

示例 12 添加并自动配置发布者

使用自动配置功能添加一个新的发布者，该发布者的系统信息库位于 /export/repo。

```
$ pkg set-publisher -p /export/repo
```

示例 13 添加并手动配置发布者

使用手动配置功能添加新的发布者 example.com，该发布者的系统信息库位于 /export/repo/example.com。

```
$ pkg set-publisher -g /export/repo example.com
```

示例 14 验证所有签名的软件包

配置一个映像以验证所有签名的软件包。

```
$ pkg set-property signature-policy verify
```

示例 15 要求签名所有软件包

配置一个映像，以要求签名所有软件包，并要求字符串 example.com 显示为信任链中某一个证书的通用名称。

```
$ pkg set-property signature-policy require-names example.com
```

示例 16 要求对来自指定发布者的所有软件包进行签名

配置一个映像，以便必须对通过发布者 example.com 安装的所有软件包进行签名。

```
$ pkg set-publisher --set-property signature-policy=require-signatures \
example.com
```

示例 17 要求信任链中存在指定的字符串

将字符串 foo 添加到映像的通用名称列表，这些通用名称必须显示在签名的信任链中才能视为有效。

```
$ pkg add-property-value signature-require-names foo
```

示例 18 从指定发布者的信任链中删除某个字符串

从通用名称列表中删除字符串 `foo`，必须显示这些通用名称才能验证发布者 `example.com` 的签名。

```
$ pkg set-publisher --remove-property-value signature-require-names=foo \
example.com
```

示例 19 添加可信 CA 证书

添加 `/tmp/example_file.pem` 中存储的证书，作为发布者 `example.com` 的可信 CA 证书。

```
$ pkg set-publisher --approve-ca-cert /tmp/example_file.pem \
example.com
```

示例 20 撤销证书

撤销发布者 `example.com` 的、包含散列 `a12345` 的证书，防止该证书验证来自 `example.com` 的软件包的任何签名。

```
$ pkg set-publisher --revoke-ca-cert a12345 example.com
```

示例 21 忘记针对某个证书执行的操作

使 `pkg` 忘记用户曾经添加或撤销了证书 `a12345`。

```
$ pkg set-publisher --unset-ca-cert a12345 example.com
```

示例 22 将软件包降级

将安装的软件包 `foo@1.1` 降级到更低的版本。

```
$ pkg update foo@1.0
```

示例 23 切换发生冲突的软件包安装

当两个软件包发生冲突时，切换所安装的软件包。假定软件包 A 依赖于软件包 B 或软件包 C，而 B 和 C 是互斥的。如果安装了 A 和 B，则使用以下命令即可改为使用 C 而不是 B，且无需卸载 A：

```
$ pkg install --reject B C
```

示例 24 列出软件包归档中的软件包

列出某个软件包归档中所有软件包的所有版本。

```
$ pkg list -f -g /my/archive.p5p
```

示例 25 列出软件包系统信息库中的软件包

列出某个系统信息库中所有软件包的所有版本。

```
$ pkg list -f -g http://example.com:10000
```

示例 26 显示有关软件包归档中某个软件包的信息

显示软件包归档中某个软件包的最新版本的软件包信息。该软件包当前不一定已安装。

```
$ pkg info -g /my/archive.p5p pkg_name
```

示例 27 显示软件包归档中某个软件包的内容

显示软件包归档中某个软件包的内容。该软件包当前未安装。

```
$ pkg contents -g /my/archive.p5p pkg_name
```

示例 28 删除发布者的所有源和镜像

删除某个发布者的所有源和镜像，并添加新的源。

```
$ pkg set-publisher -G '*' -M '*' -g http://example.com:10000 \
example.com
```

环境变量

`PKG_IMAGE`
用于软件包操作的映像所在的目录。如果指定了 `-R`，则会忽略此属性。

`PKG_CLIENT_CONNECT_TIMEOUT`
传输操作期间尝试建立连接时等待的秒数（针对每次尝试），达到此秒数后，客户端会异常中止操作。值 0 表示无限期等待。

缺省值为：60

`PKG_CLIENT_LOWSPEED_TIMEOUT`
传输操作期间低于 `lowspeed` 限制（1024 字节/秒）的秒数，达到此秒数后，客户端会异常中止操作。值 0 表示不中止运行。

缺省值为：30

`PKG_CLIENT_MAX_CONSECUTIVE_ERROR`
客户端异常中止操作之前发生瞬态传输错误的最大次数。值 0 表示不中止运行。

缺省值为：4

`PKG_CLIENT_MAX_REDIRECT`
在传输操作期间，异常中止某个连接之前允许的最大 HTTP 或 HTTPS 重定向次数。值 0 表示不中止运行。

缺省值为：5

`PKG_CLIENT_MAX_TIMEOUT`
客户端异常中止操作之前每台主机上的最大传输尝试次数。值 0 表示不中止运行。

缺省值为：4

`http_proxy`、`https_proxy`
HTTP 或 HTTPS 代理服务器。

退出状态

将返回以下退出值：

0 命令成功。

1 出现错误。

2 指定的命令行选项无效。

3 请求了多项操作，但只有一部分操作成功。

4 未进行更改－没有要执行的操作。

5 无法对实时映像执行请求的操作。

6 无法完成请求的操作，因为尚未接受所安装或更新的软件包的许可证。

7 该映像当前已被另一个进程使用，无法修改。

99 发生了意外的异常。

文件

可将 pkg(5) 映像放置在较大文件系统中的任意位置。在以下文件描述中，标记 $IMAGE_ROOT 用于区分相对路径。对于典型的系统安装，$IMAGE_ROOT 等效于 /。

$IMAGE_ROOT/var/pkg 完整或部分映像的元数据目录。

$IMAGE_ROOT/.org.opensolaris,pkg 用户映像的元数据目录。

在特定映像的元数据中，某些文件和目录可能包含修复和恢复期间有用的信息。标记 $IMAGE_META 引用元数据所在的顶层目录。$IMAGE_META 通常是上述两个路径之一。

$IMAGE_META/lost+found 在软件包操作期间移动的有冲突目录和文件的位置。某个已删除目录的未打包内容的位置。

$IMAGE_META/publisher 为每个发布者包含一个目录。每个目录存储特定于发布者的元数据。

$IMAGE_META 目录分层结构中的其他路径是专用的，可能会进行更改。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkgsend(1)、pkg.depotd(1m)、glob(3C)、pkg(5)、beadm(1M)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pkgdepend – 映像包管理系统依赖项分析器

用法概要

```
/usr/bin/pkgdepend [options] command [cmd_options] [operands]

/usr/bin/pkgdepend generate [-IMm] -d dir [-d dir]
    [-D name=value] [-k path] manifest_file

/usr/bin/pkgdepend resolve [-moSv] [-d output_dir]
    [-s suffix] manifest_file ...
```

描述

`pkgdepend` 用于生成并解析软件包的依赖项。软件包可能依赖于其他软件包中的文件。`pkgdepend` 通常用于两种传递：文件依赖项生成和文件到软件包的解析。

`generate` 子命令将检查软件包的内容，并确定该软件包所需的外部文件。

`resolve` 子命令使用执行 `generate` 步骤后输出的文件列表，然后搜索软件包的引用集来确定包含这些依赖文件的软件包的名称。为依赖文件搜索的软件包的引用集为当前在发布者的系统上安装的软件包。

提供的文件的多个组件用作依赖项信息的源：

ELF　分析提供的文件中的 ELF 头以获取依赖项信息，`-k` 和 `-D` 选项可用于修改获取的信息。有关 ELF 依赖项的更多详细信息，请参见 `ldd`(1) 和《链接程序和库指南》。

脚本　包含引用某个解释程序的 `#!` 行的 Shell 脚本会导致对提供该解释程序的程序包出现依赖项。

Python　Python 脚本首先作为脚本进行分析。此外，脚本声明的任何导入可能还会充当依赖项信息的源。

硬链接　清单中的硬链接会导致对提供链接目标的软件包出现依赖项。

SMF　提供的包含 `require_all` 依赖项的 SMF 服务清单会导致对提供 SMF 清单（这些清单提供这些 FMRI）的软件包出现依赖项。

选项

支持以下选项：

`-R` *dir*　对根目录为 *dir* 的映像进行操作。如果未根据环境指定或确定目录，则缺省值为 `/`。有关更多信息，请参见“环境变量”部分。

`--help` 或 `-?`　显示用法消息。

子命令

支持以下子命令：

`generate [-IMm] -d` *dir* `[-d` *dir*`] [-D` *name*`=`*value*`] [-k` *path*`]` *manifest_file*

生成由 *manifest_file* 指定的清单文件的依赖项。

使用 `-I` 时，将显示 *manifest_file* 中满足条件的依赖项。请勿使用带 `-I` 选项的 `pkgdepend resolve` 命令产生的结果。

使用 `-M` 时，将显示无法分析的文件类型的列表。

使用 -m 时，将重复原始清单，并在其后添加任何发现的依赖项。

使用 -d 时，将 *dir* 添加到目录列表以搜索清单的文件。

对于每个 -D，将添加 *value*，作为在 ELF 文件依赖项的运行路径中扩展标记 *name* 的一种方法。

对于每个 -k，将 *path* 添加到运行路径的列表，以搜索内核模块。使用 -k 参数会删除缺省路径 /kernel 和 /usr/kernel。

运行路径（例如 -k 选项指定的那些运行路径）也可以通过使用操作或清单属性 pkg.depend.runpath 来按操作或清单指定。pkg.depend.runpath 属性的值是要使用的路径的冒号分隔字符串。

使用的 -k 将被清单或操作中设置的任何 pkg.depend.runpath 属性覆盖。

特殊标记 $PKGDEPEND_RUNPATH 可用作 pkg.depend.runpath 属性值的一个组件，以包括所分析文件的标准系统运行路径。

在某些情况下，您可能需要防止自动生成依赖项。例如，如果软件包提供了一个用于导入一组模块的样例 Python 脚本，则该样例脚本导入的那些模块不是提供该样例脚本的软件包的依赖项。使用操作或清单属性 pkg.depend.bypass-generate 可以防止针对指定的文件生成依赖项。

pkg.depend.bypass-generate 值是与文件名匹配的 Python 正则表达式。正则表达式隐式固定在文件路径的开头和结尾。以下示例中提供的值与 this/that 匹配，但与 something/this/that/the/other 不匹配。

```
pkg.depend.bypass-generate=this/that
```

有关 Python 正则表达式语法的更多信息，请使用命令 pydoc re 或参见 http://docs.python.org/dev/howto/regex.html 中的更完整文档。

当 pkgdepend generate 输入清单包含 SMF 清单文件时，由这些 SMF 清单文件声明的任何 SMF 服务或实例都将包括在 pkgdepend 输出中。这些 SMF 服务或实例以名称为 org.opensolaris.smf.fmri 的 set 操作的形式包括在输出中。

resolve [-moSv] [-d *output_dir*] [-s *suffix*] *manifest_file* ...

将文件中的依赖项转换为提供这些文件的软件包中的依赖项。先根据命令行中给定的清单解析依赖项，然后再根据系统上安装的软件包进行解析。缺省情况下，每个清单的依赖项放置在名为 *manifest_file*.res 的文件中。

使用 -m 时，将重复清单，删除 generate 步骤生成的任何依赖项，然后添加已解析的依赖项。

使用 -o 时，会将结果写入到标准输出。该选项旨在供用户使用。将此输出附加到某个文件可能会导致产生无效的清单。在用于清单处理的管道中使用时，强烈建议使用 -d 或 -s 选项，而不要使用 -o。

使用 -d 时，会将单独文件中提供的每个清单的已解析依赖项写入 *output_dir* 中。缺省情况下，每个文件与清单（该清单是写入该文件的依赖项的源）具有相同的基名。

使用 -s 时，对于每个输出文件，会将 *suffix* 附加到文件（该文件是解析的依赖项的源）的基名。"." 附加到 *suffix* 的前面（如果未提供）。

使用 -S 时，只根据命令行上指定的清单进行解析，而不根据系统上安装的清单进行解析。

使用 -v 时，将包括其他软件包依赖项调试元数据。

示例

示例 1 生成依赖项

为 foo 中写入的清单（其内容目录在 ./bar/baz 中）生成 赖项，并将结果存储在 foo fdeps 中。

```
$ pkgdepend generate -d ./bar/baz foo > foo.fdeps
```

示例 2 解析依赖项

根据彼此的情况和当前系统上安装的软件包来解析 foo.fdeps 和 bar.fdeps 中的文件依赖项。

```
$ pkgdepend resolve foo.fdeps bar.fdeps
$ ls *.res
foo.fdeps.res    bar.fdeps.res
```

示例 3 生成并解析两个清单的依赖项

生成两个清单（foo 和 bar）的文件依赖项，并保留原始清单中的所有信息。然后解析文件依赖项，并将生成的清单放置在 ./res 中。这些生成的清单可以和 pkgsend publish 一起使用。

```
$ pkgdepend generate -d /proto/foo -m foo > ./deps/foo
$ pkgdepend generate -d /proto/bar -m bar > ./deps/bar
$ pkgdepend resolve -m -d ./res ./deps/foo ./deps/bar
$ ls ./res
foo    bar
```

示例 4 将值添加到 ELF 文件依赖项的标记

在为 foo 中写入的清单（其内容目录在 / 中）生成依赖项时，将 ELF 文件中运行路径内的所有 PLATFORM 标记替换为 sun4v 和 sun4u。

```
$ pkgdepend generate -d / -D 'PLATFORM=sun4v' -D 'PLATFORM=sun4u' foo
```

示例 5 指定内核模块目录

在为 foo 中写入的清单（其内容目录在 / 中）生成依赖项时，将 /kmod 指定为要在其中查找内核模块的目录。

示例 5 指定内核模块目录 （续）

```
$ pkgdepend generate -d / -k /kmod foo
```

示例 6 绕过依赖项生成

将 opt/python 附加到给定 Python 脚本的标准 Python 运行路径，然后根据名称为 test 的所有 Python 模块绕过作为 opt/python/foo/file.py 提供的文件的依赖项生成。

避免针对 usr/lib/python2.6/vendor-packages/xdg 中提供的任何文件生成依赖项。

```
$ cat manifest.py
set name=pkg.fmri value=pkg:/mypackage@1.0,1.0
set name=pkg.summary value="My test package"
dir path=opt mode=0755 group=sys owner=root
dir path=opt/python mode=0755 group=sys owner=root
dir path=opt/python/foo mode=0755 group=sys owner=root
file NOHASH path=opt/python/__init__.py mode=0644 group=sys owner=root
file NOHASH path=opt/python/foo/__init__.py mode=0644 group=sys owner=root
#
# Add runpath and bypass-generate attributes:
#
file NOHASH path=opt/python/foo/file.py mode=0644 group=sys owner=root \
    pkg.depend.bypass-generate=.*/test.py.* \
    pkg.depend.bypass-generate=.*/testmodule.so \
    pkg.depend.bypass-generate=.*/test.so \
    pkg.depend.bypass-generate=usr/lib/python2.6/vendor-packages/xdg/.* \
    pkg.depend.runpath=$PKGDEPEND_RUNPATH:/opt/python

$ pkgdepend generate -d proto manifest.py
```

环境变量

PKG_IMAGE　指定包含要用于软件包操作的映像的目录。如果指定 -R，则忽略该值。

退出状态

将返回以下退出值：

0　一切正常工作。

1　出现错误。

2　指定的命令行选项无效。

99　发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg

属性类型	属性值
接口稳定性	Uncommitted（未确定）

另请参见

pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pkgdiff – 比较软件包清单

用法概要

```
/usr/bin/pkgdiff [-i attribute ...] [-o attribute]
    [-v name=value ...] file1 file2
```

