概览
Pacman 包管理器是 Arch Linux 的一大亮点。它将一个简单的二进制包格式和易用的构建系统结合了起来(参见 makepkg 和 ABS)。Pacman 使得简单的管理与自定义软件包成为了可能，而不论他们来自于官方的 Arch 软件库或是用户自己创建的。
Pacman 可以通过和主服务器同步包列表来进行系统更新，这使得注重安全的系统管理员的维护工作成为轻而易举的事情。这种服务器/客户端模式也使你可以使用简单的命令下载/安装软件包，同时具备了所有必需的依赖包（和 Debian 的 apt-get 相似）。

安装软件包
安装或者升级单个软件包，或者一列软件包（包含依赖包），使用如下命令：
pacman -S package_name1 package_name2
有时候在不同的软件仓库中，一个软件包有多个版本（比如 extra 和 testing）。你可以选择一个来安装：
pacman -S extra/package_name
pacman -S testing/package_name

删除软件包
删除单个软件包，保留其全部已经安装的依赖关系
pacman -R package_name
删除指定软件包，及其所有没有被其他已安装软件包使用的依赖关系：
pacman -Rs package_name
缺省的，pacman 会备份被删除程序的配置文件，将它们加上*.pacsave 扩展名。如果你在删除软件包时要同时删除相应的配置文件（这种行为在基于 Debian 的系统中称为清除 purging），你可是使用命令：
pacman -Rn package_name
当然，它也可以加上-s 参数来删除当前无用的依赖。这样的话，真正删除一个软件包、它的配置文件以及所有不再需要的依赖的命令如下：
pacman -Rsn package_name
注意！Pacman 不会删除软件包安装后才创建的配置文件。你可以从你的 home 文件夹中手动删除它们。

升级系统
Pacman 能够只用一个指令来升级系统中所有已安装的包。升级的时间取决于你的系统有多新。
pacman -Su
当然，最好做法的是将升级系统和同步仓库数据合成为一条指令：
pacman -Syu

查询包数据库
Pacman 可以在包数据库中查询软件包，查询位置包含了包的名字和描述：
pacman -Ss package
要查询已安装的软件包：
pacman -Qs package
一旦你得到了软件包的完整名字，你可以获取关于它的更为详尽的信息：
pacman -Si package
pacman -Qi package
要获取已安装软件包所包含文件的列表：
pacman -Ql package
你也可以通过查询数据库获知目前你的文件系统中某个文件是属于哪个软件包。
pacman -Qo /path/to/a/file
要罗列所有不再作为依赖的软件包(孤立 orphans)：
pacman -Qdt
Pacman 使用-Q 参数来查询本地软件包数据库。参见：
pacman -Q --help
...而使用-S 参数来查询远程同步的数据库。参见：
pacman -S –help
其它用法
Pacman 是个非常广泛的包管理工具，这里只是它的一些其它主要特性。
下载包而不安装它：
pacman -Sw package_name
安装一个'本地'包（不从源里）：
pacman -U /path/to/package/package_name-version.pkg.tar.gz
安装一个'远程'包（不从源里）：
pacman -U http://url/package_name-version.pkg.tar.gz
清理当前未被安装软件包的缓存(/var/cache/pacman/pkg)：
pacman -Sc
完全清理包缓存：
pacman -Scc
Warning: 关于 pacman -Scc，仅在你确定不需要做任何软件包降级工作时才这样做。
pacman -Scc 会从缓存中删除所有软件包。
要删除孤立软件包（递归的，要小心）：
pacman -Rs $(pacman -Qtdq)
重新安装你系统中所有的软件包（仓库中已有的）：
pacman -S $(pacman -Qq | grep -v "$(pacman -Qmq)")
获取本地软件包和它们大小的一个已排序清单列表：
LANG=C pacman -Qi | sed -n '/^Name[^:]*: \(.*\)/ {s//\1 /;x};/^Installed[^:]*: \(.*\)/ {s//\1/;H;x;s/\n//;p}' | sort -nk2
要了解更详细的参数开关可以 pacman --help 或者 man pacman。

