NAME

package repository - format and operation of package repositories used by pkg(8).

DESCRIPTION

Package repositories used by the pkg(8) program consist of one or more collections of "package tarballs" together with package catalogues and optionally various other collected package metadata.

Each collection consists of packages suitable for installation on a specific system **ABI**: a combination of operating system, CPU architecture, OS version, word size, and for certain processors endianness or similar attributes.

The package collections are typically made available to users for download via a web or FTP server although various other means of access may be employed. Encoding the **ABI** value into the repository URL allows **pkg** to automatically select the correct package collection by expanding the special token **${ABI}** in *pkg.conf*.

Repositories may be mirrored over several sites: **pkg** has built-in support for discovering available mirrors dynamically given a common URL by several mechanisms.

FILESYSTEM ORGANIZATION

Only very minimal constraints on repository layout are prescribed by **pkg**. The following constraints are all that must be met:

- A repository may contain several package collections with parallel **REPOSITORY_ROOTs** in order to support diverse system **ABIs**.
- All of the content for one **ABI** should be accessible in a filesystem or URL hierarchy beneath the **REPOSITORY_ROOT**.
- All packages available beneath one **REPOSITORY_ROOT** should be binary compatible with a specific system **ABI**.
- The repository catalogue is located at the apex of the repository, at a specific location relative to the **REPOSITORY_ROOT**.

Package catalogues contain the paths relative to the **REPOSITORY_ROOT** for each package, allowing the full URL for downloading the package to be constructed.

Where a package may be applicable to more than one **ABI** (e.g., it contains only text files) symbolic or hard links, URL mappings or other techniques may be utilised to avoid duplication of storage.

Although no specific filesystem organization is required, the usual convention (inherited from pkg-install(8)) is to create a filesystem hierarchy thus:

$REPOSITORY_ROOT/All
: One directory that contains every package available from the repository for that **ABI**. Packages are stored as package tarballs identified by name and version. This directory may contain several different versions of each package accumulated over time, but the repository catalogue will only record the latest version for each distinct package name.

$REPOSITORY_ROOT/Latest/
: May contains symbolic links to the latest versions of packages in the *All* directory. Symbolic links contain a 'latest link' style name only, without version. As the whole 'latest link' concept is rendered obsolete by **pkg**, this will usually contain only the *pkg.txz* link, used for bootstrapping **pkg** itself on a new system.

$REPOSITORY_ROOT/packagesite.txz
: Contains a single file, usually named *packagesite.yaml*, a concatenation of the *+MANIFEST* files from the packages in the repository. Each manifest is represented as a single-line **JSON** text (no carriage returns or line feeds are used as whitespace within the **JSON** text), and the manifests are separated by newlines. The complete file is not a valid **JSON** text. This is used by **pkg-1.1** or later.

$REPOSITORY_ROOT/filesite.txz
: (Optional). Contains a single file, usually named *filesite.yaml*, a concatenation of the file lists from the packages in the repository. Each file list is represented as a single-line **JSON** text (no carriage returns or line feeds are used as whitespace within the **JSON** text), and the file lists are concatenated with no delimiters. The complete file is not a valid **JSON** text.

The repository may optionally contain sub-directories corresponding to the package origins within the ports tree.

Each of the packages listed in the repository catalogue must have a unique **name**. There are no other constraints: package sets are not required to be either complete (i.e., with all dependencies satisfied) or self-consistent within a single repository.

REPOSITORY ACCESS METHODS

pkg uses standard network protocols for repository access. Any URL scheme understood by the fetch(3) library may be used (**HTTP**, **HTTPS**, **FTP** or **FILE**) as well as remote access over **SSH**. See fetch(3) for a description of additional environment variables, including FETCH_BIND_ADDRESS, FTP_LOGIN, FTP_PASSIVE_MODE, FTP_PASSWORD, FTP_PROXY, ftp_proxy, HTTP_AUTH, HTTP_PROXY, http_proxy, HTTP_PROXY_AUTH, HTTP_REFERER, HTTP_USER_AGENT, NETRC, NO_PROXY and no_proxy.

REPOSITORY MIRRORING

Multiple copies of a repository can be provided for resilience or to scale up site capacity. Two schemes are provided to auto-discover sets of mirrors given a single repository URL.

HTTP The repository URL should download a text document containing a sequence of lines beginning with ‘URL:’ followed by any amount of white space and one URL for a repository mirror. Any lines not matching this pattern are ignored. Mirrors are tried in the order listed until a download succeeds.

SRV For an SRV mirrored repository where the URL is specified as *http://pkgrepo.example.org/* **SRV** records should be set up in the DNS:

```
$ORIGIN example.com
_http._tcp.pkgrepo IN SRV 10 1 80 mirror0
                   IN SRV 20 1 80 mirror1
```

where the **SRV** priority and weight parameters are used to control search order and traffic weighting between sites, and the port number and hostname are used to construct the individual mirror URLs.

Mirrored repositories are assumed to have identical content, and only one copy of the repository catalogue will be downloaded to apply to all mirror sites.

WORKING WITH MULTIPLE REPOSITORIES

Where several different repositories are configured **pkg** will search amongst them all in the order specified by the **PRIORITY** settings in the *repo.conf* files, unless directed to use a single repository by the **-r** flag to pkg-fetch(8), pkg-install(8), pkg-upgrade(8), pkg-search(8) or pkg-rquery(8).

Where several different versions of the same package are available, **pkg** will select the one with the highest version to install or to upgrade an installed package to, even if a lower numbered version can be found in a repository earlier in the list. This applies even if an explicit version is stated on the command line. Thus if packages *example-1.0.0* and *example-1.0.1* are available in configured repositories, then

```
pkg install example-1.0.0
```

will actually result in *example-1.0.1* being installed. To override this behaviour, on first installation of the package select the repository with the appropriate version:

```
pkg install -r repo-a example-1.0.0
```

and then to make updates to that package "sticky" to the same repository, set the value **CONSERVATIVE_UPGRADE** to **true** in *pkg.conf*.

SEE ALSO

pkg_create(3), pkg_printf(3), pkg_repos(3), pkg-keywords(5), pkg-lua-script(5), pkg-script(5), pkg-triggers(5), pkg.conf(5), pkg(8), pkg-add(8), pkg-alias(8), pkg-annotate(8), pkg-audit(8), pkg-autoremove(8), pkg-check(8), pkg-clean(8), pkg-config(8), pkg-create(8), pkg-delete(8), pkg-fetch(8), pkg-info(8), pkg-install(8), pkg-lock(8), pkg-query(8), pkg-register(8), pkg-repo(8), pkg-rquery(8), pkg-search(8), pkg-set(8), pkg-shell(8), pkg-shlib(8), pkg-ssh(8), pkg-stats(8), pkg-triggers(8), pkg-update(8), pkg-updating(8), pkg-upgrade(8), pkg-version(8), pkg-which(8)