描述

`pkgdiff` 可比较两个软件包清单并报告差异。在进行比较之前，`pkgdiff` 会将每个清单和操作按一致的顺序排序。

输出采用以下格式：

+ *complete_action*　此操作在 *file2* 中而不在 *file1* 中。

- *complete_action*　此操作在 *file1* 中而不在 *file2* 中。

actionname keyvalue [*variant values, if any*]

- *attribute1=value1*　此 *attribute,value* 在 *file1* 中而不在 *file2* 中。

+ *attribute2=value2*　此 *attribute,value* 在 *file2* 中而不在 *file1* 中。

为了完成比较，带有不同变体、但带有相同类型和关键属性值的操作将被视为不同的操作。因此，可更改属性的操作将以其完整格式显示，而不是作为属性更改显示。

选项

支持以下选项：

`-i` *attribute*　比较期间将忽略 *attribute*（如果存在）。可以使用 `-i` *hash* 忽略文件散列值。该选项不能与 `-o` 选项一起使用。可以重复该选项。

`-o` *attribute*　仅报告 *attribute* 的差异。该选项不能与 `-i` 选项一起使用。该选项会省略不影响操作 *attribute* 的任何操作更改。

`-v` *name= value*　仅计算该变体值的差异。例如，只计算 `arch=sparc` 的差异。在进行比较之前，将删除所有操作的该变体标记。只能为每个变体指定一个值。可对不同的变体重复该选项。

退出状态

将返回以下退出值：

`0`　未找到差异。

`1`　找到了差异。

`>1`　出现错误。

`99`　发生了意外的异常。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	`package/pkg`
接口稳定性	Uncommitted（未确定）

另请参见

pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pkgfmt – 格式化软件包清单

用法概要

```
/usr/bin/pkgfmt [-c|-d|-u] [package-manifest-file]
```

描述

不带 `-c` 或 `-d` 选项的 `pkgfmt` 将以一致的方式对软件包清单进行格式化，包括每 80 个字符自动换行，按类型对操作排序，以及对属性排序。未解析成操作（如宏、注释或转换）的行不会按排序顺序显示。

如果未提供参数，`pkgfmt` 将会读取 `stdin`，直到遇到 EOF，然后将格式化的清单写入 `stdout`。在命令行上指定的任何清单将在原位格式化。

带 `-c` 选项的 `pkgfmt` 将会检查清单是否格式化为 `pkgfmt` 样式。如果文件未正确格式化，`-d` 选项将显示差异。

选项

支持以下选项：

`-c` 检查清单是否格式化为 `pkgfmt` 样式。

`-d` 以统一格式显示与格式化版本的清单差异。

`-u` 达到 80 个字符时不自动换行。将传统的文本处理工具应用到软件包清单时，该选项十分有用。

退出状态

将返回以下退出值：

`0` 命令成功。

`1` 指定了 `-c` 或 `-d` 选项，并且一个或多个清单不处于 `pkgfmt` 正常格式，或出现了错误。

`2` 指定的命令行选项无效。

`99` 发生了意外的异常。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	`package/pkg`
接口稳定性	Uncommitted（未确定）

另请参见

`pkg(5)`

`http://hub.opensolaris.org/bin/view/Project+pkg/`

引用名

pkglint – 映像包管理系统软件包 lint

用法概要

```
/usr/bin/pkglint [-c dir] [-r uri] [-p regexp]
    [-f rcfile] [-b build_no] [-v]
    [-l uri] | manifest ...

/usr/bin/pkglint -L [-v]
```

描述

pkglint 对一个或多个软件包清单运行一系列检查，并有选择性地引用其他系统信息库。

在发布软件包之前，应该在软件包构建期间使用 pkglint。pkglint 将对清单执行全面的测试，在正常操作 pkgsend(1) 或 pkg.depotd(1M) 期间，执行这种测试可能会消耗过多的资源。 pkglint 检查包括对重复操作、缺少的属性和非常规文件权限进行测试。

可以在命令行上将用于 lint 的清单作为本地文件的空格分隔列表进行传递，也可以从系统信息库检索清单。

从系统信息库检索清单时，首次运行时 pkglint 将在指定的高速缓存目录中创建并填充 pkg(5) 用户映像。如果提供了 -r 选项，则为引用系统信息库创建名为 *cache_dir*/ref_image 的用户映像。如果提供了 -l 选项，则为 lint 系统信息库创建名为 *cache_dir*/lint_image 的用户映像。不会在这些映像中安装任何内容。pkglint 只使用这些映像从系统信息库中检索清单。

pkglint 的后续调用可以重新使用高速缓存目录，并可以省略任何 -r 或 -l 参数。

pkglint 对在高速缓存目录中配置发布者提供有限的支持。使用 pkg(1) 可对这些映像执行更复杂的发布者配置。

pkglint 使软件包作者能够绕过对给定清单或操作进行的检查。其中的属性 pkg.linted 设置为 True 的清单或操作不会为该清单或操作生成任何 lint 输出。

使用 pkglint 检查名称的子字符串可以进行更高粒度的 pkg.linted 设置。例如，设置为 True 的 pkg.linted.*check*.*id* 将对给定的清单或操作绕过名称为 *check*.*id* 的所有检查。

可通过指定一个 pkglintrc 文件来配置 pkglint 的行为。缺省情况下，pkglint 在 /usr/share/lib/pkg/pkglintrc 和 $HOME/.pkglintrc 中搜索配置选项。使用 -f 选项可以指定其他配置文件。

lint 运行期间，遇到的任何错误或警告将输出到 stderr。

选项

支持以下选项：

-b *build_no*　指定内部版本号，用于细化从 lint 和引用系统信息库执行 lint 期间使用的软件包列表。如果不指定 -b 选项，将使用最新版本的软件包。另请参见 version.pattern 配置属性。

-c *cache_dir*　指定用于从 lint 和引用系统信息库高速缓存软件包元数据的本地目录。

`-l` *lint_uri*	指定表示 lint 系统信息库位置的 URI。支持基于 HTTP 和基于文件系统的发布。如果指定 `-l`，则还必须指定 `-c`。
`-L`	列出已知的和排除的 lint 检查，然后退出。显示每个检查的短名称和描述。与 `-v` 标志结合使用时，将显示实现检查的方法而不显示描述。
`-f` *config_file*	使用 *config_file* 配置文件来配置 `pkglint` 会话。
`-p` *regexp*	指定正则表达式，用于细化要从 lint 系统信息库检查的软件包列表。将会加载来自引用系统信息库的所有清单（假定它们与提供的 `-b` 值匹配），并忽略此模式。
`-r` *repo_uri*	指定表示引用系统信息库位置的 URI。如果指定 `-r`，则还必须指定 `-c`。
`-v`	以详细模式运行 `pkglint`，覆盖配置文件中的任何 `log_level` 设置。
`--help` 或 `-?`	显示用法消息。

文件

`pkglintrc` 配置文件采用以下键/值参数：

`log_level`	发出 lint 消息的最低级别。低于该级别的 lint 消息将被放弃。缺省值为 `INFO`。 严重性从低到高的日志级别依次为 `DEBUG`、`INFO`、`WARNING`、`ERROR` 和 `CRITICAL`。
`do_pub_checks`	如果为 `True`，则执行可能只对已发布的软件包有意义的检查。缺省值为 `True`。
`pkglint.ext.*`	`pkglint` 的插件机制允许在运行时添加其他 lint 模块。以 `pkglint.ext.` 开头的任何键采用必须是完全指定的 Python 模块的值。有关更多信息，请参见“开发工具”一节。
`pkglint.exclude`	要从执行的检查集中省略的完全指定的 Python 模块、类或函数名称的空格分隔列表。
`use_progress_tracker`	如果为 `True`，则在 lint 运行期间迭代清单时，会使用进度跟踪器。缺省值为 `True`。
`version.pattern`	指定内部版本号以针对 (`-b`) 执行 lint 时使用的版本模式。如果未在配置文件中指定，则 `-b` 选项将使用模式 `*,5.11-0.`，并使用分支前缀 0 来匹配 5.11 内部版本的所有组件。

开发工具

扩展 `pkglint`、子类 `pkg.lint.base.Checker` 及其子类 `ManifestChecker`、`ActionChecker` 和 `ContentChecker` 执行的检查集。将包含这些类的 Python 模块名称添加到配置文件中的新 `pkglint.ext.` 键。

这些新子类的实例由 pkglint 在启动时创建。lint 会话过程中，将会调用带有特殊关键字参数 pkglint_id 的每个子类中的方法。这些方法应该与超类中的相应 check() 方法具有相同的签名。还应该为方法指定一个 pkglint_desc 属性，该属性用作 pkglint -L 输出的描述。

参数可用于 Checker 子类，使这些子类能够调优其行为。建议的参数命名约定为 *pkglint_id*.*name*。参数值可以存储在配置文件中，或者在使用 LintEngine.get_param() 方法检索的清单或操作中访问。从清单访问参数时，将在键名的前面附加前缀 pkg.lint，以确保 pkglint 参数不会与任何现有的操作或清单值重叠。

示例

示例 1 对特定系统信息库的首次运行

对给定系统信息库上首次运行 pkglint 会话。

```
$ pkglint -c /space/cache -r http://localhost:10000 mymanifest.mf
```

示例 2 对相同系统信息库的后续运行

针对示例 1 中使用的同一系统信息库的后续运行。

```
$ pkglint -c /space/cache mymanifest-fixed.mf
```

示例 3 将 Lint 系统信息库和细化的清单集一起使用

将 pkglint 会话和 lint 系统信息库结合运行，并指定要检查的清单的子集。

```
$ pkglint -c /space/othercache -l http://localhost:10000 \
-p '.*firefox.*'
```

示例 4 指定内部版本

在详细模式下针对给定的内部版本运行 pkglint 会话。

```
$ pkglint -c /space/cache -r http://localhost:10000 \
-l http://localhost:12000 -b 147 -v
```

示例 5 修改配置文件

带有新的 lint 模块的配置文件，排除某些检查。

```
$ cat ~/.pkglintrc
[pkglint]

log_level = DEBUG
# log_level = INFO

pkglint.ext.mycheck = org.timf.mychecks
pkglint.ext.opensolaris = pkg.lint.opensolaris
pkglint.exclude: pkg.lint.opensolaris.OpenSolarisActionChecker
pkg.lint.pkglint.PkgActionChecker.unusual_perms pkg.lint.pkglint.PkgManifestChecker
```

示例 5 修改配置文件 （续）

```
pkg.lint.opensolaris.OpenSolarisManifestChecker
```

退出状态

将返回以下退出值：

0 命令成功。

1 一项或多项 lint 检查提供了输出。

2 指定的命令行选项无效。

99 发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkg(1)、pkg.depotd(1m)、pkgsend(1)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pkgmerge – 映像包管理系统软件包合并实用程序

用法概要

```
/usr/bin/pkgmerge [-n] -d dest_repo
    -s variant=value[,...],src_repo ...
    [pkg_fmri_pattern ...]
```

描述

pkgmerge 是一种用于创建多变量软件包的软件包发布工具。它通过合并具有相同名称和版本的软件包（时间戳除外），使用给定源的指定变量名称和值标记合并的版本中的唯一操作，然后将新的软件包发布到目标系统信息库，从而实现此操作。仅使用每个源中各软件包的最新版本。

如果将某一操作的 pkg.merge.blend 属性设置为要合并变量的名称，则在合并之前将该操作复制到其他清单，以便在最终输出中显示该操作时不显示任何已添加的变量标记。请注意，属性 pkg.merge.blend 本身已从输出清单的所有操作中删除。对于多个传递合并，可以使用不同的值重复该属性。

在输入清单中交付到同一路径的不同操作将导致 pkgmerge 因错误而退出。

选项

支持以下选项：

-d *dest_repo*
要将合并软件包发布到的目标系统信息库的文件系统路径或 URI。目标系统信息库必须已存在。可以使用 pkgrepo(1) 创建新的系统信息库。

-n
执行试运行，不对目标系统信息库进行任何更改。

-s *variant*= *value*[,...],*src_repo*
用于该源中软件包的变量名称和值，后跟要从中检索软件包的源系统信息库或软件包归档文件的文件系统路径或 URI。可指定多个变量，以逗号分隔。必须为所有源命名相同的变量。可以多次指定此选项。

--help 或 -?
显示用法消息。

环境变量

支持以下环境变量：

TMPDIR　在程序执行期间用于存储临时数据的目录的绝对路径。如果未设置，则存储临时数据的缺省路径为 /var/tmp。

示例

示例 1 指定变量名称和值

使用为从中检索到软件包的源指定的给定变量名称和值标记在指定源中发现的每个软件包：

```
$ pkgmerge -s arch=sparc,http://src.example.com \
-d http://dest.example.com
```

样例软件包：

示例 1 指定变量名称和值 （续）

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
dir group=sys mode=0755 owner=root path=usr
```

操作后的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
set name=variant.arch value=sparc
dir group=sys mode=0755 owner=root path=usr
```

示例 2 合并和发布软件包

合并给定源中每个软件包的最新版本并将新软件包发布到目标系统信息库：

```
$ pkgmerge -s arch=sparc,http://src1.example.com \
-s arch=i386,http://src2.example.com \
-d /path/to/target/repository
```

源 1 (SPARC) 中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121410Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

源 2 (i386) 中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

合并软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
set name=variant.arch value=sparc value=i386
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=sparc
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=i386
dir group=sys mode=0755 owner=root path=usr
```

示例 3 合并 i386 和 SPARC 系统的调试和非调试软件包

在一组用于 i386 和 SPARC 系统的调试及非调试系统信息库中，合并每个软件包的最新版本：

```
$ pkgmerge -s arch=sparc,debug=false,/repo/sparc-nondebug \
-s arch=sparc,debug=true,/repo/sparc-debug \
-s arch=i386,debug=false,/repo/i386-nondebug \
-s arch=i386,debug=true,/repo/i386-debug \
-d /path/to/target/repository
```

源 1（SPARC 非调试）中的样例软件包：

示例 3 合并 i386 和 SPARC 系统的调试和非调试软件包 （续）

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121410Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

源 2（SPARC 调试）中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121411Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

源 3（i386 非调试）中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

源 4（i386 调试）中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163428Z
file id mode=0555 owner=root group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

合并软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163428Z
set name=variant.arch value=sparc value=i386
set name=variant.debug value=false value=true
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=sparc variant.debug=false
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=sparc variant.debug=true
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=i386 variant.debug=false
file id mode=0555 owner=root group=bin path=usr/bin/foo variant.arch=i386 variant.debug=true
dir group=sys mode=0755 owner=root path=usr
```

示例 4 使用 pkg.merge.blend 合并

使用 `pkg.merge.blend` 属性合并两个不相互冲突的体系结构的软件包。

```
$ pkgmerge -s arch=sparc,http://src1/example.com \
-s arch=i386,http://src2.example.com \
-d /path/to/target/repository
```

源 1 (SPARC) 中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T121410Z
file 1d5eac1aab628317f9c088d21e4afda9c754bb76 mode=0555 owner=root \
    group=bin path=usr/bin/sparc/foo pkg.merge.blend=arch
file d285ada5f3cae14ea00e97a8d99bd3e357caadc0 mode=0555 owner=root \
    group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