配置
Pacman 的配置文件位于/etc/pacman.conf。关于配置文件的进一步信息可以用 man pacman.conf 查看。

常用选项
常用选项都在[options]段。阅读 man 手册或者查看缺省的 pacman.conf 可以获得有关信息和用途。

跳过升级软件包
如果由于某种原因，你不希望升级某个软件包，可以加入内容如下：
IgnorePkg = 软件包名

跳过升级软件包组
和软件包一样，你也可以象这样跳过升级某个软件包组：
IgnoreGroup = gnome

软件仓库
你可以在/etc/pacman.conf 和/etc/pacman.d/里定义使用哪些仓库。它们可以直接在里面定义或者从其它文件里包含进来。
所有官方软件仓库都使用同一个包含了'$repo'变量的/etc/pacman.d/mirrorlist 文件，因此你只需要维护这个列表文件就够了。
下面是有许多镜像的官方软件仓库的一个范例。由于需要节流所以要避免使用 ftp.archlinux.org。
[core]
# Add your preferred servers here, they will be used first
Include = /etc/pacman.d/mirrorlist

[extra]
# Add your preferred servers here, they will be used first
Include = /etc/pacman.d/mirrorlist

[community]
# Add your preferred servers here, they will be used first
Include = /etc/pacman.d/mirrorlist
Note: 使用 testing 仓库的时候要谨慎！

错误
如果你遇到这样的错误信息
not found in sync db
...这很可能是因为软件仓库设置不正确导致不能定位软件包。

1、配置 pacman
pacman 总配置文件/etc/pacman.conf
Arch 通过 pacman 可以获得对下面 4 软件仓库的管理:
[core]
[core]（“核心”）仓库的简单原则是只提供一个基本的 Arch Linux 的基本工具：GNU 系列工具，Linux 内核，一个文字编辑器，一个命令行浏览器等（这里也有一些例外。比如说，vi 和 nano 是同时提供的，允许用户选择一个或两个）。它包含所有使你的系统正常运转的“必须”安装的包，只有它们完美地工作，你的系统才能继续运行。这些是绝对“system-critical”（对系统来说至关重要）的包。由开发者负责维护。
Core 安装媒介（The Core installation media）简单地包含一个安装脚本，一个当前发布版本的核心仓库的快照（snapshot）。
[extra]
[extra]（“额外的”）仓库包含所有的对于一个基本的 Arch 系统不是必须，但是对于构建一个完整的系统环境有用的的 Arch 包。 比如 X, KDE 和 Apache，都可以在这里面找到。它也是由开发者维护的。
[testing]
[testing]（“测试”）仓库包含的包是[core]和[extra]仓库的候选者。 下面情况下，新的软件包将进入[testing]软件仓库:
可能会导致系统出问题的升级，需要测试。
需要重新打包其他软件包。这种情况下，所有需要重新打包的软件包将被放在[testing]仓库，直到所有重新打包完成，这些软件包才回归原有的软件仓库。
开发者维护。
Note: [testing]是唯一的在命名上与其它官方仓库有冲突的官方仓库。所以，如果开启之，[testing]必须放在 pacman.conf 的软件仓库列表中的第一个。
Warning: 警告: 开启[testing]软件仓库，容易导致系统崩溃，因此只建议有经验的用户开启。
[community]
[community]（“社区”）仓库是由“信任用户”（Trusted Users (TUs)）维护的，是 Arch User Repository (AUR)（Arch 用户仓库）的一部分。它包含的是二进制的包，在用户仓库中有足够的“票数”,并由信任用戶挑选出来的。像其它上面列出的仓库一样，[community] 可以用 pacman 获取。
AUR 也包含 unsupported 分支，这不能直接通过 pacman 获取。*. [unsupported] 包含海量 PKGBUILD 脚本，用来从源代码编译。这从其它仓库中是不能使用的。
* AUR Helpers 能帮助你无缝地使用 AUR。
/etc/pacman.conf
每次调用 pacman，它都会读取/etc/pacman.conf 文件。这个配置文件分若干段，每一段定义了一个 pacman 用来搜索软件的仓库。唯一例外的是 options 段，它定义了全局选项。

nano /etc/pacman.conf
范例：
#
/etc/pacman.conf
#
See the pacman.conf(5) manpage for option and repository directives

