Making a LAN mirror of global package repositories

Intro

In this page I outline the recipe for making a package repository server containing up-to-date packages from `pkg.openindiana.org` for use in a LAN or on a host which plans to spawn many local zones. Presence of a local package repository mirror can save some traffic, as well as enables you to do further package installations independently of Internet availability to your host or LAN.

This page is largely based on knowledge published by other authors, including pieces available on OpenIndiana and illumos Wikis, I tried to reference all sources in the end of this page.

Related topics include Building in zones and Using host-only networking to get from build zones and test VMs to the Internet.

Seeding the repository

Creating the repository storage datasets

> ⚠ One of the papers (see the Links section below) suggests using compression on the package repo dataset, while another says the package files are pre-compressed and there will be no bonus from ZFS compression, only an overhead. Upon my tests with several oi_151 repos (listed below), an 8.2GB set of files got compressed 1.10x (10%) with `gzip-9`, so go figure.
>
> Why not enable the compression for WORM data anyway, however, just in case it helps?..
>
> Also at least one document suggests making a sub-dataset for the OpenIndiana "dev" repository, while your other repos can live in other datasets. Makes sense.

Create the compressed hierarchy of datasets for pkg repos:

```
:; zfs create -o atime=off -o compression=gzip-9 rpool/export/pkg
:; zfs create rpool/export/pkg/dev
:; zfs create rpool/export/pkg/tarballs
```

Using prepackaged repository snapshots

The OpenIndiana download site provides several assorted tarballs of the package repositories for *some* of the (intermediate) releases. They can be used to pre-populate a local pkg-repo, and this pkg-repo can be updated by `rsync`.

Alternately, `rsync` can be used right away to initiate the repository with all published data (starting from release oi_147); if you prefer this route – scroll down to the chapter "rsync the package repository contents and/or updates".

Fetch the repository snapshots

Archives of some "dev-releases" of the package repository are provided at http://dlc-int.openindiana.org/repos/. They can be downloaded and unpacked to start up a package repository of a particular release.
A README file or some other map would be nice 😎

If you want, you can get all of the repos (note that each oi_151 build repo is 2.6Gb in size, and note that the `legacy` repo is almost 60Gb). Assuming you have an `/export/pkg/tarballs` and it is big enough:

```
:; cd /export/pkg/tarballs
:; wget -r -nd -nH -l 1 -c -N -np -A=gz,bz2,md5,sha1,sha256 http://dlc-int.openindiana.org/repos/
```

Alternately, look into that resource with a web browser and get just the latest available repository tarball and its checksums, i.e.:

```
:; cd /export/pkg/tarballs
:; wget http://dlc-int.openindiana.org/repos/oi_151a_prestable2_repo.tar.bz2.md5
:; wget http://dlc-int.openindiana.org/repos/oi_151a_prestable2_repo.tar.bz2.sha1
:; wget http://dlc-int.openindiana.org/repos/oi_151a_prestable2_repo.tar.bz2.sha256
:; wget http://dlc-int.openindiana.org/repos/oi_151a_prestable2_repo.tar.bz2
```

Then, test the received files, like this:

```
:; for F in oi_*_repo.tar.bz2; do for S in md5 sha1 sha256; do
    [ -s "$F.$S" ] && echo "=== $S:" && cat "$F.$S" && "$S"sum "$F"
done; done
```

This should output sets of two lines per checksum method per file with identical checksums, like these:

```
=== md5:
8f78b573779608a93cfec61a6d9ef3fa  oi_151a_prestable1_repo.tar.bz2
8f78b573779608a93cfec61a6d9ef3fa  oi_151a_prestable1_repo.tar.bz2
=== sha1:
3c0fd38db9a5c849257085bda80e020bb4519142  oi_151a_prestable1_repo.tar.bz2
3c0fd38db9a5c849257085bda80e020bb4519142  oi_151a_prestable1_repo.tar.bz2


=== md5:
6bc84c9109130d6543e8c8337f0a2ea0
6bc84c9109130d6543e8c8337f0a2ea0  oi_151a_prestable2_repo.tar.bz2
=== sha1:
511040f11796d1fad0195303e65bad80663c8f84
511040f11796d1fad0195303e65bad80663c8f84  oi_151a_prestable2_repo.tar.bz2
=== sha256:
5660dd8243dfe00534aee319c7c939cd02af9d2ff662bd6f4d27b1bd79fb4993
5660dd8243dfe00534aee319c7c939cd02af9d2ff662bd6f4d27b1bd79fb4993  oi_151a_prestable2_repo.tar.bz2
```

Unpack the package repository snapshots

Inside these archives have similar structures topped by variably-named top-level directories like:

- `dev_151a` (from `oi_151a.tar.bz2`),
- `prestable` (from `oi_151a_prestable0_repo.tar.bz2`),
- `out-repo` (from `oi_151a_prestable1_repo.tar.bz2` and `oi_151a_prestable1_repo_incr.tar.bz2`).