示例 4 使用 pkg.merge.blend 合并 （续）

源 2 (i386) 中的样例软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
file a285ada5f3cae14ea00e97a8d99bd3e357cb0dca mode=0555 owner=root \
    group=bin path=usr/bin/i386/foo pkg.merge.blend=arch
file d285ada5f3cae14ea00e97a8d99bd3e357caadc0 mode=0555 owner=root \
    group=bin path=usr/bin/foo
dir group=sys mode=0755 owner=root path=usr
```

合并软件包：

```
set name=pkg.fmri value=pkg://example.com/foo@5.11,5.11-0.200:20381001T163427Z
set name=variant.arch value=sparc value=i386
file d285ada5f3cae14ea00e97a8d99bd3e357caadc0 mode=0555 owner=root \
    group=bin path=usr/bin/foo
file a285ada5f3cae14ea00e97a8d99bd3e357cb0dca mode=0555 owner=root \
    group=bin path=usr/bin/i386/foo
file 1d5eac1aab628317f9c088d21e4afda9c754bb76 mode=0555 owner=root \
    group=bin path=usr/bin/sparc/foo
dir group=sys mode=0755 owner=root path=usr
```

退出状态

将返回以下退出值：

`0` 命令成功。

`1` 出现错误。

`2` 指定的命令行选项无效。

`99` 发生了意外的异常。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	`package/pkg`
接口稳定性	Uncommitted（未确定）

另请参见

`pkgrepo(1)`、`pkg(5)`

`http://hub.opensolaris.org/bin/view/Project+pkg/`

引用名

pkgmogrify – 映像包管理系统清单变换程序

用法概要

```
/usr/bin/pkgmogrify [-vi] [-I includedir ...]
    [-D macro=value ...] [-O outputfile]
    [-P printfile] [inputfile ...]
```

描述

pkgmogrify 允许对软件包清单进行编程式编辑，以简化自动生成软件和自动发布软件包时所需的典型变换。

pkgmogrify 提供：

- 宏替换，以便于在各种体系结构和平台间共享单个清单。
- 包含其他清单或清单片段，例如标准组件和变换。
- 变换软件包操作，包括修改、删除或添加操作属性。

选项

支持以下选项：

-D *name*= *value*
将 *name* 定义为一个宏，其值为 *value*。宏在输入文件中显示为 $(macro)。将重复执行替换操作，直到再也找不到变换。常见用语包括：

- 通过在行的开头使用特定于体系结构的标记来删除其他体系结构清单中的行。

  ```
  $(sparc_ONLY)file ...
  ```

  处理 SPARC 体系结构时，该宏将被设置为空字符串。而处理其他体系结构时，该宏在命令行中会被设置为 #，从而从当前体系结构的清单中删除此操作。

- 指定路径名称中特定于平台的部分，例如，可执行文件和库的 64 位体系结构目录的名称：

  ```
  file NOHASH path=usr/bin/$(ARCH64)/cputrack ...
  ```

  应当将这些宏设置为命令行中所需的值。不存在预定义的宏值。

-I *include_directory*
将指定目录添加到在命令行中和在嵌入式 include 指令中指定的文件的搜索路径中。

-O *outputfile*
将清单输出写入指定的文件中。如果出现错误或 transform 指令引起中止操作，则不会写入该文件。缺省情况下，清单输出会写入 stdout。

-P *printfile*
将 transform 指令输出操作生成的输出写入指定的文件。如果出现错误或 transform 指令引起中止操作，则不会写入该文件。缺省情况下，输出会写入 stdout。

-i
忽略文件中的 include 指令。仅处理在命令行（或 stdin）中指定的文件。

-v
将注释写入显示变换效果的输出清单。该信息可以帮助调试。

`--help` 或 `-?`
显示用法消息。

嵌入式指令 清单文件支持两种类型的指令：include 指令和 transform 指令。

Include 指令的格式为：

`<include` *file*`>`

该指令能够使 `pkgmogrify` 首先在当前目录中搜索名为 `file` 的文件，然后在 `-I` 选项指定的目录中搜索该文件。如果找到，则会将该文件的内容插入清单中遇到该指令的位置。如果未找到，`pkgmogrify` 将退出并显示错误。

Transform 指令的格式为：

`<transform` *matching-criteria* `->` *operation*`>`

这些指令会进行累积（直到将所有输入都读入内存中），然后按遇到这些指令的顺序应用到操作。

匹配条件的格式为：

`[`*action-type* `... ] [`*attribute*`=<`*value-regexp*`> ...]`

必须匹配其中一个指定的 *action-types*。必须匹配所有指定的 *attributes*。使用的正则表达式语法是 Python 语法。有关 Python 正则表达式语法的更多信息，请使用命令 `pydoc re` 或参见 `http://docs.python.org/dev/howto/regex.html` 中的更完整文档。该正则表达式固定于开头而非结尾。因此，通过文件扩展名进行匹配的正则表达式匹配文件必须包含前导 `.*`，且应当包含尾随 `$`。

可以指定多个条件，以空格分隔。

有以下操作可用：

`add` 将值添加到属性中。此操作使用两个参数。第一个参数是属性的名称，第二个参数是对应值。

`default` 如果不存在属性值，则设置该属性值。此操作与 `add` 操作使用相同的两个参数。

`delete` 删除属性值。此操作使用两个参数。第一个参数是属性的名称。第二个参数是要与删除的属性值匹配的正则表达式。与用于匹配操作的正则表达式不同，该正则表达式并不是固定的。

`drop` 放弃该操作。

`edit` 修改操作的属性。此操作使用三个参数。第一个参数是属性的名称，第二个参数是与属性值匹配的正则表达式。第三个参数是用于替换正则表达式匹配的部分值的替换字符串。与用于匹配操作的正则表达式不同，该正则表达式并不是固定的。如果在正则表达式中定义了组，则在替换字符串中可以使用 `\1`、`\2` 等格式的一般正则表达式向后引用。

`emit` 向清单输出流中发出一行。此行必须是有效操作字符串、为空（导致空白行）或注释（ # 后面跟有任意文本）。

`exit` 终止清单处理过程。不输出任何清单，也不应用任何 `print` 操作。如果给定一个参数，该参数必须是整数并且用作退出代码。缺省为 0。如果给定两个参数，则第一个参数是退出代码，第二个参数是要输出到 `stderr` 的消息。

`print` 将消息输出到 `-P` 指定的输出文件中。

`set` 设置属性的值。此操作与 `add` 操作使用相同的两个参数。

除了 `delete` 和 `drop` 以外的所有操作都使用其内容输出到输出流的参数（可能是可选的）。这些字符串可能包含三种不同类型的特殊标记，这些标记允许输出包含不基于每种操作的固定变换的信息。

第一种替换通过将属性的名称放置在括号内（跟随在百分号后面），允许操作引用当前操作的属性值。例如，`%(alias)` 引用操作的 `alias` 属性的值。

存在几个合成属性。以下两个对 `pkgmogrify` 而言是唯一的：

- `pkg.manifest.filename` 引用在其中找到操作的文件的名称。
- `pkg.manifest.lineno` 引用在其中找到操作的行。

以下三个合成属性类似于 `pkg`(1) 中使用的属性：

- `action.hash` 引用操作的散列值（如果该操作携带有效负荷）。对于携带有效负荷的操作， `set` 操作可以通过对 `action.hash` 属性进行操作来更改该操作的散列。
- `action.key` 引用关键属性的值。
- `action.name` 引用操作类型的名称。

如果请求其值的属性不存在，`pkgmogrify` 将退出并显示错误。为防止出现错误退出，请在属性名称后面附加 `;notfound=` 以及要用于替换属性值的值。例如，如果存在 `alias` 属性，则 `%(alias;notfound='no alias')` 输出该属性的值，否则输出 `no alias`。

如果请求其值的属性具有多个值，则输出每个值，以空格隔开。与 `notfound` 标记类似，标记 `prefix`、`suffix` 和 `sep` 也可用于更改此行为。由 `prefix` 表示的字符串放置在每个值之前，由 `suffix` 表示的字符串放置在每个值之后，而 `sep` 放置在某个值的后缀与下一个值的前缀之间。

与操作属性类似，`pkgmogrify` 指令也可以使用花括（而非圆括号）引用 件包属性：`%{pkg.fmri}`。应用 transform 指令时，必须已在 `set` 操作中定义了属性，否则会将其视为 `notfound`，如上所述。当处理过程到达清单文件（介绍软件包）的结尾处时，将清除下一个软件包的属性。

这不仅在将软件包属性视为操作属性来引用方面，而且在匹配甚至暂时修改这些属性方面，都很有用。因此，在这些情况下都可以使用合成属性名称 pkg（仅在 pkgmogrify 上下文中）。

如果 pkgmogrify 完成读取在命令行中指定的清单并且该清单定义了 pkg.fmri 软件包属性，pkgmogrify 会创建此合成 pkg 操作，其属性为软件包的属性。<transform> 指令随后会对此操作进行匹配（正如任何其他操作类型一样）。

pkg 操作中的操作是特殊的，它们仅在内存中进行，不会直接影响发出的清单。例如，尝试通过 add、default 或 set 操作设置 pkg 操作的属性时，不会使 set 操作添加到清单中，虽然该操作将可用于其他 <transform> 指令进行匹配。尝试 emitpkg 操作会导致错误。要添加软件包属性，改为 emitset 操作。

第三种替换是逆向引用功能。该替换与在 edit 操作中可使用的替换不同，它是对 -> 左侧的变换匹配中列出组的引用。它们由 %<1>、 %<2> 等表示（以在匹配条件中显示的顺序）。

处理顺序如下所示：

1. 从输入文件中读取行。
2. 应用宏。
3. 处理 <include ...> 和 <transform> 指令，从而找到并读取更多文件。
4. 累积所有输入之后，输出中的每行都会转换为操作并应用所有变换。
5. 成功完成处理后，写入输出。

示例

示例 1 将标记添加到 SMF 清单中

将标记添加到服务管理工具 (Service Management Facility, SMF) 清单中，以便在活动系统上安装软件包时导入这些标记。

```
<transform file path=(var|lib)/svc/manifest/.*\.xml -> \
    add restart_fmri svc:/system/manifest-import:default>
```

示例 2 移动文件

将文件从 usr/sfw/bin 移至 usr/bin。

```
<transform file -> edit path usr/sfw/bin usr/bin>
```

示例 3 指定需要重新引导

将 reboot-needed 标记添加到 /kernel 下 .conf 文件以外的文件中。请注意，以下示例利用了按照在输入文件中看见的顺序将变换应用到每个操作的方式。

```
<transform file path=kernel/.* -> set reboot-needed true>
<transform file path=kernel/.*\.conf -> delete reboot-needed .*>
```

这还可以通过包含正则表达式的单个变换规则完成。

示例 4 将 FMRI 属性转换为 Depend 操作

将软件包属性 `pkg.fmri` 转换为 `depend` 操作，使其成为集合的一部分。

```
<transform set name=pkg.fmri -> \
    emit depend type=incorporate fmri=%(value)>
<transform set name=pkg.fmri -> drop>
```

示例 5 输出错误编号列表

输出带有双引号和前缀的错误编号的逗号分隔列表。

```
set name=bugs value=12345 value=54321 value=13579 value=97531
<transform set name=bugs -> \
    print %(value;sep=",";prefix="bug='";suffix="'")>
```

示例 6 允许丢失属性

即使是在丢失属性时，也可以安全输出消息。

```
<transform driver -> print Found aliases: %(alias;notfound=<none>)>
```

示例 7 设置缺省值

设置缺省所有者、组以及权限的值。

```
<transform file dir -> default owner root>
<transform file dir -> default group bin>
<transform file -> default mode 0444>
<transform dir -> default mode 0755>
```

示例 8 将依赖项添加到未标记为已过时的软件包中

对于未标记为已过时的任何软件包，为提供软件包的合并添加集合的依赖项。必须在读入清单后执行该组变换，否则始终发出依赖项。因为修改 `pkg` 操作不会永久起作用，所以无需清除匹配 `pkg.obsolete=false` 的属性。

```
<transform pkg -> default pkg.obsolete false>
<transform pkg pkg.obsolete=false -> emit depend \
    fmri=consolidation/$(CONS)/$(CONS)-incorporation type=require>
```

示例 9 发现问题时退出并输出消息

在清单中发现已过时属性时退出并输出错误消息。

```
<transform file dir link hardlink opensolaris.zone=.* -> \
    exit 1 The opensolaris.zone attribute is obsolete.>
```

示例 10 设置合适的语言环境侧面

设置适用于正在考虑的路径名称的语言环境侧面。

示例 10 设置合适的语言环境侧面 **（续）**

```
<transform dir file link hardlink path=.*/locale/([^/]+).* -> \
    default facet.locale.%<1> true>
```

退出状态

将返回以下退出值：

0 一切正常工作。

1 出现预料中的错误情况。

2 指定的命令行选项无效。

99 意外的处理错误。

文件

`/usr/share/pkg/transforms` 该目录包含使用有用变换设置侧面、执行器和其他属性的文件。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	`package/pkg`
接口稳定性	Uncommitted（未确定）

另请参见

`pkg(1)`、`pkg(5)`

`http://hub.opensolaris.org/bin/view/Project+pkg/`

引用名

pkgrecv – 映像包管理系统内容检索实用程序

用法概要

```
/usr/bin/pkgrecv [-s src_uri] [-a] [-d (path|dest_uri)]
    [-c cache_dir] [-kr] [-m match] [-n] [--raw]
    [--key keyfile --cert certfile] (fmri|pattern) ...