#
GENERAL OPTIONS
#
[options]
The following paths are commented out with their default values listed.
If you wish to use different paths, uncomment and update the paths.
#RootDir = /
#DBPath = /var/lib/pacman/
#CacheDir = /var/cache/pacman/pkg/
#LogFile = /var/log/pacman.log
HoldPkg = pacman glibc
# If upgrades are available for these packages they will be asked for first
SyncFirst = pacman
#XferCommand = /usr/bin/wget --passive-ftp -c -O %o %u
#XferCommand = /usr/bin/curl %u > %o

```
# Pacman won't upgrade packages listed in IgnorePkg and members of IgnoreGroup
#IgnorePkg =
#IgnoreGroup =

#NoUpgrade =
#NoExtract =

# Misc options (all disabled by default)
#NoPassiveFtp
#UseSyslog
#ShowSize
#UseDelta
#TotalDownload
#
# REPOSITORIES
# - can be defined here or included from another file
# - pacman will search repositories in the order defined here
# - local/custom mirrors can be added here or in separate files
# - repositories listed first will take precedence when packages
# have identical names, regardless of version number
# - URLs will have $repo replaced by the name of the current repo
#
# Repository entries are of the format:
# [repo-name]
# Server = ServerName
# Include = IncludePath
#
# The header [repo-name] is crucial - it must be present and
# uncommented to enable the repo.
#

# Testing is disabled by default. To enable, uncomment the following
# two lines. You can add preferred servers immediately after the header,
# and they will be used before the default mirrors.
#[testing]
#Include = /etc/pacman.d/mirrorlist

[core]
# Add your preferred servers here, they will be used first
Include = /etc/pacman.d/mirrorlist

[extra]
# Add your preferred servers here, they will be used first
Include = /etc/pacman.d/mirrorlist

[community]
# Add your preferred servers here, they will be used first
Include = /etc/pacman.d/mirrorlist

# An example of a custom package repository. See the pacman manpage for
# tips on creating your own repositories.
#[custom]
#Server = file:///home/custompkgs
```

要开启一个软件仓库，只需要将'Include ='和'[repository]'行前的注释符号(#)去除。"Server ="行如果取消注释，将会强制此服务器被首先搜索，详细的仓库配置位于/etc/pacman.d/下。
注意：选择仓库时，请确认将仓库名和'Include ='行都一同取消注释。没有这样做的话会导致你选择的仓库被忽略！这是个很普遍的错误。

（如果有必要）配置镜像列表 /etc/pacman.d/mirrorlist
更快的镜像能显著改善 pacman 的性能和你对 Arch Linux 的总体体验。
手动编辑/etc/pacman.d/mirrorlist：
nano /etc/pacman.d/mirrorlist
删除所有不在你所在洲或者太远的镜像。（在 nano 里你可以用 CTRL-K 删除每个不需要的行。）保存文件并退出。
rankmirrors 脚本可以根据镜像的响应延迟对其自动排名。Rankmirrors 是 python 脚本，需要安装 python（提示：首先要更新一下包列表）：
pacman -Syy
pacman -S python
然后，运行 rankmirrors 脚本按响应延迟顺序对镜像进行排名：
rankmirrors -v /etc/pacman.d/mirrorlist
输出会列出所有镜像和它们的响应延迟（ping），例如：

```
# United States
# http://mirrors.easynews.com/linux/archlinux/$repo/os/i686 ... 0.96
# ftp://ftp.nethat.com/pub/linux/archlinux/$repo/os/i686 ... unreachable
# ftp://locke.suu.edu/linux/dist/archlinux/$repo/os/i686 ... 2.43
# ftp://mirrors.unixheads.org/archlinux/$repo/os/i686 ... 1.96
```

脚本将从低到高的响应延迟顺序显示：

```
Server = http://mirrors.easynews.com/linux/archlinux/$repo/os/i686
Server = ftp://mirrors.unixheads.org/archlinux/$repo/os/i686
Server = ftp://locke.suu.edu/linux/dist/archlinux/$repo/os/i686
Server = ftp://ftp.nethat.com/pub/linux/archlinux/$repo/os/i686
```

要注意的是这个结果对于镜像排列并非很精确，最接近你的镜像或者延迟最低的镜像并不一定是最佳选择。（反应迅速但是内容过时的镜像、或者低延迟但是窄带宽的镜像对你毫无用处）。
编辑/etc/pacman.d/mirrorlist，将最佳的镜像置于在列表顶端。为了体验多个镜像你可能要不断修改此配置文件，请精明地选择。

