Building with oi-userland

Contents

1 Using OpenIndiana's unified build system

OpenIndiana Hipster's primary build framework is oi-userland. It's tied into OpenIndiana continuous integration platform.

When an update is committed to the oi-userland git repository:

- an automated build is kicked off,
- then automatically the binary package will be published to the /hipster repository,
- finally, the status of the build will be reported by oibot to the #oi-dev libera.chat IRC channel.

1.1 Overview of oi-userland

Originally oi-userland is a fork of Oracle's userland-gate which evolved in an independent way. The layout is very similar.

Inside oi-userland is a directory called "components", under which directories for software package category groups live. Inside each of these package group directories are typically directories for all software packages belonging to the category. Some have even more sub-directories before reaching the leaf directories containing the software packages. Note that some software packages don't have a package category group yet. Inside each of the software package directories there is a main Makefile and other files necessary to build one or more packages. The complete set of instructions to build one or more related packages is called "a component".

1.1.1 A component usually consists of several files:

- `Makefile`: the recipe to build the software and install it locally (usually to the `build/prototype/$(MACH)` directory)
- `patches/`: directory containing patches applied before the configuration
- `files/`: directory containing additional files distributed with packages
- `*.p5m`: manifests used to generate the IPS package
- `$(COMPONENT_NAME).license`: file containing the licenses applicable to the software

To build a component you simply cd into the directory of the software, and type "gmake TARGET" (here we use gmake to call GNU make), where TARGET can be one of:

Target	Description
`clobber`	cleans up the component directory completely, including deleting source
`download`	fetches the source archive and verify its `SHA256` sum
`prep`	extract and apply patches
`build`	configure and build
`install`	install software into the prototype directory
`sample-manifest`	generate a sample IPS manifest based on the files installed to the prototype directory
`publish`	publish the package to the local repository

Target	Description
`pre-publish`	run all pre-publication checks (does actually what publish does, just without sending package to local repository)
`REQUIRED_PACKAGES`	guess and generate build dependencies for the packages, manual edit might be needed
`env-check`	check build environment for missing packages
`env-prep`	install missing build dependencies (requires elevated privileges)

> ⓘ **NOTE:**
> **Before adding new packages to oi-userland…**
> Before considering adding a new package to oi-userland, please check first whether someone else is working on the package by checking the issue tracker, mailing oi-dev@openindiana.org or asking on the IRC (#oi-dev at irc.libera.chat)
> - If you don't find anyone already working on a port, please register your effort by opening an issue.
> - If you wish to update an existing port, look at the log for the component Makefile ("git log Makefile") and make sure you either contact the person who last updated the Makefile or include them on notifications for the issue by ticking their name.
>
> This will ensure efforts aren't duplicated and help to ensure sanity and comity amongst project members.

1.2 Setting up your build environment

We strongly recommend building packages inside a fresh local zone set up exclusively for building. See Quick zone setup example for simple instructions.

Further we assume that you are logged into the build zone if you set up the build environment in a zone the directory were oi-userland is cloned can otherwise be anywhere you like.

1.2.1 Adding RBAC profile to the build user

Installing software requires privileges, so your build user must have at minimum the 'Software Installation' profile:

```
$ profiles
Software Installation
ZFS File System Management
Console User
Suspend To RAM
Suspend To Disk
Brightness
CPU Power Management
Network Autoconf User
Desktop Removable Media User
```

```
Basic Solaris User
All
```

If it is not the case add this profile to your build user:

```
pfexec su -
usermod -P'Software Installation' <username>
```

This is not necessary if your user has already the ‘Primary Administrator’ profile.

1.2.2 Downloading oi-userland

Start by forking oi-userland repository on Github and then check out the repository (subdirectory oi-userland must not pre-exist):

```
cd ~
git clone https://github.com/mylogin/oi-userland
cd oi-userland
```

Add https://github.com/OpenIndiana/oi-userland/ as upstream to your repository to resync your repository with oi-userland.

```
git remote add upstream https://github.com/OpenIndiana/oi-userland/
```

Run the setup stage which will prepare some tools and create an IPS pkg5 repository for first use under the i386 directory:

```
cd $HOME/oi-userland
gmake setup
```