/usr/bin/pkgrecv [-s src_uri] --newest
```

描述

pkgrecv 允许用户检索 pkg(5) 系统信息库或软件包归档文件中的软件包。pkgrecv 还可以选择性地将检索到的软件包重新发布到另一个软件包系统信息库中或者对其进行归档。缺省情况下，会以 pkg(1)、pkg.depotd(1M) 以及软件包发布工具可使用的软件包系统信息库格式检索软件包。

完成 pkgrecv 操作后，在系统信息库上运行 pkgrepo refresh 或 pkgrepo rebuild 以构建搜索索引。

选项

支持以下选项：

`-a` 将检索的软件包数据存储在 -d 指定的位置中的 pkg(5) 归档文件中。该文件不能已存在。此选项仅可以与基于文件系统的目标一起使用。尽管不要求，但还是强烈建议使用文件扩展名 .p5p（例如：archive.p5p）。该选项不能与 --raw 结合使用。

`-c` *cache_dir* 将用于缓存已下载内容的目录的路径。如果没有提供此目录，客户机将自动选择一个高速缓存目录。如果下载中断并且已自动选择了一个高速缓存目录，可使用该选项继续执行下载。有关如何设置用于存储临时数据的位置的详细信息，请参见下面的“环境变量”部分。

`-d` *path_or_uri* 要将软件包重新发布到的目标的文件系统路径或 URI。如果指定了 -a，则目标为尚不存在的新软件包归档文件。否则，目标必须是已存在的软件包系统信息库。可以使用 pkgrepo(1) 创建新的系统信息库。

`-h` 显示用法消息。

`-k` 使检索的软件包内容保持压缩状态。重新发布时会忽略该选项。压缩的软件包内容不得与 pkgsend(1) 一起使用。

`-m` *match* 使用以下值控制匹配行为：

- `all-timestamps` 包括所有匹配的时间戳，而不仅仅是最新的时间戳（意味着 all-versions）。
- `all-versions` 包括所有匹配的版本，而不仅仅是最新的版本。

`-n` 执行试运行，不进行任何更改。

`-r` 递归检索提供的软件包列表的所有依赖项。

`-s` *src_repo_uri* URI 代表要从中接收软件包数据的 pkg(5) 系统信息库或软件包归档文件的位置。

`--cert` *file* 指定用于从 HTTPS 系统信息库进行软件包检索的客户机 SSL 证书文件。

`--key` *file* 指定用于从 HTTPS 系统信息库进行软件包检索的客户机 SSL 密钥文件。

`--newest` 列出指定系统信息库中提供的最新版本的软件包，然后退出。（忽略 `-s` 以外的所有其他选项。）

`--raw` 按主干和版本，检索一组目录结构中的原始软件包数据并将其存储在 `-d` 指定的位置中。此选项仅可以与基于文件系统的目标一起使用。此软件包数据可用于方便地修改和重新发布软件包，也许通过更正文件内容，也许通过提供附加的软件包元数据。该选项不能与 `-a` 结合使用。

示例

示例 1 列出最新的软件包

列出名为 `test` 的系统上的系统信息库中的最新软件包。

```
$ pkgrecv -s http://test --newest
pkg://solaris/system/library/c++-runtime@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190358Z
pkg://solaris/system/library/freetype-2@2.4.8,5.11-0.175.1.0.0.7.1234:20120109T215840Z
pkg://solaris/system/library/math@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190432Z
```

示例 2 检索原始软件包数据

以能够与 `pkgsend publish` 结合使用的合适格式从示例 1 中接收 `c++-runtime` 软件包。

```
$ pkgrecv -s http://test \
-d /local/repo --raw \
c++-runtime@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190358Z
Processing packages for publisher solaris ...
Retrieving and evaluating 1 package(s)...
PROCESS                                          ITEMS     GET (MB)    SEND (MB)
Completed                                          1/1      3.5/3.5      0.0/0.0
$ ls /local/repo
pkg5.repository  publisher  system%2Flibrary%2Fc%2B%2B-runtime
```

示例 3 从系统中检索依赖项

从名为 `test` 的系统中接收软件包 `editor/vim` 及其所有依赖项。

```
$ pkgrecv -s http://test -d /local/repo -r editor/vim
```

示例 4 检索所有版本

从名为 `test` 的系统中接收软件包 `editor/vim` 的所有版本。

示例 4 检索所有版本 （续）

```
$ pkgrecv -s http://test -d /local/repo -m all-versions editor/vim
Processing packages for publisher solaris ...
Retrieving and evaluating 2 package(s)...
PROCESS                                          ITEMS     GET (MB)    SEND(MB)
Completed                                          2/2    16.7/16.7    44.9/44.9
```

示例 5 检索所有版本并远程重新发布

从名为 test 的系统中接收软件包 library/zlib 的所有版本，然后将其重新发布到名为 remote 的系统上的远程系统信息库中。

```
$ pkgrecv -s http://test -d http://remote:10000 -m all-versions library/zlib
```

示例 6 从系统信息库中检索依赖项

从位于 /export/repo 的系统信息库中接收软件包 editor/gnu-emacs 及其所有依赖项。

```
$ pkgrecv -s /export/repo -d /local/repo -r editor/gnu-emacs
```

示例 7 检索其他软件包

从位于 http://example.com:10000 的系统信息库中接收并非已存在的所有软件包。

```
$ pkgrecv -s http://example.com:10000 -d /my/pkg/repo '*'
```

示例 8 创建软件包归档文件

根据位于 http://example.com:10000 的系统信息库创建包含软件包 editor/gnu-emacs 及其所有依赖项的软件包归档文件。

```
$ pkgrecv -s http://example.com:10000 -d /my/emacs.p5p -a -r editor/gnu-emacs
```

示例 9 将软件包从归档文件复制到系统信息库中

将软件包归档文件中的所有软件包复制到位于 /export/repo 中的现有系统信息库。

```
$ pkgrecv -s /my/archive.p5p -d /export/repo '*'
```

环境变量

支持以下环境变量：

PKG_DEST　要将检索到的软件包保存到的目录的路径，或者要复制软件包的系统信息库或软件包归档文件的文件系统路径或 URI。

PKG_SRC　URI 或文件系统路径代表要从中检索软件包的 pkg(5) 系统信息库或软件包归档文件的位置。

TMPDIR　在程序执行期间用于存储临时数据的目录的绝对路径。如果未设置，则存储临时数据的缺省路径为 /var/tmp。

退出状态

将返回以下退出值：

0 命令成功。

1 出现错误。

2 指定的命令行选项无效。

3 请求了多项操作，但只有一部分操作成功。

99 发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkgrepo(1)、pkgsend(1)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pkgrepo – 映像包管理系统的系统信息库管理实用程序

用法概要

```
/usr/bin/pkgrepo create [--version ver] uri_or_path

/usr/bin/pkgrepo add-publisher -s repo_uri_or_path publisher ...

/usr/bin/pkgrepo get [-F format] [-p publisher ...]
    -s repo_uri_or_path [section/property ...]

/usr/bin/pkgrepo info [-F format] [-H] [-p publisher ...]
    -s repo_uri_or_path

/usr/bin/pkgrepo list [-F format] [-H] [-p publisher ...]
    -s repo_uri_or_path [pkg_fmri_pattern ...]

/usr/bin/pkgrepo rebuild [-p publisher ...]
    -s repo_uri_or_path [--no-catalog] [--no-index]

/usr/bin/pkgrepo refresh [-p publisher ...]
    -s repo_uri_or_path [--no-catalog] [--no-index]

/usr/bin/pkgrepo remove [-n] [-p publisher ...]
    -s repo_uri_or_path pkg_fmri_pattern ...

/usr/bin/pkgrepo set [-p publisher] -s repo_uri_or_path
    section/property=[value] ... or
    section/property=([value]) ...

/usr/bin/pkgrepo help

/usr/bin/pkgrepo version
```

描述

通过 pkgrepo 可以创建和管理 pkg(5) 软件包系统信息库。软件包系统信息库是一组预定义的目录和文件，允许 pkg(1) 和发布客户机（例如 pkgsend (1) 或 pkgrecv(1)）存储和检索软件包数据。此外，当需要对软件包系统信息库进行基于网络的访问时，pkg.depotd(1m) 可以提供对该系统信息库的客户机访问权限，以存储和/或检索软件包数据。

选项

支持以下选项：

--help 或 -?　　显示用法消息。

子命令

支持以下子命令：

create [--version *ver*] *uri_or_path*

在指定的位置创建 pkg(5) 系统信息库。

该子命令仅可以与基于文件系统的系统信息库一起使用。

使用 --version 时，将以与指定版本兼容的格式创建系统信息库。缺省情况下，会创建版本为 4 的系统信息库。支持的版本包括：

3　　支持为单个发布者存储软件包，目录版本为 1，搜索版本为 1。

4　　支持为多个发布者存储软件包，目录版本为 1，搜索版本为 1。

`add-publisher -s` *repo_uri_or_path publisher ...*
将指定的发布者添加到系统信息库中。新的发布者没有软件包或内容。

该子命令仅可以与基于第 4 版文件系统的系统信息库一起使用。

`get [-F` *format*`] [-p` *publisher ...*`] -s` *repo_uri_or_path* `[`*section/property ...*`]`
显示系统信息库或其发布者的属性信息。

缺省情况下，会在单独的行中显示每个属性及其值。空的 ASCII 字符串值用一对双引号 ("") 表示。ASCII 字符串值中的以下 Bourne shell 元字符以及换行符、空格符和制表符都必须使用反斜杠 (\) 进行转义：

```
; & ( ) | ^ < > \ " ' `
```

请参见“示例”部分。

有关可能的属性的列表以及每个属性的用途和值，请参见下面的 `set` 子命令。

使用 `-F` 时，指定备用输出格式。*format* 的值可以是 `tsv`（以制表符分隔的值）、`json`（单行 JavaScript 对象表示法）或 `json-formatted`（格式易于阅读的 JavaScript 对象表示法）。

使用 `-H` 时，将忽略列表的标题。

使用 `-p` 时，将显示给定发布者的属性信息。特殊值 `all` 显示所有发布者的属性。可以多次指定此选项。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

`info [-F` *format*`] [-H] [-p` *publisher ...*`] - s` *repo_uri_or_path*
显示系统信息库已知的软件包发布者的列表。该列表包括每个发布者的软件包数量、最后一次更新发布者的软件包数据的时间以及发布者的软件包数据的状态（例如当前是否处于正在处理状态）。

使用 `-F` 时，指定备用输出格式。*format* 的值可以是 `tsv`（以制表符分隔的值）、`json`（单行 JavaScript 对象表示法）或 `json-formatted`（格式易于阅读的 JavaScript 对象表示法）。

使用 `-H` 时，将忽略列表的标题。

使用 `-p` 时，仅显示给定发布者的数据。如果没有提供该选项，将显示所有发布者的数据。可以多次指定此选项。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

`list [-F` *format*`] [-H] [-p` *publisher ...*`] -s` *repo_uri_or_path* `[`*pkg_fmri_pattern ...*`]`
列出 *repo_uri_or_path* 系统信息库中与指定 *pkg_fmri_pattern* 模式匹配的软件包。如果未指定任何模式，将列出系统信息库中的所有软件包。

在缺省输出中，第一列包含软件包发布者的名称。第二列包含软件包的名称。第三列是显示软件包状态的标志。状态列中的 `o` 值表示软件包已过时。状态列中的 `r` 值表示已重命名软件包，但格式已过时。第四列包含软件包的发行版本和分支版本。有关发行版本和分支版本的信息，请参见 `pkg(5)`。

使用 `-F` 时，指定备用输出格式。*format* 的值可以是 `tsv`（以制表符分隔的值）、`json`（单行 JavaScript 对象表示法）或 `json-formatted`（格式易于阅读的 JavaScript 对象表示法）。

使用 `-H` 时，将忽略列表的标题。

使用 `-p` 时，仅显示给定发布者的软件包。如果没有提供该选项，将显示所有发布者的软件包。可以多次指定此选项。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

`rebuild [-p` *publisher* `...] - s` *repo_uri_or_path* `[--no-catalog] [--no-index]`
放弃在系统信息库中找到的所有目录、搜索以及其他缓存信息，然后根据系统信息库的当前内容重新创建这些信息。

使用 `-p` 时，仅对给定发布者执行操作。如果没有提供该选项或者指定了特定值 `all`，则对所有发布者执行操作。可以多次指定此选项。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

使用 `--no-catalog` 时，不重新生成软件包数据。

使用 `--no-index` 时，不重新生成搜索索引。

`refresh [-p` *publisher* `...] - s` *repo_uri_or_path* `[--no-catalog] [--no-index]`
将在系统信息库中找到的所有新软件包编入目录并更新所有搜索索引。这主要供延迟的发布使用（`pkgsend` 的 `--no-catalog` 或 `--no-index` 选项）。

使用 `-p` 时，仅对给定发布者执行操作。如果没有提供该选项或者指定了特定值 `all`，则对所有发布者执行操作。可以多次指定此选项。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

使用 `--no-catalog` 时，不添加任何新的软件包。

使用 `--no-index` 时，不更新搜索索引。

`remove [-n] [-p` *publisher* `...] -s` *repo_uri_or_path pkg_fmri_pattern* `...`
从系统信息库中删除与指定模式匹配的软件包，其中包括这些软件包引用的且其他任何软件包没有使用的所有文件。

注 – 删除关联发布者的所有搜索索引数据。

该子命令仅可以与基于文件系统的系统信息库一起使用。

注意 – 此操作不可逆并且不得在其他客户机正在访问系统信息库时使用，因为这样会使得它们在执行检索操作期间出现故障。

使用 `-n` 时，会试运行操作而不进行软件包更改。在退出之前，会显示要删除的软件包的列表。

使用 `-p` 时，仅删除给定发布者的匹配软件包。如果没有提供该选项，会删除所有发布者的所有匹配软件包。可以多次指定此选项。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

`set [-p` *publisher*`] - s` *repo_uri_or_path section/property* `=[`*value*`] ... or` *section/property* `=([`*value*`]) ...`

为系统信息库或发布者设置指定属性的值。

该子命令仅可以与基于文件系统的系统信息库一起使用。

使用 `-p` 时，仅设置给定发布者的属性数据。如果发布者尚未存在，将添加该发布者。特殊值 `all` 可用于设置所有发布者的属性。

使用 `-s` 时，将对位于给定 URI 或文件系统路径的系统信息库进行操作。

可以使用以下格式之一指定属性和值：

section/*property*=	清除属性值。
section/*property*= *value*	将属性值替换为给定值。
section/*property*=(*value1 value2 valueN*)	将属性值替换为值列表。

对于系统信息库第 3 和 4 版，可以为系统信息库设置以下属性：

`publisher/prefix`	代表缺省发布者名称的字符串。第一个字符必须是 a-z、A-Z 或 0-9。该字符串的剩余部分只能包含字符 0-9、-、.、a-z 以及 A-Z。该值指明存在多个发布者的软件包时或有多个软件包已发布到系统信息库但未指定发布者时应当使用的发布者。

对于系统信息库第 3 和 4 版，可以为系统信息库中的各发布者设置以下属性：

`publisher/alias`	字符串，代表在使用系统信息库的配置数据添加发布者时客户机应当使用的缺省别名。第一个字符必须是 a-z、A-Z 或 0-9。该字符串的剩余部分只能包含字符 0-9、-、.、a-z 以及 A-Z。
`repository/collection_type`	可以使用值 `core` 或 `supplemental`，以表明此系统信息库中提供的软件包类型。 `core` 类型表明系统信息库包含该库中的软件包所声明的所有依赖项。`core` 类型主要用于操作系统的系统信息库。

supplemental 类型表明系统信息库包含依赖于另一个系统信息库中的软件包或要与另一个系统信息库中软件包一起使用的软件包。

repository/description	纯文本段落，描述系统信息库的用途和内容。
repository/detailed_url	URI，代表提供更多有关系统信息库信息的文档的位置（例如网页）。
repository/legal_uris	文档的位置列表 (URI)，提供关于系统信息库的其他法律信息。
repository/mirrors	系统信息库的位置列表 (URI)，这些系统信息库包含系统信息库的软件包内容的副本但不包含软件包元数据。
repository/name	纯文本字符串，包含系统信息库的名称。
repository/origins	系统信息库的位置列表 (URI)，这些系统信息库包含该系统信息库的软件包元数据和内容的完整副本。
repository/refresh_seconds	整数值，表示客户机在每次更新检查之后和检查系统信息库以查找更新的软件包数据之前应当等待的秒数。
repository/registration_uri	代表资源位置的 URI，必须使用该位置才能获取访问系统信息库的证书。注册网页就是一个示例。
repository/related_uris	系统信息库的位置列表 (URI)，这些系统信息库包含用户可能感兴趣的软件包。

此处没有记录但列在 get 子命令输出中的属性保留供内部使用，不得对其进行设置。

version
显示标识 pkg(5) 系统版本的唯一字符串。由 version 操作生成的值不能进行排序，并且对于在不平等情况下的比较而言是不安全的。

示例

示例 1 创建软件包系统信息库

```
$ pkgrepo create /my/repository
```

示例 2 显示信息

显示发布者摘要以及系统信息库中软件包的数量。

```
$ pkgrepo info -s /my/repository
PUBLISHER   PACKAGES STATUS UPDATED
example.com 5        online 2011-07-22T18:09:09.769106Z
```

示例 2 显示信息 （续）