常用的命令

与远程软件仓库同步和更新本地包数据库 (推荐在安装和更新包之前这样做):
pacman -Sy
升级系统中所有的包:
pacman -Su
一条同步、更新、升级 所有系统中的包的命令:
pacman -Syu
安装或者升级单个软件包或者一串包(包括其依赖包):
pacman -S packageA packageB
移除软件包, 但保留依赖的包:
pacman -R package
移除单个包，并且移除没有被别的软件依赖的依赖包:
pacman -Rs package
移除包所有不需要的依赖包并删除其配置文件:
pacman -Rsn package
通过给定关键词（列表）搜索远程软件仓库数据库(repo):
pacman -Ss keyword
列出系统中所有的包
pacman -Q
在本地包数据库搜索 (查询)指定软件包:
pacman -Q package
在本地包数据库搜索 (查询)指定软件包并列出相关信息:
pacman -Qi package
To defragment pacman 的数据库缓存和速度优化选项:
pacman-optimize
统计当前系统中的包数量:
pacman -Q | wc -l
使用 ABS 和 makepkg 从源代码编译安装包:
pacman -U packagename.pkg.tar.gz
Note: pacman 有很多很多的选项和功能组合。你可以试试 man pacman 或者查阅 lt the pacman 维基条目

软件包管理增强工具
Yaourt 可查询 aur 的 pacman 增强工具
Tupac 快速查询的 yaourt 增强工具
Powerpill 多进程下载的 pacman 增强工具
Note: 强烈建议更新系统完毕后，安装 yaourt 和 powerpill，以便更充分享受 Arch Linux 的方便和强大。
在/etc/pacman.conf 文件里，最后添加两行(如果是 i686 的话，把 x86_64 换成 i686 即可):

```
[archlinuxfr]
Server = http://repo.archlinux.fr/x86_64
```

然后使用 pacman 安装 yaourt
pacman -Sy yaourt
以后就可以使用 yaourt 命令来查找并安装包括 AUR 里的软件了。比如 bcm 无线网卡，需要安装 b43-firmware（此软件包在 AUR 仓库里，使用 pacman 无法查找到），使用 yaourt 命令查找并安装:
yaourt b43-firmware
找到之后，根据相应提示，一直到安装完成。

图形化管理工具
qt 界面的 shaman：比较成熟的 pacman 图形化管理工具
gtk 界面的 gtkpacman： 目前 BUG 比较多，不稳定。

Pacman 的更多信息
用此命令查看 pacman 手册：
man pacman
有空的话可以看看 pacman (简体中文)维基条目。

2、更新 Pacman 自己
一开始，系统可能会提示你先更新 pacman 自身，这取决于 Arch 安装包的新旧程度：
pacman -Sy pacman
pacman 开始查看是否需要更新，如果有提示需要更新，则按'y'，开始 pacman 更新。有时候，还需要用户主动修改某些配置；请仔细阅读升级过程中的输出获取相关信息。

pacman

-S 安装
-y, --refresh download fresh package databases from the server，同步更新 package list 列表
-f, --force force install, overwrite conflicting files
-i, --info view package information
-l, --list <repo> view a list of packages in a repo
-w, --downloadonly download packages but do not install/upgrade anything
-g, --groups view all members of a package group，包括运程服务器上的 group
-s, --search <regex> search remote repositories for matching strings
-p, --print-uris print out URIs for given packages and their dependencies
-r, --root <path> set an alternate installation root（类似 chroot）
-b, --dbpath <path> set an alternate database location
--cachedir <dir> set an alternate package cache location
--ignore <package,...> 不要升级指定的软件包
--ignoregroup base,base-devel 不要升级 group 中的软件包
--config <path> set an alternate configuration file
--noconfirm do not ask for any confirmation
--noscriptlet do not execute the install scriptlet if one exists

--asdeps install packages as non-explicitly installed
--asexplicit install packages as explicitly installed
-c, --clean remove old packages from cache directory (-cc for all)
-d, --nodeps skip dependency checks
-u, --sysupgrade upgrade all packages that are out of date
--needed don't reinstall up to date packages

```
--ignore <pkg>      ignore a package upgrade (can be used more than once)
--ignoregroup <grp> ignore a group upgrade (can be used more than once)
-q, --quiet         show less information for query and search
--logfile <path>    set an alternate log file
--noprogressbar     do not show a progress bar when downloading files
-v, --verbose       be verbose
```