Each such directory contains a `publishers/openindiana.org` directory with package files, and a `pkg5.repository` metadata snippet.

```
:; cd /export/pkg/dev
:; gtar -xjf /export/pkg/tarballs/oi_151a_prestable2_repo.tar.bz2
```

It would take some effort to merge the directory contents into one repository, by moving files from older sources first, then overwriting (as necessary) with newer ones; I did this lazily, with Midnight Commander (`mc`). As a result, the `publishers/` sub-directory and `pkg5.repository` metadata reside directly in my `/export/pkg/dev/`.

rsync the package repository contents and/or updates

The whole repository can be updated (or even fetched right away, skipping the downloadable tarballs part) from the origin servers, but note that your repository would then include revisions for many releases of OpenIndiana (147-151a4 as of now) and take over 8GB of disk in size.

```
:; rsync -zavPHK pkg-origin.openindiana.org::pkgdepot-dev /export/pkg/dev/
```

Set up a pkg.depotd SMF instance

Now that you have repository data, you can create the SMF instance for the package depot server – which would actually server IPS packages to IPS clients (the `pkg(5)` program and its relatives).

Following another manual (with additions which I found were needed):

```
:; svccfg -s pkg/server
setprop startd/duration = contract
add dev
select dev
addpg pkg application
setprop pkg/inst_root = astring: "/export/pkg/dev/"
setprop pkg/port = count: 10002
setprop pkg/proxy_base = astring: http://myhostname.domain.org:10002
exit

:; svcadm refresh pkg/server:dev
:; svcadm enable pkg/server:dev
```

NOTES:

- The default mode for this service is `child`. That failed on my box (SMF thought the service died after daemonization and spawned new daemons doomed to fail due to busy TCP port; this can also lead to memory leaks in `svc.startd`, tracked in bug #2801); the `contract` mode takes care of this, seemingly properly.
- It is possible to add listeners on a dedicated IP with the same port number, but SMF dependencies should be set correctly so that the catch-all listener starts last, as detailed in Using host-only networking to get from build zones and test VMs to the Internet#(Optional) Prepare a package repository listener on the private network.
- Without the `proxy_base`, browser accesses to the port were redirected to "`http://0.0.0.0:10002/...`" which did not work 😅
- I did not yet figure out if I can make it use an URL like "`http://myhostname.domain.org:10002/dev/`" without changing any files in the repo (so as to not lose these changes with another `rsync` run later). So at the moment the one server can publish one repository. It can, however, live along with a different pkg depot server which works on a different port and publishes different packages, i.e. my `on_nightly` (see Redistributing built packages). For the global or local zone installation and maintenance one or another should be primary, however...
  It also seems critical that if the `proxy_base` is used, at least with no relative URI, then **its value should not end** with the slash. Otherwise, there's an infinite redirect loop to `...:10002//en/index.shtml` URL.

Test the working repository

```
:; pkgrepo info -s http://localhost:10002
PUBLISHER       PACKAGES STATUS           UPDATED
openindiana.org 3908     online           2012-05-07T22:30:33.986403Z
```

Add the local mirror as an optional pkg source

Using a mirror, apparently, should direct the metadata traffic (i.e. checks for current package versions) to the "origin" server if available, while the mirror itself should serve whatever content-data it has.

Note that this is different from setting a preferred publisher with, possibly, different sets or builds of packages.

```
:; pkg set-authority -m http://localhost:10002/ openindiana.org

:; pkg publisher
PUBLISHER                             TYPE     STATUS   URI
openindiana.org                       origin   online   http://pkg.openindiana.org/dev/
openindiana.org                       mirror   online   http://localhost:10002/
```

NOTES:

- The examples above use the `dev` repository. Your system might use another, hence the check. Also note the comment above regarding the paths – until I figure out how to serve relative URLs, the repo is hosted at the HTTP namespace's root.
- To remove a mirror, the `-M` flag can be used instead of `-m`, the other parameters remain on the command-line.
- When making a mirror for the LAN, obviously, the server's known name should be used instead of `localhost`.