```
$ pkgrepo info -s http://pkg.oracle.com/solaris/release/
PUBLISHER PACKAGES STATUS UPDATED
solaris   3941     online 2010-11-12T19:24:25.967246Z
```

示例 3 重新生成目录和搜索数据

重新生成系统信息库的目录和搜索数据。

```
$ pkgrepo rebuild -s /my/repository
```

示例 4 刷新目录和搜索数据

刷新系统信息库的目录和搜索数据。

```
$ pkgrepo refresh -s /my/repository
$ pkgrepo refresh -s http://example.com/repository
```

示例 5 显示所有系统信息库属性

```
$ pkgrepo get -s /my/repository
SECTION    PROPERTY VALUE
publisher  prefix   ""
repository version  4
$ pkgrepo get -s http://pkg.oracle.com/solaris/release/
SECTION    PROPERTY VALUE
publisher  prefix   solaris
repository version  4
```

示例 6 显示所有发布者属性

```
$ pkgrepo get -s http://pkg.oracle.com/solaris/release/ -p all
PUBLISHER SECTION    PROPERTY         VALUE
solaris   publisher  alias
solaris   publisher  prefix           solaris
solaris   repository collection-type  core
solaris   repository description      This\ repository\ serves\ the\ Oracle\
Solaris\ 11\ Package\ repository.
solaris   repository legal-uris       ()
solaris   repository mirrors          (http://pkg-cdn1.oracle.com/solaris.release/)
solaris   repository name             Oracle\ Solaris\ 11\ Package\ Repository
solaris   repository origins          ()
solaris   repository refresh-seconds
solaris   repository registration-uri ""
solaris   repository related-uris     ()
```

示例 7 设置缺省发布者

```
$ pkgrepo set -s /my/repository publisher/prefix=example.com
```

示例 8 设置发布者属性

```
$ pkgrepo set -s /my/repository -p example.com \
repository/origins=http://example.com/repository
```

示例 9 将新的发布者添加到系统信息库中

```
$ pkgrepo add-publisher -s /my/repository example.com
```

退出状态

将返回以下退出值：

0 命令成功。

1 出现错误。

2 指定的命令行选项无效。

3 请求了多项操作，但只有一部分操作成功。

4 没有进行更改时，无需执行任何操作。

99 发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkg(1)、pkgrecv(1)、pkgsend(1)、pkg.depotd(1m)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pkgsend – 映像包管理系统发布客户机

用法概要

```
/usr/bin/pkgsend [options] command [cmd_options] [operands]

/usr/bin/pkgsend generate [-T pattern] [--target file]
    source ...

/usr/bin/pkgsend publish [-b bundle ...] [-d source ...]
    [-s repo_uri_or_path] [-T pattern] [--no-catalog]
    [manifest ...]
```

描述

通过 pkgsend，可使用软件包清单将新软件包和新软件包版本发布到映像包管理系统信息库。要创建或管理系统信息库，请参见 pkgrepo(1)。要从现有系统信息库的软件包中创建软件包归档文件，请参见 pkgrecv(1)。有关软件包清单的更多信息，请参见 pkg(5)。

完成 pkgsend 操作后，在系统信息库上运行 pkgrepo refresh 或 pkgrepo rebuild 以构建搜索索引。

选项

支持以下选项：

--help 或 -?　显示用法消息。

子命令

支持以下子命令：

generate [-T *pattern*] [--target *file*] *source* ...

读取每个 *source*（例如，SVR4 软件包、目录或 tar 文件）并将介绍 *source* 的清单发送到 stdout。在输出清单中，file 和 dir 操作将所有者设置为 root，并将组设置为 bin。

然后可以注释输出清单，使用 pkgdepend(1) 添加或分析依赖项，并在将其传递到 publish 子命令之前使用 pkglint(1) 验证其正确性。

以下是支持的源：

- 文件系统格式 SVR4 软件包
- 数据流格式 SVR4 软件包
- tar 文件
- 目录

如果源中文件的基本名称与使用 -T 指定的模式匹配，则将文件的时间戳添加到该文件的操作中。*pattern* 使用 shell 匹配规则：

*	匹配所有内容。
?	匹配任何单个字符。
[*seq*]	匹配 *seq* 中的任何字符。
![*seq*]	匹配不在 *seq* 中的任何字符。

如果指定源是一个目录，当单个 inode 具有多个路径名称时，则没有明确的方式区分 `file` 操作与 `hardlink` 操作。通常，在文件系统遍历中发现的第一个视为文件，其余的视为硬链接。这可以是任意的，具体取决于文件系统的实现方式。要指定哪些路径名称应视为文件，请将每个路径名称作为参数传递到 `--target` 选项。该选项不会影响其他类型的源，因为它们可以表明哪些路径名称是文件，哪些是硬链接。

当提供 SVR4 软件包作为源时，`pkgsend` 会确认不存在具有类操作脚本的文件，以及不存在安装前、安装后、删除前或删除后脚本。但与 `manifest` 类一起安装的任何 SMF 清单除外。将从所有可重定位路径中删除 `BASEDIR` 。

SVR4 `DESC` 参数将转换为 `pkg.description` 值。SVR4 `NAME` 参数将转换为 `pkg.summary` 值。

`publish [-b` *bundle* `...] [-d` *source* `...] [-s` *repo_uri_or_path*`] [-T` *pattern*`] [--no-catalog] [`*manifest* `...]`

将使用指定软件包清单的软件包发布到目标软件包系统信息库，并从提供的源中检索该软件包的文件。如果指定了多个清单，它们将以提供的顺序联接。如果未指定清单，则从 `stdin` 中读取清单。

使用 `-b` 时，将指定的包添加到源列表，以便在清单中查找文件时进行搜索。包是 tar 文件和 SVR4 软件包等源。如果多次指定该选项，则以在命令行显示的顺序对源进行搜索。如果同时指定 `- b` 和 `-d`，则首先搜索 `-d` 源。有关支持的包及其使用方式的说明，请参阅以上的 `generate` 子命令。

使用 `-d` 时，将指定的目录添加到源列表，以便在清单中查找文件时进行搜索。如果多次指定该选项，则以在命令行显示的顺序对源进行搜索。有关支持的源及其使用方式的说明，请参阅以上的 `generate` 子命令。

使用 `-s` 时，将软件包发布到位于给定 URI 或文件系统路径的系统信息库。有关发布限制和建议的更多信息，请参见下面的“附注”部分。另请参见“环境变量”部分。

使用 `--no-catalog` 时，不将软件包添加到发布者的目录。当一次发布多个软件包，并且必须连续执行发布者目录更新时，建议使用该选项。完成发布后，可使用 `pkgrepo` (1) 的 `refresh` 子命令将新软件包添加到相应的发布者目录。

有关所有其他选项的用法及其影响，请参阅以上的 `generate` 子命令。

环境变量

`PKG_REPO` 目标系统信息库的路径或 URI。

示例

示例 1 生成并发布软件包

使用 `pkgsend generate` 创建软件包并将其发布。

```
$ pkgsend generate /path/to/proto > /path/to/manifests/foo.p5m
```

将 `example.com` 发布者的软件包 FMRI 添加到 `foo.p5m` 的开头。

```
set name=pkg.fmri value=pkg://example.com/foo@1.0
```

示例 1 生成并发布软件包 （续）

结果清单应类似于以下内容：

```
set name=pkg.fmri value=pkg://example.com/foo@1.0
dir group=sys mode=0755 owner=root path=usr
dir group=bin mode=0755 owner=root path=usr/bin
file usr/bin/foo group=bin mode=0555 owner=root path=usr/bin/foo
```

```
$ pkgsend publish -s http://example.com:10000 -d /path/to/proto \
/path/to/manifests/foo.p5m
```

示例 2 创建和发布普通软件包

为包含以下行的发布者 example.com 创建清单：

```
set name=pkg.fmri value=pkg://example.com/foo@1.0-1
file /exdir/foo mode=0555 owner=root group=bin path=/usr/bin/foo
```

发布软件包：

```
$ pkgsend publish -s http://example.com:10000 -d /exdir
```

示例 3 使用已经存在的清单

使用基于文件系统的发布和已经存在的清单发布软件包。

```
$ pkgsend publish -s /tmp/example_repo -d /tmp/pkg_files \
/tmp/pkg_manifest
```

退出状态

将返回以下退出值：

0 命令成功。

1 出现错误。

2 指定的命令行选项无效。

99 发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkgdepend(1)、pkgrepo(1)、pkg.depotd(1m)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

附注

由于发布协议限制，当发布大小超过 128 MB 的单个软件包文件时，必须使用基于文件系统的发布。当需要系统信息库的访问控制时，也建议使用基于文件系统的发布。

当使用基于文件系统的发布时，在完成发布后必须重新启动提供目标系统信息库服务的任何 `pkg.depotd` 进程，以便在其 Web 界面或搜索响应中反映更改。有关更多信息，请参见 `pkg.depotd`(1M)。

引用名

pkgsign – 映像包管理系统签名实用程序

用法概要

```
/usr/bin/pkgsign [-a hash_algorithm]
    [-c path_to_signing_certificate]
    [-i path_to_intermediate_cert] ...
    [-k path_to_private_key] [-n] -s path_or_uri
    [--help] [--no-index] [--no-catalog]
    (fmri|pattern) ...
```

描述

通过使用提供的密钥和证书添加签名操作，pkgsign 可在系统信息库中更新给定 FMRI 的清单。修改后的软件包保留原来的时间戳。

选项

支持以下选项：

使用 -a 时，将使用签名算法 *hash_algorithm* 而不是缺省值。缺省签名算法为 rsa-sha256。支持的签名算法包括 rsa-sha256、rsa-sha384、rsa-sha512、sha256、sha384 和 sha512。仅指定散列算法的签名算法会导致签名值为软件包清单的散列。指定 rsa 和散列算法的签名算法会导致签名值为使用提供的私钥签名的清单的散列（请参见 -c 和 - k 选项）。

使用 -c 时，将添加证书 *path_to_signing_certificate* 作为验证操作中的签名值时所使用的证书。-c 选项仅可以与 -k 选项一起使用。

使用 -i 时，将添加证书 *path_to_intermediate_cert* 作为验证证书 *path_to_signing_certificate*（作为参数提供给 -c）时所使用的证书。通过多次指定 -i，可提供多个证书。

使用 -k 时，将使用存储在 *path_to_private_key* 中的私钥对清单进行签名。-k 选项仅可以与 -c 选项一起使用。如果未设置 -k，则签名值为清单的散列。

使用 -n 时，将执行试运行，它不会以任何方式更改系统信息库。

使用 -s 时，将对位于 *path_or_uri* 的系统信息库中的软件包进行签名。

使用 --help 时，将显示用法消息。

使用 --no-index 时，在重新发布已签名的清单后不更新系统信息库搜索索引。

使用 --no-catalog 时，将在重新发布已签名的清单后不更新系统信息库目录。

示例

示例 1 使用清单的散列值进行签名

使用清单的散列值对发布到 http://localhost:10000 的软件包进行签名。这通常用于测试。

```
$ pkgsign -s http://localhost:10000 -a sha256 \
example_pkg@1.0,5.11-0:20100626T030108Z
```

示例2 使用密钥和证书进行签名

使用 rsa-sha384 对发布到位于 /foo/bar 的文件系统信息库的软件包进行签名，以便对清单执行散列和签名操作。签名密钥位于 /key/usr2.key 中，其关联的证书位于 /key/usr2.cert 中，而用于验证证书的证书位于 /icerts/usr1.cert 中。

```
$ pkgsign -s file:///foo/bar/ -a rsa-sha384 \
-k /key/usr2.key -c /key/usr2.cert -i /icerts/usr1.cert \
example_pkg@1.0,5.11-0:20100626T031341Z
```

退出状态

将返回以下退出值：

0 命令成功。

1 出现错误。

2 指定的命令行选项无效。

3 请求了多项操作，但只有一部分操作成功。

99 发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkg(1)、pkgrecv(1)、pkgsend(1)、pkgrepo(1)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

引用名

pm-updatemanager – 用于更新软件包的应用程序

用法概要

```
/usr/bin/pm-updatemanager [options]

/usr/bin/pm-updatemanager [-hdR] [--help] [--debug]
    [--image-dir dir]
```

描述

`pm-updatemanager` 检查系统上安装的软件包是否存在可用更新并安装这些更新。

注 – 如果 `package/pkg`、`package/pkg/package-manager` 或 `package/pkg/update-manager` 软件包需要更新，则 `pm-updatemanager` 首先更新这些软件包，然后重新启动以完成其余所有更新。

选项

支持以下选项：

`-h` 或 `--help`	显示用法消息。
`-d` 或 `--debug`	在调试模式下运行 `pm-updatemanager`。
`-R` 或 `--image-dir` *dir*	对根目录为 *dir* 的映像（而不是自动搜索到的映像）执行操作。

示例

示例 1 更新当前映像

对当前映像调用 `pm-updatemanager`。这会检查当前映像中安装的软件包的所有可用更新并安装这些更新。

```
$ /usr/lib/pm-launch pm-updatemanager
```

这是桌面菜单选项“系统”>“管理”> "Update Manager" 调用的同一命令。

示例 2 更新指定映像

对存储在 `/aux0/example_root` 中的映像调用 `pm-updatemanager`。

```
$ /usr/lib/pm-launch pm-updatemanager -R /aux0/example_root
```

退出状态

将返回以下退出值：

`0` 一切正常工作。

`1` 出现错误。

`2` 指定的命令行选项无效。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	`package/pkg/update-manager`
接口稳定性	Uncommitted（未确定）

另请参见 packagemanager(1)、pkg(1)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

附注 对您不具有的映像执行操作时，需要具有足够的特权才能调用 pm-updatemanager。在这些情况下，通常使用 /usr/lib/pm-launch 调用 pm-updatemanager 。

参 考 文 档

系统管理命令

引用名

pkg.depotd – 映像包管理系统 depot 服务器

用法概要