-Q 查询

```
-i, --info          view package information (-ii for backup files)
-g, --groups        view all members of a package group，仅本地安装的 group
-l, --list          list the contents of the queried package
-o, --owns <file>   query the package that owns <file>
-p, --file <xxx.pkg.tar.gz> 获取该软件包的名称以及版本号
-s, --search <regex> search locally-installed packages for matching strings
-t, --unrequired    list all packages not required by any package
-u, --upgrades      list all packages that can be upgraded
-v, --verbose       be verbose
-r, --root <path>   set an alternate installation root
-b, --dbpath <path> set an alternate database location
--cachedir <dir> set an alternate package cache location
```

```
-c, --changelog     view the changelog of a package
-d, --deps          list all packages installed as dependencies
-e, --explicit      list all packages explicitly installed
-m, --foreign       list installed packages not found in sync db(s)
-q, --quiet         show less information for query and search
--config <path> set an alternate configuration file
--logfile <path> set an alternate log file
--noconfirm     do not ask for any confirmation
--noprogressbar do not show a progress bar when downloading files
--noscriptlet   do not execute the install scriptlet if one exists
```

-U 升级

```
-U xxx.pkg.tar.gz 直接安装该软件包，要给出全路径
-c, --cascade       remove packages and all packages that depend on them
-d, --nodeps        skip dependency checks
-k, --dbonly        only remove database entry, do not remove files
-n, --nosave        remove configuration files as well
-s, --recursive     remove dependencies also (that won't break packages)
(-ss includes explicitly installed dependencies too)
-u, --unneeded      remove unneeded packages (that won't break packages)
--config <path> set an alternate configuration file
--logfile <path> set an alternate log file
--noconfirm     do not ask for any confirmation
--noprogressbar do not show a progress bar when downloading files
--noscriptlet   do not execute the install scriptlet if one exists
-v, --verbose       be verbose
-r, --root <path>   set an alternate installation root
-b, --dbpath <path> set an alternate database location
--cachedir <dir> set an alternate package cache location
```

ABS - the Arch Build System

Arch Build System（以下简称 ABS）用于：
①制作新的软件包
②根据自己的需要定制软件包（使用 enabling 或 disabling 选项）
③用你自己的编译选项重新编译整个系统（就像 gentoo 一样了）
对于 Arch Linux 来说，ABS 不是必须的，但很有用。
本文将简要介绍 ABS 及 Arch 的软件包，这不是一个完全参考指南！如果您想详细了解，您应该去读一读手册页。

1.安装软件包

使用 ABS 之前，你必须先安装 cvsup 及 wget：

```
pacman -Sy cvsup wget
```

如果你已将软件包下载到一个名为 foo 的目录中：

```
cd foo
pacman -A cvsup-*.pkg.tar.gz wget-*.pkg.tar.gz
```

2.什么是软件包文件?

a.一般地，软件包文件就是一个名如 foo.pkg.tar.gz 的文件。
b.实际上，软件包文件只是一个用 gzip 压缩的 tar 档，包含：
①需安装的文件
②.PKGINFO： 包含 pacman 处理该软件包的所有信息，依赖关系等等
③.FILELIST： 软件包中所有文件的列表，用来删除软件或检查文件冲突
④.INSTALL： 存放在安装/升级/删除软件后执行的命令（只有在 PKGBUILD 中指定，才会有此文件）。

3.PKGBUILD 是什么？它包含哪些内容?