1.2.3 Adding the local repository to your publisher list

```
pfexec pkg set-publisher -g file://$HOME/oi-userland/i386/repo userland
pfexec pkg set-publisher --non-sticky openindiana.org
```

1.2.4 Setting the local repository as primary publisher

```
pfexec pkg set-publisher --search-first userland
```

1.2.5 Optional: Running a local pkg server for installation on other zones/hosts

If you would like to use your oi-userland repository on other zones or hosts, you can run a pkg server:

```
$ pfexec svccfg -s  pkg/server
svc:/application/pkg/server> add oi-userland
svc:/application/pkg/server> select oi-userland
svc:/application/pkg/server:oi-userland> addpg pkg application
svc:/application/pkg/server:oi-userland> setprop
 ↪  pkg/inst_root=astring:"/export/home/username/oi-userland/i386/repo"
svc:/application/pkg/server:oi-userland> setprop pkg/port=count:"10000"
svc:/application/pkg/server:oi-userland> setprop pkg/readonly=boolean:"true"
svc:/application/pkg/server:oi-userland> refresh
```

```
svc:/application/pkg/server:oi-userland> exit
$ pfexec svcadm enable oi-userland
```

On other hosts you can then specify http://hostname:10000 instead of the file:// address above. If you only intend to install and test packages locally this is not necessary as on-disk repository access suffices.

1.3 Building and installing your first packages

Enter to oi-userland/components/PATH/TO/COMPONENT directory and run:

```
cd $HOME/oi-userland/components/SOFTWARE
gmake env-prep
gmake publish
pfexec pkg refresh
pfexec pkg install pkg://userland/PACKAGE_FMRI
```

Here PACKAGE_FMRI is a full name (FMRI) of package which you want to install. The FMRIs of published packages will be printed in the end of `publish` stage. Note, that running `gmake env-prep` is strictly not required, if you are sure that all build requirements are satisfied.

To speed up the compilation you can pass an optional argument to gmake setting COMPONENT_BUILD_ARGS variable in your environment, for instance with

```
export COMPONENT_BUILD_ARGS=-j4
```

to use 4 jobs for builds.

2 Creating your first component

The easiest way to create new component is to take one which is similar to yours and modify it as needed. Also you can look at Makefile templates delivered with oi-userland.

2.1 Creating Makefile

A component Makefile usually contains variables describing how a component should be built, installed and packaged. On the top of the Makefile it includes `../../../make-rules/shared-macros.mk` where `../../../` is the relative path from the component directory to the make-rules directory. The file shared-macros.mk contains global constants used by other makefiles. Sometimes some global variables that alter these constants are declared before this include. The first section of the Makefile contains definitions of the component name, version, an url where the software should be fetched from, a short description embedded in the package metadata and so on. Look, for example, at `library/libjpeg6-ijg/Makefile`:

```
COMPONENT_NAME=         libjpeg6-ijg
COMPONENT_VERSION=      6.0.2
LIBJPEG_API_VERSION=    6b
COMPONENT_FMRI=         image/library/libjpeg6-ijg
COMPONENT_CLASSIFICATION=System/Multimedia Libraries
COMPONENT_PROJECT_URL=  http://www.ijg.org/
COMPONENT_SUMMARY=      libjpeg - Independent JPEG Group library version 6b
```

```
COMPONENT_SRC=          jpeg-$(LIBJPEG_API_VERSION)
COMPONENT_ARCHIVE=      $(COMPONENT_NAME)-$(COMPONENT_VERSION).tar.gz
COMPONENT_ARCHIVE_HASH= \
 sha256:75c3ec241e9996504fe02a9ed4d12f16b74ade713972f3db9e65ce95cd27e35d
COMPONENT_ARCHIVE_URL=
 ↪  http://www.ijg.org/files/jpegsrc.v$(LIBJPEG_API_VERSION).tar.gz
COMPONENT_LICENSE=      IJG,GPLv2.0
COMPONENT_LICENSE_FILE= $(COMPONENT_NAME).license
```