```
/usr/lib/pkg.depotd [-a address] [-d inst_root] [-p port]
    [-s threads] [-t socket_timeout] [--add-content]
    [--cfg] [--content-root] [--debug feature_list]
    [--disable-ops=op[/1][,...]]
    [--log-access] [--log-errors] [--mirror]
    [--proxy-base url] [--readonly] [--rebuild]
    [--ssl-cert-file] [--ssl-dialog] [--ssl-key-file]
    [--writable-root]
```

描述

`pkg.depotd` 是映像包管理系统的 depot 服务器。它提供对软件包系统信息库中数据的网络访问。对于不支持通过文件系统直接访问系统信息库的客户机，或网络访问是唯一可用的或首选的传输方式的客户机，通常使用软件包库 (depot)。

有些客户机（如检索客户机）可以通过 `pkg`(1) 直接从系统信息库或通过 depot 服务器从系统信息库检索软件包和软件包元数据的列表。发布客户机可以通过 `pkgsend`(1) 将软件包的新版本直接发送到系统信息库或通过 depot 服务器发送到系统信息库。`pkgrepo`(1) 可用于创建供 depot 服务器使用的系统信息库，或用于直接管理系统信息库以及通过 depot 服务器管理系统信息库。

`pkg.depotd` 通常作为服务在系统上运行。软件包和软件开发者可能希望运行专用副本进行测试。

库 (depot) 自身未提供任何访问控制方法。缺省情况下，所有能够连接 depot 服务器的客户机都能够读取所有软件包数据和发布新的软件包版本。但在服务管理工具 (Service Management Facility, SMF) 下运行 depot 服务器时除外，在此情况下缺省设置为以只读模式运行。下面的“附注”部分描述了维护内容不断变化的公共 depot 服务器的一些最佳做法。

Smf 属性

`pkg.depot` 服务器通常通过与其服务关联的 `smf`(5) 属性来配置。可以识别以下属性：

`pkg/address`
: (`net_address`) 要在其上侦听连接的 IP 地址。缺省值为 0.0.0.0 (`INADDR_ANY`)，表示侦听所有活动接口。要侦听所有活动的 IPv6 接口，请使用 `::`。只使用第一个值。

`pkg/content_root`
: (`astring`) 实例应在其中查找静态 Web 内容以及其他 Web 内容的文件系统路径。缺省值为 `/usr/share/lib/pkg`。

`pkg/debug`
: (`astring`) 以逗号分隔的要启用的调试功能列表。可能的值为：

  `headers` 将每个请求的标头记录到错误日志。

`pkg/disable_ops`
: (`astring`) 以逗号分隔的应在 depot 服务器中禁用的操作列表。操作以下面的形式提供：*operation*[/*version*]（例如 `catalog` 或 `search_1`）。

pkg/image_root	(astring）其文件信息将用作文件数据高速缓存的映像的路径。
pkg/inst_root	(astring) 实例应在其中查找系统信息库数据的文件系统路径。除非已提供 file_root 或 PKG_REPO，否则必须指定该路径。缺省值为 /var/pkgrepo。
pkg/log_access	(astring) depot 进程记录任何与访问相关的信息的目标位置。可能的值为：stderr、stdout、none 或绝对路径名。如果 stdout 是 tty，则缺省值为 stdout。如果 stdout 不是 tty，则缺省值为 none。
pkg/log_errors	(astring) depot 进程记录任何错误或其他信息的目标位置。可能的值为：stderr、stdout、none 或绝对路径名。缺省值为 stderr。
pkg/mirror	(boolean) 设置是否使用软件包镜像模式。为 true 时，禁用发布和元数据操作，只提供操作受限制的浏览器用户界面。pkg/readonly 属性为 true 时，该属性不能为 true。缺省值为 false。
pkg/port	(count) 实例侦听传入软件包请求应使用的端口号。如果未提供 SSL 证书和密钥信息，则缺省值为 80；否则，缺省值为 443。
pkg/proxy_base	(uri) 该属性更改 depot 服务器的基 URL，在采用反向代理配置的 Apache 或其他 Web 服务器上运行 depot 服务器时最有用。
pkg/readonly	(boolean) 设置是否禁用修改操作，如 pkgsend(1) 启动的那些操作。检索操作仍可用。pkg/mirror 属性为 true 时，该属性不能为 true。缺省值为 true。
pkg/socket_timeout	(count) 服务器在断开连接之前应该等待客户机响应的最大秒数。缺省值为 60。
pkg/sort_file_max_size	(count) 索引器排序文件的最大大小。用于限制库 (depot) 创建索引可使用的 RAM 量，也可增大该大小以提高速度。
pkg/ssl_cert_file	(astring) 以 PEM 编码的证书文件的绝对路径名。缺省值为 none。此属性必须与 ssl_key_file 一起使用。如果同时提供了 ssl_cert_file 和 /ssl_key_file，则库 (depot) 只响应 SSL 请求。

`pkg/ssl_dialog` (astring) 指定应使用何种方法来获取用于 `ssl_key_file` 解密的口令短语。可能的值为：

`builtin`	提示输入口令短语。这是缺省值。
`exec:`*/path/to/program*	执行指定的外部程序来获取口令短语。程序的第一个参数为 `''`，这是保留字。程序的第二个参数是服务器的端口号。口令短语输出到 `stdout`。
`smf:fmri`	尝试从与 FMRI 相关的服务实例中检索属性 `pkg_secure/ssl_key_passphrase` 的值。

`pkg/ssl_key_file` (astring) 以 PEM 编码的私钥文件的绝对路径名。此属性必须与 `ssl_cert_file` 属性一起使用。如果同时提供了 `/ssl_key_file` 和 `ssl_cert_file`，则库 (depot) 只响应 SSL 请求。

`pkg/threads` (count) 将启动多少个线程为请求提供服务。缺省值为 60。只适用于小型部署。该值应该为并发客户机数量的 20 倍左右。`threads` 的最大值为 5000。

`pkg/writable_root` (astring) 程序对其具有写入访问权限的目录的文件系统路径。此属性可与 `-readonly` 选项一起使用，以便 depot 服务器无需具有对软件包信息的写入访问权限即可创建文件（如搜索索引）。

`pkg_secure/ssl_key_passphrase` (astring) 用于解密 `pkg/ssl_key_file` 的口令。该值受读取授权保护（使用属性 `solaris.smf.read.pkg-server` 实现）。

depot 服务器的浏览器用户界面 (Browser User Interface, BUI) 的显示和行为由以下属性控制：

`pkg_bui/feed_description` (astring) RSS/Atom 源的描述性段落。

`pkg_bui/feed_icon` (astring) 用于以可视方式表示 RSS/Atom 源的小型图像的路径名。路径名应相对于 `content_root`。缺省值为 `web/_themes/pkg-block-icon.png`。

`pkg_bui/feed_logo`	(`astring`) 将用于以可视方式标记或标识 RSS/Atom 源的大型图像的路径名。该值应相对于 `content_root`。缺省值为 `web/_themes/pkg-block-icon.png`。
`pkg_bui/feed_name`	(`astring`) 由为系统信息库提供服务的库 (depot) 生成的 RSS/Atom 源的简短描述性名称。缺省值为 "package repository feed"。
`pkg_bui/feed_window`	(`count`) 上次生成系统信息库源之前经过的小时数，包括生成源的时间。

软件包库 (depot) 还可以用作 `pkg`(5) 中本地客户机映像的镜像服务器。这使得在 LAN 上共享一个子网的客户机可以对其文件高速缓存进行镜像。客户机可以相互下载文件，从而减少软件包 depot 服务器上的负载。此功能可作为 `smf`(5) 配置的备用 depot 服务提供。它使用 mDNS 和 `dns-sd` 完成服务发现。

mDNS 镜像通常通过与其服务关联的 `smf`(5) 属性来配置。可以识别以下属性：

`pkg/image_root`	(`astring`）其文件信息将用作文件数据高速缓存的映像的路径。缺省值为 `/`。
`pkg/port`	(`count`) 实例侦听传入软件包请求应使用的端口号。缺省值为 80。

选项

`pkg.depotd` 可从文件或从现有 `smf`(5) 服务实例的属性数据读取其基本配置信息。

`--cfg` *source*	读取和写入配置数据时使用的文件的路径名，或格式为 `smf:`*fmri* 的字符串，其中 *fmri* 是从中读取配置数据的实例的服务故障管理资源标识符 (fault management resource identifier, FMRI)。有关指定文件格式的详细信息，请参见下面的“库 (depot) 配置”。

如果没有已存在的配置源可用，或者要覆盖从使用 `--cfg` 提供的配置文件中读取的值，则可以使用下列选项来更改 depot 服务器的缺省行为：

`-a address`	请参见上面的 `pkg/address`。
`--content-root` *root_dir*	请参见上面的 `pkg/content_root`。
`-d` *inst_root*	请参见上面的 `pkg/inst_root`。
`--debug` *features*	请参见上面的 `pkg/debug`。
`--disable-ops` *op_list*	请参见上面的 `pkg/disable_ops`。
`--image-root` *path*	请参见上面的 `pkg/image_root`。
`--log-access` *dest*	请参见上面的 `pkg/log_access`。
`--log-errors` *dest*	请参见上面的 `pkg/log_errors`。
`--mirror`	请参见上面的 `pkg/mirror`。
`-p` *port*	请参见上面的 `pkg/port`。

`--proxy-base` *url*	请参见上面的 `pkg/proxy_base`。如果提供空值，则忽略该选项。
`--readonly`	请参见上面的 `pkg/readonly`。
`-s` *threads*	请参见上面的 `pkg/threads`。
`-s-ort-file-max-size` *bytes*	请参见上面的 `pkg/sort_file_max_size`。
`--ssl-cert-file` *source*	请参见上面的 `pkg/ssl_cert_file`。
`--ssl-dialog` *type*	请参见上面的 `pkg/ssl_dialog`。
`--ssl-key-file` *source*	请参见上面的 `pkg/ssl_key_file`。
`-t` *socket_timeout*	请参见上面的 `pkg/socket_timeout`。
`--writable-root` *path*	请参见上面的 `pkg/writable_root`。

`pkgrepo`(1) 提供了软件包系统信息库的其他管理功能。

库 (depot) 配置

使用 `--cfg` 选项（而非 `smf`(5) FMRI）提供配置文件时，depot 服务器以简单文本格式读取和写入所有配置数据。在上面的“SMF 属性”中介绍了配置数据。配置数据包含多个部分，以 [*section*] 标头开始，后跟 `name` = *value* 条目。后续部分采用 RFC 822 样式。可以将数据拆分到多个行中，以空格开始后续行即可。

必须使用上面的“选项”中列出的选项提供配置文件中未提供的所有必需值。样例配置文件可能与下面的类似：

```
[pkg]
port = 80
inst_root = /export/repo

[pub_example_com]
feed_description = example.com's software
  update log
```

示例

示例 1 启用 depot 服务器

```
# svcadm enable application/pkg/server
```

示例 2 更改服务器的侦听端口。

```
# svccfg -s application/pkg/server setprop pkg/port = 10000
# svcadm refresh application/pkg/server
# svcadm restart application/pkg/server
```

示例 3 启用镜像

```
# svcadm enable application/pkg/dynamic-mirror
```

环境变量

`PKG_REPO`	指定要提供的系统信息库所在的目录。如果指定 `-d`，将忽略该值。
`PKG_DEPOT_CONTENT`	指定库 (depot) 提供的静态内容所在的目录。该目录应包含下面介绍的“文件”下的文件，虽然其中的内容可能与提供的缺省内容不同。

退出状态

将返回以下退出值：

`0` 操作成功。

`1` 出现错误。

`2` 指定的命令行选项无效。

`99` 发生了意外的异常。

文件

`/usr/share/lib/pkg`	缺省显示内容的位置。修改 `pkg/content_root` 以选择其他位置。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	`package/pkg`
接口稳定性	Uncommitted（未确定）

另请参见

`dns-sd(1M)`、`mdnsd(1M)`、`pkg(1)`、`pkgrepo(1)`、`pkgsend(1)`、`syslogd(1M)`、`smf(5)`

`http://hub.opensolaris.org/bin/view/Project+pkg/`

附注

`pkd.depotd` 服务由 `svc:/application/pkg/server` 服务标识符下的 SMF 管理。

mDNS 镜像服务由 `svc:/application/pkg/dynamic-mirror` 服务标识符下的 SMF 管理。

要控制对库 (depot) 的读取访问权限，可以将 HTTP 反向代理与授权方法结合使用，如 `pkg(1)` 本身支持的基于客户机的 SSL 证书访问权限。

更改配置或使用基于文件系统的操作更改软件包数据后，需要重新启动 depot 服务器进程，以便更改能够反映在操作和输出中。使用下列方法之一来重新启动 depot 服务器进程：

- 使用 `svcadm(1M)` 重新启动 `application/pkg/server` 实例。
- 使用 `kill(1)` 将 `SIGUSR1` 信号发送到 depot 服务器进程。这将执行正常的重新启动，使进程保持原样，但重新加载所有配置、软件包和搜索数据：

  `# kill -USR1` *pid*

引用名

pkg.sysrepo – 映像包管理系统的系统信息库配置

用法概要

```
pkg.sysrepo -p port [-c cache_dir] [-s cache_size]
    [-w http_proxy] [-W https_proxy]
```

描述

`pkg.sysrepo` 用于生成映像包管理系统 (Image Packaging System, IPS) 系统信息库的配置文件。`pkg.sysrepo` 由 `svc:/application/pkg/system-repository` 服务管理工具 (Service Management Facility, SMF) 服务调用。要更改配置，应对 SMF 服务中的属性进行更改。

系统信息库负责通过集中代理提供对在参考映像中配置的软件包系统信息库的访问。配置为使用系统信息库的任何客户机可立即看到对该参考映像所做的发布者配置更改。

系统信息库主要用于全局区域，以允许非全局区域访问在全局区域中配置的系统信息库。SMF 服务 `svc:/application/pkg/zones-proxyd` 和 `svc:/application/pkg/zones-proxy-client` 负责非全局区域与全局区域之间的传输。只有 `pkg`(5) 使用该传输。

请注意，仅支持 http、https 和 v4 文件系统信息库。不支持基于 p5p 的文件系统信息库或更早的文件系统信息库格式。有关系统信息库版本的更多信息，请参见 `pkgrepo`(1)。

选项

支持以下选项：

`-c` *cache_dir*
: 系统信息库缓存来自配置的发布者的响应所应使用的目录的绝对路径。

  缺省情况下，使用文件高速缓存。但是，可以使用特殊值 `memory` 来表示应使用内存高速缓存。特殊值 `None` 可用于表示系统信息库不应执行任何缓存。应使用 `config/cache_dir` SMF 属性配置该设置。

`-p` *port*
: 系统信息库侦听请求所应使用的端口。应使用 `config/port` SMF 属性配置该设置。

`-s` *cache_size*
: 定义系统信息库的最大高速缓存大小的整数值 (MB)。应使用 `config/cache_max` SMF 属性配置该设置。

`-w` *http_proxy*
: 采用以下格式的字符串：*scheme*`://` *hostname*`[:`*port*`]`。该字符串定义了系统信息库访问基于 http 的软件包系统信息库可使用的 Web 代理服务器。可以使用 `config/http_proxy` SMF 属性配置此设置。

`-W` *https_proxy*
: 采用以下格式的字符串：*scheme*`://` *hostname*`[:`*port*`]`。该字符串定义了系统信息库访问基于 https 的软件包系统信息库可使用的 Web 代理服务器。可以使用 `config/https_proxy` SMF 属性配置此设置。

示例

示例 1 启用系统信息库

```
$ svcadm enable svc:/application/pkg/system-repository
```

退出状态

将返回以下退出值：

0 命令成功。

1 命令无法写入有效配置。

2 指定的命令行选项无效。

99 发生了意外的异常。

属性

有关下列属性的说明，请参见 attributes(5)：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkg(1)、pkg.depotd(1m)、pkg(5)

http://hub.opensolaris.org/bin/view/Project+pkg/

参 考 文 档

标准、环境和宏

引用名

pkg – 映像包管理系统

描述

映像包管理系统 pkg(5) 是用于提供软件生命周期管理（安装、升级和删除）的框架。映像包管理以软件包为单位对软件进行管理，软件包是由一组关键字/值对以及可能的数据有效负荷定义的操作的集合。在许多情况下，操作是在文件系统中找到的文件，但它们也表示其他可安装对象，例如驱动程序、服务和用户。

软件包 Fmri 和版本

每个软件包均由带有机制 pkg: 的故障管理资源标识符 (fault management resource identifier, FMRI) 表示。软件包的完整 FMRI 由机制、发布者、软件包名称和以下格式的版本字符串组成：