PKGBUILD 文件包含软件包有关的一些信息，它只是一个简单的纯文本文件。这儿有一个例子：

```
# $Id: PKGBUILD,v 1.12 2003/11/06 08:26:13 dorphell Exp $
# Maintainer: judd
# Contributor: Judd Vinet
pkgname=foo
pkgver=0.99
pkgrel=1
pkgdesc="short description of foo"
url="http://www.foo.org"
groups=
provides=
depends=('qt' 'python')
makedepends=('guile')
conflicts=('yafoo')
replaces=('mffoo')
backup=('/etc/foo/foo.conf')
install=('foo.install')
source=('http://www.foo.org/download/$pkgname-$pkgver.tar.gz')
md5sums=('2c0cca3ef6330a187c6ef4fe41ecaa4d35175bee593a7cc7d6205584a94d8625')

build() {
cd $startdir/src/$pkgname-$pkgver
./configure --prefix=/usr
make || return 1
make prefix=$startdir/pkg/usr install
}
```

以下是说明：

```
# ：#后为注释
# $Id: PKGBUILD,v ... : 该软件包的 cvs-tag（cvs 标记），是由 archlinux-cvs 系统建立的。

# Maintainer : 维护者。维护者负责维护此软件包的官方版本。
# Contributor : 贡献人。为这个软件包写第一个 PKGBUILD 的人。
pkgname : 该软件包的名字。
pkgver : 该软件包的版本号。
pkgrel : Arch 软件包的发布号。它与版本号是不同的，发布号随 PKGBUILD 的修改而改变。有很多原因导致修改 PKGBUILD，例如，你可能会为打开 compile-time 支持而改变 PKGBUILD。
pkgdesc : 该软件包的主要描述，这就是你在浏览软件包数据库（http://archlinux.org/packages.php）时所看到的。
url : 包中软件的主页（当你在浏览软件包数据库时，点击软件包名就进入包中的软件的主页）。
groups : 用于软件包组。例如，当你试图安装 kde 时，所有属于 kde 组的软件包都将被安装。
provides : 表示该软件包作为另一个软件包的补充。例如，kernel-scsi 作为 kernel 的补充。

depends : 软件包的依赖关系（它需要哪些软件包）。
makedepends : 编译该软件包时才需要的包，一旦完成编译就不再需要了。
conflicts : 相冲突的包，这些包不能同时安装。本例中，foo 与 yafoo 相冲突。
replaces : 该软件包所取代的旧包。本例中，foo 取代了 mffoo，并且不再提供对 mffoo 的支持。
backup : 当软件包被卸载时需备份的文件（备份后的文件加上.pacsave 后缀，形如 file.pacsave）。
install : 软件包中的安装脚本（必须与 PKGBUILD 在同一目录中）。
source : 软件包的下载地址，可以是本地、也可以是通过"http"或"ftp"访问的远程地址。软件包以 pkgname 和 pkgver 两个变量来命名，以免每次升级都要改变源码。
md5sums : 软件包的 MD5 校验和。

下面是函数的说明：
build : 建立软件包所需的所有步骤（下文将详细说明）。

现在，你可以发现 PKGBUILD 文件所包含的信息是软件包管理者所必须的，它是 pacman 与 abs 的核心。
还有安装脚本，上例中，PKGBUILD 指定"foo.install"作为安装脚本。举例如下：

post_install() {
/bin/true
}

post_upgrade() {
/bin/true
}

pre_remove() {
/bin/true
}

op=$1
shift

$op $*

说明：
post_install : 这段脚本在文件安装后立即执行，它只有一个参数：软件包的版本。
post_upgrade : 这段脚本在所有文件升级后执行，它有两个参数：
①新的软件包的版本
②老的软件包的版本
pre_remove : 这段脚本在文件被删除之前执行（例如，停止一个守护进程），它也只有一个参数：软件包的版本
最后三行是每个安装脚本都必须的，因此，它们都会被执行。

4.build 函数

如果你对建立软件包不太熟悉，你应该知道大多数（不是全部）软件包可以这样建立：
①源文件解压

```
tar -xzf foo-0.99.tar.gz
tar -xjf foo-0.99.tar.bz2
```

②进入源码目录

```
cd foo-0.99
```

③配置软件包：一般来说，源码目录中有一个名为 configure 的短小脚本文件，这就是用
```

来配置软件包的（添加或删除支持信息，选择安装目的目录等等）。检查你的系统，确保已安装此软件所需的所有包，然后执行：
./configure [option]
在正式配置之前，你应该试试 help 选项，以便更好地理解如何配置：
./configure --help
④编译源码
make
⑤安装
make install
不论怎样，你都应该看看 INSTALL 文件，搞清软件应怎样配置和安装！并不所有的软件都可以 configure; make; make install 这样配置安装的！