Here

Variable	Value	Comment
COMPONENT_NAME	libjpeg6-ijg	The name of the component, usually it's a well-known software name
COMPONENT_VERSION	6.0.2	The software version. If the version contains letters, the IPS_COMPONENT_VERSION variable should define a numerical version used for the package, as the IPS version string doesn't allow for letters
LIBJPEG_API_VERSION	6b	In this example this is a local variable declared in the Makefile.
COMPONENT_FMRI	image/library/libjpeg6-ijg	This variable can be used in an IPS manifest to specify the FMRI (a name) of the package. It should folllow the conventions for package FMRIs.
COMPONENT_CLASSIFICATION	System/Multimedia Libraries	This entry should be in the OpenSolaris IPS Classification 2008
COMPONENT_PROJECT_URL	http://www.ijg.org/	Upstream project website
COMPONENT_SUMMARY	libjpeg - Independent JPEG Group library version 6b	A short description (one-liner)
COMPONENT_SRC	`jpeg-$(LIBJPEG_API_VERSION)`	The name of source after unpacking the archive
COMPONENT_ARCHIVE	`$(COMPONENT_NAME)-$(COMPONENT_VERSION).tar.gz`	The software archive
COMPONENT_ARCHIVE_HASH	sha256:75c3ec241e9996504fe02a9ed4d12f16b74ade713972f3db9e65	The SHA256 checksum of the software archive
COMPONENT_ARCHIVE_URL	`http://www.ijg.org/files/jpegsrc.v$(LIBJPEG_API_VERSION).tar.gz`	The URL to get the COMPONENT_ARCHIVE
COMPONENT_LICENSE	IJG,GPLv2.0	A comma separated list of licenses
COMPONENT_LICENSE_FILE	`$(COMPONENT_NAME).license`	The file with license text

Components are usually based on one of the following Makefiles depending on build system used by packaged software (look in the `make-rules` directory for more makefiles):

File	Build
`ant.mk`	Ant
`attpackagemake.mk`	AT&T package tools
`cmake.mk`	CMake
`configure.mk`	Autotools
`gem.mk`	Ruby
`justmake.mk`	plain Makefile
`makemaker.mk`	Perl
`setup.py`	Python distutils

Read the `.mk` file to see which variables you can modify, in general you can find variables such as:

- `*_ENV`
- `*_OPTIONS`
- `PRE_*_ACTION`
- `POST_*_ACTION`

For example, you may add this line for an Autotools-based component:

```
CONFIGURE_OPTIONS+= --enable-shared
```

After creating the component Makefile you can run `gmake prep`.

Now you can create necessary patches for the component and put them in the patches directory.

When the component is built and installed correctly (via `gmake build` and `gmake install`), look if you can run the test suite if one comes with the software.

It's advised to put the expected test ouput in test/results-BITS.master (where BITS are either 32 or 64) and to ensure that the `gmake test` target generates reproducible results. You can use the COMPONENT_TEST_TRANSFORMS variable to set a list of sed directives to transform the test output and make it reproducible.

2.2 Creating p5m files

When the `install` target passes you can run:

```
gmake sample-manifest
```

to generate a manifest from the list of installed files.

Copy the file `manifests/sample-manifest.p5m` to `$(COMPONENT_NAME).p5m` and edit it:

- Add your name as a contributor
- Remove unused entries from the manifest:
- directories: `:%g/^dir/d` (Vim)
- static libs: `:%g/.a$/d` (Vim)
- libtool files: `:%g/.la$/d` (Vim)

- Python *.pyc: `:%g/.pyc$/d` (Vim)

For some components, specific rules need to be applied: they can be implemented with *transforms*. Some Makefile targets defined in `make-rules/ips.mk` apply transforms from files in the `transforms` directory at the root of oi-userland. These transforms can be used as examples if you need custom transforms for your component.

2.3 Publishing packages

After creating a p5m file run `gmake REQUIRED_PACKAGES` to automatically generate a list of run-time dependencies of the package (REQUIRED_PACKAGES section of the Makefile).

Add necessary build time dependencies on the top of the generated section.

Run `gmake publish`. If the manifest is valid your package is published to the local repository.

To be able to search for the new packages in the local repository you need to rebuild search indexes:

```
pkgrepo refresh -s /path/to/my_repo
```

You can even rebuild the entire metadata:

```
pkgrepo rebuild -s /path/to/my_repo
```

2.4 Installing the package

After you've published the package to your local repository and rebuilt the repository index or metadata you can install the package and perform whatever testing is appropriate.

```
pkg publisher
```

If the package you built and published to the local userland repository is not already part of hipster it should be straightforward to install it:

```
pfexec pkg install your/package/name
```

If, however, the package you built is an updated version of an existing package then you may have to take additional steps before it can be updated.