```
pkg://solaris/system/library/c++-runtime@0.5.11,5.11-0.174.0.0.0.0.0:20110921T190358Z
```

solaris 是发布者。system/library/c++-runtime 是软件包名称。虽然名称空间是有层次的且可具有任意深度，但不存在强制包含；名称从本质上而言是任意的。发布者信息为可选项，但必须位于 pkg:// 之后（如果存在）。包括发布者的 FMRI 通常称为“全限定”FMRI。如果不存在发布者信息，则软件包名称通常应位于 pkg:/ 之后。

如果 FMRI 不包含发布者信息，打包客户端通常允许省略 FMRI 的机制。例如，pkg:/system/library/c++-runtime 可以写为 system/library/c++-runtime。如果省略了机制，客户端也允许省略除软件包名称的最后一个组成部分（用于匹配目的）之外的所有其他内容。例如，system/library/c++-runtime 可以写为 library/c++-runtime 或 c++-runtime，这将与名为 c++-runtime 的软件包或以 /c++-runtime 结尾的软件包名称匹配。

发布者名称将个人、个人组或组织标识为一个或多个软件包的源。为避免发布者名称冲突并有帮于标识发布者，最佳做法是使用代表发布软件包实体的域名作为发布者名称。

版本跟在软件包名称后面，由 @ 符号分隔。版本包含四个数字序列，由标点符号分隔。前三个序列中的元素由圆点分隔，各序列可具有任意长度。不允许版本的组成部分以零开头（例如 01.1 或 1.01）。允许以零结尾（例如 1.10）。

版本的第一个部分是组件版本。对于紧密绑定到操作系统上的组件，此序列通常是操作系统中该版本的 uname -r 值。对于具有自己的开发生命周期的组件，此序列是一个由小圆点分隔的发行编号，例如 2.4.10。

版本的第二个部分（如果存在）必须跟在逗号 (,) 后面，是内部版本。内部版本指定构建软件包内容的操作系统版本，提供了希望软件包内容可成功运行的最低操作系统版本。

版本的第三个部分（如果存在）必须跟在连字符 (-) 后面，是分支版本。分支版本是用于提供特定于供应商的信息的版本控制组件。分支版本可在打包元数据发生更改时增大，独立于组件版本。分支版本可能包含内部版本号或其他信息。

版本的第四个部分（如果存在）必须跟在冒号 (:) 后面，是一个时间戳。时间戳表示软件包的发布时间。

在版本之间进行比较时，不考虑完整版本的任何组件，除非左侧的组件与其相同。因此 "4.3-1" 大于 "4.2-7"，因为 "4.3" 大于 "4.2"；"4.3-3" 大于 "4.3-1"，因为 "3" 大于 "1"。

系统的许多部分（如果适用）会在显示 FMRI 时缩短 FMRI 并接受简短格式的输入，以减少显示的或所需的信息量。通常，可以省略机制、发布者、内部版本和时间戳。有时可以省略所有版本控制信息。

操作

操作表示系统上的可安装对象。在软件包的清单中对操作进行了描述。每个操作主要由其名称和一个关键属性组成。这些属性共同引用一个唯一的对象，与该对象一样遵循版本历史记录。操作可以具有其他属性。某些属性由包管理系统直接解释。其他属性可能仅对系统管理员或最终用户有用。

操作具有一种简单文本表示法：

action_name attribute1=value1 attribute2=value2 ...

属性名称中不能包含空格、引号或等号 (=)。第一个等号后的所有字符都属于值。值中可以包含所有这些符号，虽然空格必须括在单引号或双引号中。单引号处于括在双引号中的字符串内时不需要进行转义，双引号处于括在单引号中的字符串内时也不需要进行转义。可使用反斜杠 (\) 字符作为引号的前缀来避免终止带引号的字符串。反斜杠可使用反斜杠进行转义。

可以使用多个值对属性进行多次命名。这些值被视为无序列表。

具有多个属性的操作可以在清单文件中创建长行。可以通过使用反斜杠终止每个不完整的行来对此类行进行折行。请注意，属性/值对之间必须具有此接续字符。属性、属性值及其组合均不可分离。

下面列出的属性不是全部属性。事实上，任意属性都可附加到操作中，标准属性组易于扩大以合并将来的开发。

某些操作属性会导致在打包上下文外部执行其他操作。这些属性记录在下面的“执行器”部分中。

文件操作

`file` 操作表示普通文件。`file` 操作引用有效负荷，具有四个标准属性：

`path` 安装文件的文件系统路径。此属性是 `file` 操作的关键属性。

`mode` 文件的访问权限（采用数字格式）。这些只是简单权限，并非 ACL。

`owner` 拥有文件的用户的名称。

`group` 拥有文件的组的名称。

有效负荷是一个位置属性，因为它未命名。它是操作名称后面的第一个词。在发布的清单中，它是文件内容的 `SHA-1` 散列。如果存在于尚待发布的清单中，则它表示可以找到有效负荷的路径。请参见 `pkgsend`(1)。如果值包括一个等号，则可以使用散列属性代替位置属性。这两种属性可用于同一操作中。但是，散列必须完全相同。

其他属性包括：

preserve　此属性指定在升级时不应覆盖文件的内容（如果确定自文件安装或上次升级后其内容已发生了更改）。在初始安装时，如果找到现有文件，则挽救该文件（存储在 /var/pkg/lost+found 中）。

如果 preserve 的值是 renameold，则使用扩展名 .old 重命名现有文件，并将新文件放入相应位置。

如果 preserve 的值是 renamenew，则现有文件保持不变，并使用扩展名 .new 安装新文件。

如果 preserve 的值是 legacy，则在初始软件包安装时不会安装此文件。在升级时，会使用扩展名 .legacy 重命名任何现有文件，并在随后将新文件放入相应位置。

如果 preserve 的值是 true（或是上面未列出的值，例如 strawberry），则现有文件保持不变，而且也不安装新文件。

overlay　此属性指定操作是允许其他软件包在同一位置提供文件，还是提供用于覆盖其他文件的文件。此功能设计用于不参与任何自组装（例如 /etc/motd）且可安全覆盖的配置文件。

如果未指定 overlay，多个软件包将无法向同一位置提供文件。

如果 overlay 的值是 allow，则允许另外一个软件包向同一位置提供文件。除非也设置了 preserve 属性，否则此值没有效果。

如果 overlay 的值是 true，操作提供的文件将覆盖已指定 allow 的任何其他操作。基于覆盖文件的 preserve 属性值保留对已安装文件进行的更改。在删除时，如果仍要安装将覆盖的操作，则将保留文件的内容，无论是否指定了 preserve 属性。只能一个操作覆盖另一个操作，且 mode、owner 和 group 属性必须匹配。

也可以“体验”文件，而且文件可根据自身情况具有其他属性。对于 ELF 文件，可识别下列属性：

elfarch　ELF 文件的体系结构。此属性是 uname -p 查询文件所基于的体系结构后的输出。

elfbits　此属性为 32 或 64。

elfhash　此属性是文件中“相关”ELF 部分的散列。这些部分已在装入二进制文件时映射到内存中。在确定两个二进制文件的可执行行为是否将不同时，仅需要考虑这些部分。

original_name　此属性用于处理可编辑文件在软件包之间或在位置之间（或在这两者之间）的移动操作。此属性采用的格式为源软件包的名称后跟一个冒号和文件的原始路径。所删除的任何文件将使用其软件包和路径或 original_name 属性的值（如果指定）进行记录。所安装的已

设置 original_name 属性的任何可编辑文件将使用具有该名称的文件（如果它在同一打包操作中被删除）。

revert-tag　此属性用于标记应恢复为一个组的可编辑文件。可以指定多个 revert-tag 值。在使用所指定的任意标记调用 pkg revert 时，文件将恢复为其清单定义的状态。请参见 pkg(1)。

目录操作

dir 操作类似于 file 操作，因为它表示文件系统对象。但 dir 操作表示目录而不是普通文件。dir 操作具有与 file 操作相同的四个标准属性，path 是关键属性。

目录是 IPS 中包括的引用。当显式或隐式引用某目录的最新软件包不再引用该目录时，将删除该目录。如果该目录包含未打包的文件系统对象，则将这些项移动到 $IMAGE_META/lost+found 中。有关 $IMAGE_META 的更多信息，请参见“文件”部分。

要将未打包的内容移动到新的目录中，以下属性可能会有用：

salvage-from　此属性指定所挽救项的目录。具有此属性的目录在创建时可继承所挽救目录的内容（如果存在）。

链接操作

link 操作表示符号链接。link 操作具有以下标准属性：

path
安装符号链接的文件系统路径。此属性是 link 操作的关键属性。

target
符号链接的目标。链接将解析到的文件系统对象。

mediator
指定由给定中介组（例如 python）中涉及的所有路径名称共享的中介名称空间中的条目。可基于 mediator-version 和/或 mediator-implementation 执行链接中介。给定路径名称的所有中介链接必须指定同一中介者。但是，并非所有中介者版本和实现都需要在给定路径上提供链接。如果中介不提供链接，则会在选定该中介时删除链接。mediator 与特定版本和/或实现组合起来表示可选择供包管理系统使用的中介。

mediator-version
指定 mediator 属性描述的接口的版本（表示为非负整数的点分序列）。如果指定了 mediator 而未指定 mediator-implementation，则此属性是必需的。本地系统管理员可以显式设置要使用的版本。指定的值通常应与提供链接的软件包版本相匹配（例如，runtime/python-26 应使用 mediator-version=2.6），尽管这不是必需的。

mediator-implementation
指定除 mediator-version 之外还使用中介者实现，或使用中介者实现代替 mediator-version。不认为实现字符串应进行排序，如果系统管理员未显式指定，则由 pkg(5) 任意选择一个字符串。

该值可以是由字母数字字符和空格组成的任意长度的字符串。如果实现本身可被版本化或已被版本化，则应在字符串的结尾处在 @ 之后指定版本（表示为非负整数的点分序列）。如果存在实现的多个版本，则缺省行为是选择最高版本的实现。

如果系统上仅安装了特定路径的实现中介链接的一个实例，则会自动选择该实例。如果以后安装了该路径的其他链接，除非应用供应商、站点或本地覆盖或者如果某一链接进行了版本中介，否则不会切换链接。

`mediator-priority`
在解决中介链接中的冲突时，pkg(5) 通常会选择 `mediator-version` 值最大的链接，如果不可能，则会基于 `mediator-implementation` 进行选择。此属性用于为常规冲突解决方案过程指定覆盖。

如果未指定此属性，则会应用缺省中介者选择逻辑。

如果值为 `vendor`，则与未指定 `mediator-priority` 的链接相比，会优先选择该链接。

如果值为 `site`，则与值为 `vendor` 或未指定 `mediator-priority` 的链接相比，会优先选择该链接。

本地系统管理员可以覆盖上面所述的选择逻辑。

硬链接操作

`hardlink` 操作表示硬链接。它具有与 `link` 操作相同的属性，`path` 也是其关键属性。

驱动程序操作

`driver` 操作表示设备驱动程序。`driver` 操作不引用有效负荷。驱动程序文件自身必须作为 `file` 操作进行安装。可识别下列属性（有关更多信息，请参见 `add_drv`(1M)）：

`name`	驱动程序的名称。这通常是（但并不总是）二进制驱动程序文件的文件名。此属性是 `driver` 操作的关键属性。
`alias`	此属性表示驱动程序的别名。给定的驱动程序可以具有多个 `alias` 属性。无需任何特殊的引号规则。
`class`	此属性表示驱动程序类。给定的驱动程序可以具有多个 `class` 属性。
`perms`	此属性表示驱动程序的设备节点的文件系统权限。
`clone_perms`	此属性表示此驱动程序的克隆驱动程序次要节点的文件系统权限。
`policy`	此属性指定设备的其他安全策略。给定的驱动程序可以具有多个 `policy` 属性，但次要设备规范不可以存在于多个属性中。
`privs`	此属性指定驱动程序所用的特权。给定的驱动程序可以具有多个 `privs` 属性。
`devlink`	此属性指定 `/etc/devlink.tab` 中的条目。该值是定义了进入文件的确切行，带有由 `\\t` 表示的制表符。有关更多信息，请参见 `devlinks`(1M)。给定的驱动程序可以具有多个 `devlink` 属性。

依赖操作

depend 操作表示软件包间的依赖性。一个软件包可以依赖于另一个软件包，因为第一个软件包需要第二个软件包中的功能才能运行自身包含的功能或者甚至进行安装。依赖性可以是可选的。如果安装时未满足某个依赖性，包管理系统会尝试安装或更新依赖软件包至足够新的版本（受其他约束限制）。

可以识别下列属性：

`fmri` 表示依赖软件包的 FMRI。此属性是 `dependency` 操作的关键属性。`fmri` 值不得包括发布者。假定软件包名称是完整的。`require-any` 类型的依赖性可具有多个 `fmri` 属性。`fmri` 值中的版本是可选项，虽然对于某些类型的依赖性来说，不带版本的 `fmri` 没有任何意义。

`type` 依赖性的类型。

如果值为 `require`，则依赖性是必需的，且必须具有等于或大于 `fmri` 属性中指定版本的版本。如果未指定版本，则任何版本都满足依赖性。如果不能满足其任一必需依赖性，则无法安装软件包。

如果值为 `optional`，则依赖性（如果存在）必须处于指定的版本级别或更高级别。

如果值为 `exclude`，则当依赖性存在于指定的版本级别或更高级别时，无法安装包含软件包。如果未指定版本，则依赖软件包无法与指定依赖性的软件包同时安装。

如果值为 `incorporate`，则依赖性是可选的，但依赖软件包的版本会受到约束。请参见下面的“约束和冻结”。

如果值为 `require-any`，则多个 `fmri` 属性指定的多个依赖软件包中的任一个都可满足依赖性（遵循与 `require` 依赖性类型相同的规则）。

如果值为 `conditional`，则依赖性仅在系统上存在 `predicate` 属性定义的软件包时是必需的。

如果值为 `origin`，依赖性（如果存在）必须在要在安装之前修改的映像上具有指定值或更优值。如果 `root-image` 属性的值为 `true`，则依赖性必须存在于根目录为 / 的映像上，才能安装此软件包。

如果值为 `group`，则依赖性是必需的，除非软件包出现在映像避免列表上。请注意，过时软件包会无提示地满足组依赖性。请参见 `pkg`(1) 中的 `avoid` 子命令。

如果值为 `parent`，则在映像不是子映像时忽略依赖性。如果映像是子映像，则依赖性必须存在于父映像中。符合 `parent` 依赖性的软件包版本与用于 `incorporate` 依赖性的软件包版本相同。

`predicate` 表示 `conditional` 依赖性的谓词的 FMRI。

`root-image` 仅对 `origin` 依赖性有影响，如上所述。

许可证操作

`license` 操作表示许可证或其他与软件包内容相关联的信息文件。软件包可以通过 `license` 操作将许可证、免责声明或其他指南提供给软件包安装程序。

`license` 操作的有效负荷将提供到与软件包相关的映像元数据目录中，且应仅包含用户可读的文本数据。不应包含 HTML 或任何其他形式的标记。通过各属性，`license` 操作可以向客户端指示必须显示相关的有效负荷并/或要求接受它。显示并/或接受的方法由客户端决定。

可以识别下列属性：

`license` 此属性是 `license` 操作的关键属性。此属性为许可证提供有意义的描述，以帮助用户在无需阅读许可证文本本身的情况下确定内容。其中一些示例值包括：

- ABC Co. Copyright Notice
- ABC Co. Custom License
- Common Development and Distribution License 1.0 (CDDL)
- GNU General Public License 2.0 (GPL)
- GNU General Public License 2.0 (GPL) Only
- MIT License
- Mozilla Public License 1.1 (MPL)
- Simplified BSD License

`license` 值在软件包内必须唯一。建议在说明中包括许可证的版本，如上面的几个示例所示。如果软件包有对应多种不同许可证的代码，请使用多个 `license` 操作。许可证属性值的长度不得超过 64 个字符。

`must-accept` 如果为 `true`，则用户必须先接受此许可证，才能安装或更新相关软件包。省略此属性等效于 `false`。接受的方法（例如，交互式或基于配置）由客户端决定。

`must-display` 如果为 `true`，则在执行打包操作期间客户端必须显示操作的有效负荷。省略此值等效于 `false`。此属性不应用于版权声明，仅用于实际许可证或执行操作期间必须显示的其他材料。显示的方法由客户端决定。

传统操作

`legacy` 操作表示由传统包管理系统使用的软件包数据。与此操作相关联的属性将添加到传统系统的数据库中，以便查询这些数据库的工具可以像实际安装了传统软件包一样工作。需特别指出的是，这应足以使传统系统确信系统上已安装了 `pkg` 属性指定的软件包，如此便可使用软件包来满足依赖性。

可识别根据 `pkginfo`(4) 上的参数指定的下列属性：

`category` `CATEGORY` 参数的值。缺省值为 `system`。

desc	DESC 参数的值。
hotline	HOTLINE 参数的值。
name	NAME 参数的值。缺省值为 none provided。
pkg	要安装的软件包的缩写。缺省值为软件包的 FMRI 中的名称。此属性是 legacy 操作的关键属性。
vendor	VENDOR 参数的值。
version	VERSION 参数的值。缺省值为软件包的 FMRI 中的版本。

设置操作

set 操作表示软件包级别的属性或元数据，例如软件包描述。

可以识别下列属性：

name	属性的名称。
value	提供给属性的值。

set 操作可以提供软件包作者选择的任何元数据。但是，存在大量定义明确的对包管理系统具有特定意义的属性名称。

pkg.fmri	请参见“说明”部分中的“软件包 FMRI 和版本”。
info.classification	一个 pkg(5) 客户端可以使用一个或多个标记对软件包进行分类。该值应具有一个机制（例如 "org.opensolaris.category.2008" 或 "org.acm.class.1998"）和实际分类（例如 "Applications/Games"），由冒号 (:) 分隔。
pkg.description	软件包的内容和功能的详细描述，长度通常约为一个段落。
pkg.obsolete	如果为 true，则将软件包标记为过时。过时的软件包除了设置操作外不能具有任何其他操作，且不得标记为已重命名。
pkg.renamed	如果为 true，则软件包已被重命名。软件包中还必须存在一个或多个 depend 操作，且指向此软件包已重命名到的软件包版本。软件包不能同时标记为已重命名和过时，但在其他情况下可以具有任意多个设置操作。
pkg.summary	软件包的一行简短描述。

组操作

group 操作定义 UNIX 组，如 group(4) 中所定义。不存在对于组口令的支持。使用此操作定义的组最初不具有用户列表。可以使用 user 操作添加用户。可以识别下列属性：

groupname	组名的值。
gid	组的唯一数字 ID。缺省值为 100 之下的第一个自由组。

用户操作

user 操作定义 UNIX 用户，如 /etc/passwd、/etc/shadow、/etc/group 和 /etc/ftpd/ftpusers 文件中所定义。使用此属性定义的用户具有添加到相应文件中的条目。

可以识别下列属性：

username	用户的唯一名称
password	用户的加密口令。缺省值为 *LK*。请参见 shadow(4)。
uid	用户的唯一 UID。缺省值为 100 之下的第一个自由值。
group	用户的主组名称。必须可在 /etc/group 中找到。
gcos-field	/etc/passwd 中 gcos 字段的值。缺省值为 username。
home-dir	用户的起始目录。缺省值为 /。
login-shell	用户的缺省 shell。缺省值为空。
group-list	用户所属的辅助组。请参见 group(4)。
ftpuser	可设置为 true 或 false。缺省值 true 指示允许用户通过 FTP 登录。请参见 ftpusers(4)。
lastchg	1970 年 1 月 1 日至上次修改口令的日期之间的天数。缺省值为空。请参见 shadow(4)。
min	所需的相邻两次更改口令之间的最小天数。必须将此字段设置为 0 或更大值才能启用口令有效期。缺省值为空。请参见 shadow(4)。
max	口令的最大有效天数。缺省值为空。请参见 shadow(4)。
warn	用户在口令到期之前多少天收到警告。请参见 shadow(4)。
inactive	允许该用户不活动的天数。按每台计算机对此进行计数。可从计算机的 lastlog 文件获取有关上次登录的信息。请参见 shadow(4)。
expire	表示为自 UNIX 纪元（1970 年 1 月 1 日）后的天数的绝对日期。达到此数字时，将无法再进行登录。例如，到期值为 13514 指定登录将在 2007 年 1 月 1 日失效。请参见 shadow(4)。
flag	设置为空。请参见 shadow(4)。

执行器

在某些上下文中，附加操作可能适合在为特定操作做准备时执行或者在引入特定操作后执行。这些附加操作通常仅在实时系统映像上才需要，而且特定于操作系统。当软件包安装或删除过程中涉及的多个操作具有相同的执行器时，会为该安装或删除过程执行一次与执行器存在情况相对应的操作。

错误指定的执行器可能会导致软件包安装失败，如果该执行器无法确定进行安全安装的方法。

系统定义了以下执行器：

`reboot-needed`
可设置为 `true` 或 `false`。如果在软件包安装期间安装或更新某个操作（此执行器设置为 `true`），则可以将打包事务通告为需要重新引导。某些客户端实现可能会执行附加步骤，例如，在映像是实时系统映像的情况下，使用该映像的克隆执行整个软件包操作。

`disable_fmri`、`refresh_fmri`、`restart_fmri`、`suspend_fmri`
其中每个执行器在软件包安装或删除过程中均使用服务实例的 FMRI 值进行操作。根据 `svcadm`(1M) 的 `disable` 子命令，`disable_fmri` 会导致给定的 FMRI 在删除操作之前被禁用。根据 `svcadm`(1M) 的各个子命令，`refresh_fmri` 和 `restart_fmri` 会导致给定的 FMRI 在安装、更新或删除操作后被刷新或重新启动。最后，`suspend_fmri` 会导致给定的 FMRI 在安装操作阶段之前被临时禁用，并在该阶段完成后被重新启用。

该值可以包含与多个服务实例匹配的模式。但是，它必须使用 `svcs`(1) 所接受的 glob 显式执行此操作，而不是通过不指示任何实例来隐式执行此操作。

约束和冻结

在将软件包转换为新版本、添加到系统中或从系统中删除时，所选的版本或是否允许删除由对软件包施加的各种约束确定。这些约束可由其他软件包以依赖性的形式进行定义，或者由管理员以冻结的形式进行定义。

最常见的约束形式由 `require` 依赖性提供，如上面的“依赖操作”中所述。此类约束可防止软件包被降级或删除。

操作系统的大多数部分由名为 *incorporation* 的软件包进行封装。这些软件包主要提供由 `incorporate` 依赖性表示的约束。

如上所述，合并的软件包不需要存在于系统上，但如果存在，它将同时指定一个非独占最低版本和一个独占最高版本。例如，如果依赖 FMRI 具有版本 1.4.3，则低于 1.4.3 的版本不能满足依赖性，且任何高于或等于 1.4.4 的版本也不能满足依赖性。但是，可以允许以点分序列扩展的版本，如 1.4.3.7。

Incorporation 用于强制系统的各部分进行同步升级。对于某些组件（例如 C 库和内核），这是一项基本要求。对于其他组件（例如，不具有任何其他依赖性的简单用户级组件），同步升级仅仅用来提供一组经过测试的已知软件包版本，这些软件包版本可由 incorporation 的特定版本进行引用。

因为 incorporation 只是一个软件包，所以可将其删除，它提供的所有约束也将随之解除。但是，Oracle Solaris 提供的许多 incorporation 是合并软件包所必需的，因为解除其约束可能会不安全。

尝试将软件包升级到已安装的 incorporation 所不允许的版本，将不会尝试查找更高的 incorporation 版本来满足该请求，反而会失败。如果必须移动约束本身，而又无法删除指定它的 incorporation，则必须将 incorporation 升级到指定所需约束版本的版本。升级 incorporation 会导致不能满足新版本提供的约束的所有合并软件包也进行升级。

系统管理员可以使用 pkg freeze 命令约束软件包。在未提供版本的情况下，将指定软件包约束为系统上已安装的版本。如果提供了版本化软件包，则此管理约束或冻结会像已安装了合并依赖性（其中 fmri 属性具有所提供的软件包版本值）一样进行操作。

包管理系统永远不会自动解除冻结。要解除约束，请使用 pkg unfreeze 命令。

发布者和系统信息库

如上所述，发布者只是软件包客户端用来标识软件包提供者的一个名称。发布者可使用软件包系统信息库和/或软件包归档来发行其软件包。软件包系统当前支持以下两种类型的系统信息库：源系统信息库和镜像系统信息库。

源是包含一个或多个软件包的所有元数据（例如，目录、清单和搜索索引）和内容（文件）的软件包系统信息库。如果在一个映像中为给定发布者配置了多个源，则软件包客户端 API 会尝试选择从其检索软件包数据的最佳源。这是最常见的系统信息库类型，当每次在软件包系统信息库上使用 pkgsend 或 pkgrecv 时进行隐式创建。

mirror 是仅包含软件包内容（文件）的软件包系统信息库。如果在一个映像中为给定发布者配置了一个或多个镜像，则客户端 API 会优先使用镜像进行软件包内容检索并尝试选择从其检索软件包内容的最佳镜像。如果镜像不可访问、不具有所需内容或者运行缓慢，客户端 API 将从任何已配置的源系统信息库检索内容。镜像设计用于通过 pkg.depotd(1M) 的动态镜像功能在一组可信客户端之间共享内容。镜像还设计用于验证对软件包元数据的访问，但发行软件包内容不需要进行验证。例如，客户端可能配置有需要具有 SSL 密钥和证书对才能访问的 https 源，以及提供软件包内容的 http 镜像。这样，只有经过授权的客户端才可以安装或更新软件包，同时避免了验证软件包内容检索的开销。通过删除系统信息库中除名为 file 的子目录及其父目录之外的所有子目录可创建镜像。通过使用 pkg.depotd(1M) 的镜像模式还可将源系统信息库置备为镜像。

侧面和变量

软件可以具有可选组件和互斥组件。可选组件的示例包括语言环境和文档。互斥组件的示例包括 SPARC 或 x86 和调试或非调试二进制文件。

在 IPS 中，可选组件称为**侧面**，互斥组件称为**变量**。侧面和变量指定为软件包操作中的标记。每个侧面和变量标记都有一个名称和值。单个操作可以具有多个侧面和变量标记。具有多个侧面和变量标记的组件有多种，例如供开发者使用的特定于体系结构的头文件，或仅用于 SPARC 全局区域的组件。

以下是变量标记的一个示例 variant.arch=sparc。以下是侧面标记的一个示例 variant.arch=sparc。引用侧面和变量时，通常不带前导字符串 facet. 和 variant.。

侧面和变量是映像的特殊属性，无法在单个软件包上设置。要查看映像上设置的侧面和变量的当前值，请按 pkg(1) 手册页中所示，使用 pkg facet 和 pkg variant 命令。要修改映像上设置的侧面和变量的值，请使用 pkg change-facet 和 pkg change-variant 命令。

侧面为布尔型：只能设置为 true（启用）或 false（禁用）。缺省情况下，映像中的所有侧面都被视为 true。某一操作的侧面标记只能具有 true 值；其他值代表的行为不确定。映像上设置的侧面可以为完整侧面（如 doc.man）或模式（如

locale.*）。要禁用侧面名称空间的一部分，仅启用其中的几个侧面时，此方式很有用。例如，您可以禁用所有语言环境，然后仅启用一个或两个特定语言环境，如以下示例所示：