Use the local mirror as a pkg source when installing local zones

If you want to use a particular package source (publisher and URI) for installing a local zone, that can be arranged too:

```
:; dladm create-vnic -l e1000g0 vnic1

:; zonecfg -z zone1
create -b
set zonepath=/zones/build/zone1
set brand=ipkg
set autoboot=true
set ip-type=exclusive
add net
set physical=vnic1
end
verify
commit
exit

:; zoneadm -z zone1 install -P openindiana.org=http://`hostname`.`domainname`:10002/
A ZFS file system has been created for this zone.
   Publisher: Using openindiana.org (http://openindiana.local.net:10002/ ).
       Image: Preparing at /zones/build/zone1/root.
Caching catalogs ...
Refreshing catalog 1/1 openindiana.org
Caching catalogs ...
       Cache: Using /var/pkg/publisher.
Sanity Check: Looking for 'entire' incorporation.
  Installing: Packages (output follows)
Creating Plan -
           Packages to install: 134
       Create boot environment:  No
Create backup boot environment:  No
            Services to change:   4

DOWNLOAD                                  PKGS       FILES    XFER (MB)
Completed                               134/134 28046/28046  149.2/149.2

PHASE                                        ACTIONS
Install Phase                              41385/41385

PHASE                                          ITEMS
Package State Update Phase                   134/134
Image State Update Phase                         2/2

        Note: Man pages can be obtained by installing pkg:/system/manual
 Postinstall: Copying SMF seed repository ... done.
 Postinstall: Applying workarounds.
        Done: Installation completed in 135.944 seconds.

  Next Steps: Boot the zone, then log into the zone console (zlogin -C)
              to complete the configuration process.
```

Likewise, specific packages (above the predefined minimum) can be requested with `-e` option. For more details see:

- http://docs.oracle.com/cd/E19082-01/819-2252/6n4i8rtqs/index.html

Also note that the first run may take a while, I guess some things are getting cached (my first local installation took 233 seconds, subsequent ones took ~135sec; install from internet was ~140sec). In my case I rather saved on traffic (installing many zones on many hosts) and internet-independence during the installation than on wallclock time; the system was likely IOPS-bound with its local disk. YMMV.

Also note that you CAN use "`localhost`" as the package server name in the call above, and the local zone will be installed by the global zone accessing its `localhost` package depot listener. However, package installations and updates with `pkg(5)` calls issued from inside the local zone would likely fail, unless you pass accesses to the local zone's `localhost:10002` up to a real package depot server, or install one in the zone. Indeed, this may be what you want to create local zones incapable of auto-updates from inside...

In real life you'd likely clone the first installed "golden" local zone, though, with `zfs` and/or `zoneadm` tools, rather than install many identical zones on the same machine with the complete original procedure. Why waste time and space? 😷

Links

Most of the work was done and described by others, in assorted posts. Links include:

- http://wiki.openindiana.org/oi/Mirroring+OpenIndiana

- http://wiki.openindiana.org/oi/Mirrors
- http://wiki.openindiana.org/display/oi/Package+Repositories
- http://wiki.openindiana.org/display/oi/Distribution+Constructor
- http://wiki.openindiana.org/display/oi/Automated+Installer+and+Networked+Installation
- http://www.agileweboperations.com/setting-up-an-opensolaris-pkg-repository-mirror
- http://notallmicrosoft.blogspot.com/2010/04/complete-local-copy-mirror-of.html
- http://www.phwinfo.com/forum/comp-unix-solaris/484066-how-set-default-publisher-repository.html
- `pkg.depotd`(1M), `pkgrecv`(1), `ipkg`(5) brand and related manpages:
  http://docs.oracle.com/cd/E23824_01/html/E21796/pkg-depotd-1m.html
  http://docs.oracle.com/cd/E23824_01/html/E21796/pkgrecv-1.html
  http://docs.oracle.com/cd/E19082-01/819-2252/6n4i8rtqs/index.html
- http://www.jmcp.homeunix.com/blog/tag/pkg-depotd/
- http://www.cuddletech.com/blog/pivot/entry.php?id=1127
- https://wikis.oracle.com/display/IpsBestPractices/Setting+Up+and+Maintaining+Package+Repositories

---

rsync the package repository updates