那么，我们来看一看一个“标准的”build 函数：

build() {
cd $startdir/src/$pkgname-$pkgver
./configure --prefix=/usr
make || return 1
make prefix=$startdir/pkg/usr install
}

我们所做的是：
①进入解压后的源码目录：
cd $startdir/src/$pkgname-$pkgver
但如果你尝试去建立自己的软件包，注意一定要进入正确的目录。有时，解压后的目录名是不同的：
tar -xzf foo-0.99.tar.gz
执行 ls 会返回：
.
..
foo-0.99.tar.gz
foo/
并不是
.
..
foo-0.99.tar.gz
foo-0.99/
②配置，安装目录为/usr：
./configure --prefix=/usr
③编译
make || return 1
④安装，不是在/usr 中，而是在$startdir/pkg/usr 目录中，这样，pacman 就可以控制这些文件。
make prefix=$startdir/pkg/usr install
我们所要做的是建立软件包，而不是要安装它。所以，我们告诉 make 将所有文件放在我们指定的目录：$startdir/pkg/usr，而不是安装到标准位置/usr。这样，makepkg 就可以查找并发现哪些文件是软件包要安装的，并将他们压缩进 Arch 软件包。
注意：有时 Makefile 中并没有 prefix 选项，通常使用 DESTDIR 代替。如果所生成的文件数明显少于应有的数目（译注：也就是说有文件已经被安装进你的系统），试试 make DESTDIR=$startdir/pkg install。如果仍旧不行，你就必须仔细查找以“make <...> install”形式执行的 install 命令的参数。

5.The ABS tree
当你第一次运行 abs 命令时：
root @ localhost # abs
它将使用 CVS，确保“the ABS tree（以下称为 ABS 树）”与 arch 服务器同步。那么，ABS 树到底是什么？它位于/var/abs 目录下，看起来如下图：

| -- base/
| |-- autoconf/
| |-- automake/
| |-- ...
| -- devel/
| -- ...
| -- extra/
| | -- deamons/
| | | -- acpid/
| | | | -- PKGBUILD
... ...

ABS 树的结构与软件包数据库的结构精确地相同：
①第一级目录代表类别
②第二级目录代表软件包
③ PKGBUILD 文件包含软件包所需的所有信息
不管怎样，/var/abs 下还有一个特定的目录：local，这个目录是你的。你可以在此目录中做任何事——但你不应该改变 ABS 树的其他部份。
注意：第一次下载的 ABS 树大一点，接下来就只需要下载升级的部份就可以了。所以即使你是用 56k 的小猫，也不必担心：这些只是文本文件，并且在传输过程中是被压缩的。
现在你知道什么是 ABS 树了，那我们怎么利用它呢?

6.第一次使用 ABS：定制软件包
这种情况比你想到的要多得多：官方软件包编译的选项仅仅是--enable 或--disable，并没有你所必需的选项。
举个例子：foo 软件的 arts 支持被禁止了，假如我们要打开 arts 支持，可以按下面的方法来做：
①找到 foo 包
在 http://archlinux.org/packages.php 查找 foo
使用 find 命令：
find /var/abs -name "foo"
使用 slocate 命令：
slocate foo | grep /var/abs
无论如何，你会发现 foo 是 extra 中 multimedia 的一部份（仅仅是举例）
②拷贝 foo 的 PKGBUILD 文件至/var/abs/local/foo：
mkdir /var/abs/local/foo
cp /var/abs/extra/multimedia/foo/PKGBUILD /var/abs/local/foo
cd /var/abs/local/foo
③修改 PKGBUILD 文件，加入我们需要的 arts 支持：

将
build() {
cd $startdir/src/$pkgname-$pkgver
./configure --prefix=/usr
make || return 1
make prefix=$startdir/pkg/usr install
}

改成
build() {
cd $startdir/src/$pkgname-$pkgver
./configure --enable-arts --prefix=/usr
make || return 1
make prefix=$startdir/pkg/usr install
}

④运行 makepkg：

makepkg

⑤用下面的两条命令中的一条来安装（-A 表示安装，-U 表示升级已安装的软件包）：
pacman -A foo-*.pkg.tar.gz
pacman -U foo-*.pkg.tar.gz

7.ABS 的其他用途
WiKi 上还有两篇文章，所述更深入：
* The Arch package making HOW-TO - with guidelines
* Custom local repository with ABS and gensync