```
# pkg change-facet locale.*=false
[output about packages being updated]
# pkg change-facet locale.en_US=true
[output about packages being updated]
```

大多数变量可以具有任意数量的值。例如，arch 变量可以设置为 i386、sparc、ppc、arm 或分发支持的任何体系结构。（Oracle Solaris 中仅使用 i386 和 sparc。）但 debug 变量例外。debug 变量只能设置为 true 或 false；其他值的行为不确定。如果文件操作同时具有非调试和调试版本，则必须针对这两个版本明确设置适用的 debug 变量，如以下示例所示：

```
file group=sys mode=0644 overlay=allow owner=root \
  path=etc/motd pkg.csize=115 pkg.size=103 preserve=true \
  variant.debug.osnet=true

file group=sys mode=0644 overlay=allow owner=root \
  path=etc/motd pkg.csize=68 pkg.size=48 preserve=true \
  variant.debug.osnet=false
```

要安装使用变量的软件包，必须在映像上设置变量值。arch 和 zone 变量由创建映像和安装其初始内容的程序设置。缺省情况下，debug.* 变量在映像中设置为 false。

映像中设置的侧面和变量会影响是否安装特定操作。

- 始终会安装不带侧面或变量标记的操作。
- 仅当与侧面标记匹配的所有侧面或侧面模式在映像中都设置为 false 时，才不会安装带这些标记的操作。只要有任何侧面设置为 true 或未明确设置（true 为缺省值），就会安装该操作。
- 仅当所有变量标记的值与映像中设置的值相同时，才会安装带这些变量标记的操作。
- 如果侧面和变量都允许安装操作，则会安装带有这两种标记的操作。

您可以创建自己的侧面和变量标记。以下是 Oracle Solaris 中的常用标记。

变量名	可能值
variant.arch	sparc, i386
variant.opensolaris.zone	global, nonglobal
variant.debug.*	true、false

以下列表描述了 Oracle Solaris 中使用的一小部分侧面标记：

```
facet.devel             facet.doc
facet.doc.html          facet.doc.info
facet.doc.man           facet.doc.pdf
facet.locale.de         facet.locale.en_GB
facet.locale.en_US      facet.locale.fr
facet.locale.ja_JP      facet.locale.zh_CN
```

映像策略

映像策略由具有布尔值的映像属性所定义。有关 `flush-content-cache-on-success` 和 `send-uuid` 属性的说明以及如何查看和修改其值的信息，请参见 pkg(1) 手册页中的“映像属性”。

文件

因为 pkg(5) 映像可位于任意一个较大的文件系统内，需要使用标记 `$IMAGE_ROOT` 来区分相对路径。对于典型的系统安装，`$IMAGE_ROOT` 等效于 /。

`$IMAGE_ROOT/var/pkg` 完整或部分映像的元数据目录。

`$IMAGE_ROOT/.org.opensolaris,pkg` 用户映像的元数据目录。

在特定映像的元数据中，某些文件和目录可能包含修复和恢复期间有用的信息。标记 `$IMAGE_META` 用于指示包含元数据的顶层目录。 `$IMAGE_META` 通常是以上给出的两个路径之一。

`$IMAGE_META/lost+found` 在软件包操作期间移动的有冲突目录和文件的位置。

`$IMAGE_META/publisher` 为每个发布者包含一个目录。每个目录存储特定于发布者的元数据。

`$IMAGE_META` 目录分层结构中的其他路径是专用的，但可以进行更改。

属性

有关下列属性的说明，请参见 `attributes(5)`：

属性类型	属性值
可用性	package/pkg
接口稳定性	Uncommitted（未确定）

另请参见

pkg(1)、pkgsend(1)、pkg.depotd(1m)、pkg.sysrepo(1m)、svcs(1)、svcadm(1M)

http://hub.opensolaris.org/bin/view/Project+pkg/