If `pkg` refuses to install the package from your local repository it may be because the userland-incorporation is preventing updates to the version of the package:

```
$ pfexec pkg update image/library/libjpeg6-ijg
No updates available for this image.
$ pfexec pkg update
 ↪  pkg://userland/image/library/libjpeg6-ijg@6.0.2-2018.0.0.1:20180211T125627Z
pkg update: No matching version of image/library/libjpeg6-ijg can be installed:
  Reject:  pkg://userland/image/library/libjpeg6-ijg@6.0.2-2018.0.0.1
  Reason:  This version is excluded by installed incorporation
   ↪  consolidation/userland/userland-incorporation@0.5.11-2018.0.0.11745
```

If you will install many test versions of packages on your development system you may find it easiest to uninstall the userland-incorporation. Alternately, if you want to test a package on a system while keeping userland-incorporation, you can use `pkg change-facet` to relax the version constraint for just that package:

```
pfexec pkg change-facet facet.version-lock.your/package/name/here=false
```

After you have performed one of these steps to remove the version constraint there is one more issue you may encounter. Because the installed version of the package came from the openindiana.org publisher but the updated version you want to install and test is associated with the userland publisher, `pkg` will by default not allow the package update to switch which publisher provides the package.

One option to work around this is to make the openindiana.org publisher non-sticky:

```
pfexec pkg set-publisher --non-sticky openindiana.org
```

You only need to perform that operation on your development system once.

Alternately, you can force `pkg` to apply an update from a different publisher by specifying the full FMRI for the package, including the publisher:

```
pfexec pkg update
 ↪  pkg://userland/image/library/libjpeg6-ijg@6.0.2-2018.0.0.1:20180211T125627Z
```

3 Contributing changes back to oi-userland

3.1 Every time you add or modify a component, create a new branch:

```
git checkout -b my_feature
```

3.2 Keep this branch synchronized with upstream/oi/hipster:

```
git pull --rebase upstream oi/hipster
```

Your local branch is forwarded to the last commit of oi/hipster and your additional commits are kept on top of the stack.

3.3 Committing changes

When you think you are ready with changes you need to commit them locally and push those changes back to your Github repository.

```
cd ~/oi-userland/components/SOFTWARE
git add Makefile *.p5m <etc.>
git commit
```

Commit messages should be simple and describe what you did, e.g “Added XYZ”, “XYZ: updated to SOME_VERSION”, “XYZ: fixed SOMETHING” or “BUG_ID BUG_SUMMARY”, where BUG_ID is issue number from issue tracker and BUG_SUMMARY is the issue name.

If you’ve created several commits while working on your component it’s necessary to *squash* all your commits into one.

To do it, first check how many commits are to be considered:

```
git log
```

then

```
git rebase -i HEAD~N
```

with N the number of commits to be squashed and follow the instructions: the letter ‘s’ should be put in place of ‘pick’ for the N - 1 commits before the last.

If you made a mistake with the commit message or author, use:

```
git commit --amend
```

with the relevant option.

Now push your changes to your repository to GitHub.

```
git push my_name my_feature
```

or

```
git push -f my_name my_feature
```

if the branch you just rebased had already been pushed: since the history is rewritten you need to force the push, be careful.

3.4 Asking for change integration

This is as simple as creating a Pull Request into the main oi-userland repository and asking developers to review your changeset. We should beware of possibly breaking packages as it adds additional work and can be unpleasant for other contributors (imagine a situation where gcc, perl or anything else needed for building packages is broken).

Changes can be reverted quite easily but once the package is built and published additional steps are needed. So try taking per-package testing and asking for wider testing into consideration.

If you contribute a package which is known to work but its functionality might be broken because of some issues, consider disabling it until the issue is removed.

3.5 Checking Jenkins instance

Once the changes are merged into the main oi-userland repository, the Jenkins instance will pick up those bits and build them. If the build was successful, the built packages will be pushed into http://pkg.openindiana.org/hipster repository. If the package build was unsuccessful check the build logs and please try to come up with a solution and fix the problem so you can have the package published into the repository.